



**PlateMate**

# PlateMate

*Taste the Difference*

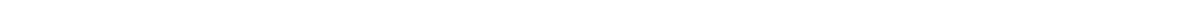


## **Csoporttagok:**

Barta Máté (Csapatvezető)

Gere Csanád

Bene Dominik



# Tartalomjegyzék

Tartalomjegyzék.....	2
I. A szoftver célja .....	4
Kinek és miért? .....	4
II. Fejlesztői dokumentáció.....	6
Fejlesztőkörnyezet .....	6
Használt IDE-k:.....	6
Technikai Információk .....	8
Adatszerkezetek.....	10
Adatbázis dokumentációja .....	11
ER (Entity Relationship) Diagramm .....	13
Egyedi algoritmusok.....	14
authController.js.....	15
Szerver oldali komponens .....	19
Tesztdokumentáció .....	21
openingHourController.test.js.....	21
authController.test.js .....	26
Fejlesztési lehetőségek.....	30
III. Felhasználói dokumentáció .....	31
A program funkciói .....	31
Szükséges technikai eszközök .....	32
Technikai adatok Windows PC-hez.....	32
Telepítés és indítás .....	33
Adatbázis telepítése és futtatása .....	33
Frontend és Backend telepítése és futtatása .....	35
A program bemutatása .....	38
Az alkalmazás főbb részei .....	38
Első beléptetés az alkalmazásba .....	44
Helytelen használat esetei.....	44
Hibajelzések .....	45
Információkérés .....	46
IV. Összefoglalás, köszönetnyilvánítás .....	47



---

V. Irodalomjegyzék.....	49
Linkek.....	49

## I. A szoftver célja

A PlateMate egy innovatív megoldás, amely segíti az éttermeket a modernizálásban és a digitális átalakulásban. A rendszer célja, hogy egyszerűsítse és optimalizálja az éttermi működést, miközben kényelmes és gördülékeny élményt nyújt a vendégek számára. A szoftver segítségével az éttermek lépést tarthatnak a modern fogyasztói elvárásokkal, minimalizálva a manuális folyamatokat és csökkentve a hibalehetőségeket.

A PlateMate lehetővé teszi a gyorsabb és hatékonyabb kiszolgálást, csökkentve a várakozási időt és növelve az ügyfél elégedettségét. Az intelligens rendeléskezelési rendszernek köszönhetően a személyzet könnyebben és átláthatóbban tudja kezelni a rendeléseket, amely gyorsabb kiszolgálást és precízebb munkavégzést eredményez. Az automatizált folyamatok révén csökkenthető a személyzeti túlterheltség, és optimalizálható a munkafolyamatok hatékonysága.

A szoftver egy felhasználóbarát, akár mobilról is elérhető kezelőfelületet biztosít, amely lehetővé teszi a tulajdonosok és alkalmazottak számára, hogy a nap bármely pillanatában nyomon kövessék és módosítsák az éttermi működéshez kapcsolódó adatokat. Az élő frissítéseknek köszönhetően az étlapon vagy a foglalások kezelésében végzett módosítások azonnal érvénybe lépnek, megkönnyítve az üzleti folyamatok irányítását. A PlateMate telepítés után gyorsan és hatékonyan integrálható az étterem mindennapi működésébe, lehetővé téve a testreszabott beállításokat és a folyamatos fejlesztéseket.

---

## Kinek és miért?

Azért döntöttünk a PlateMate megvalósítása mellett, mivel nagyon jól tudjuk, hogy rengeteg étterem még a mai napig papíron vezeti az éttermet, és papírra írják fel a rendeléseket. Ezt szeretnénk most megváltoztatni.

A PlateMate minden olyan étterem számára ajánlott, amely jelenleg nem rendelkezik digitális megoldásokkal, vagy elavult rendszereket használ. Az étterem tulajdonosok számára a szoftver lehetőséget nyújt a működés optimalizálására, az



asztalfoglalások és rendelések automatizálására, valamint az ügyfélkapcsolatok hatékonyabb kezelésére. Ezáltal jelentős idő- és erőforrás-megtakarítást eredményez, miközben növeli az ügyfélélményt.

A rendszer előnyei az alkalmazottak számára is jelentősek. A rendelési folyamatok könnyebben követhetők, csökkentve az emberi hibák esélyét, és gyorsabb, pontosabb kiszolgálást biztosítva. Az étterem vezetősége könnyedén ellenőrizheti az üzlet forgalmát, a népszerűbb ételeket és az ügyfélpreferenciákat, így jobban alakíthatja az üzleti stratégiát és növelheti a profitabilitást.

A PlateMate a vendégek számára is számos előnnyel jár. Az okostelefonról elérhető funkciók lehetővé teszik az előzetes rendeléseket, gyors asztalfoglalásokat és az étlap böngészését, így az étkezési élmény gördülékenyebbé és kényelmesebbé válik. Az automatizált fizetési rendszer csökkenti a várakozási időt, és lehetőséget biztosít az érintésmentes fizetésre, amely különösen fontos a modern digitális korszakban.

A PlateMate nemcsak egy egyszerű rendeléskezelő rendszer, hanem egy átfogó, digitális megoldás, amely elősegíti az éttermek versenyképességének növelését, a hatékonyság javítását és az ügyfélélmény fejlesztését. A technológia alkalmazásával az éttermek képesek megfelelni a folyamatosan változó piaci igényeknek, és hosszú távon is sikeresen működni a vendéglátóiparban.

## II. Fejlesztői dokumentáció

A fejlesztői dokumentáció magába foglalja az összes eszközt, ami használva lett a projekt megvalósításához, illetve leírja: a program telepítését, annak futási környezetét, a kialakított adatbázis és backend adatszerkezeteket, bemutat néhány metódust a programból, leírja a program tesztelését, és kínál néhány fejlesztési lehetőséget a programhoz, amit a későbbiekben meg lehet még valósítani a program bővítéséhez.

### Fejlesztőkörnyezet

A fejlesztőkörnyezet tartalmazza azon eszközöket, programokat, struktúrákat, amelyek használva voltak a PlateMate megírásához, és így hozzájárultak/hozzájárulnak a program kiváló működéséhez. Ezen ponton szeretnénk megjegyezni, hogy ***a PlateMate csapata semmilyen felelősséget nem vállal a program értékelése során tapasztalt olyan problémákért, hibákért, amelyek a program ellenőrzése közben abból fakadnak***, hogy:

- Az applikáció komponenseit nem a dokumentációban megadott verzióval futtatják a programot (pl: MySQL, Xampp, NPM, NODE, VUE, TailwindCSS, Express) Ezek csupán példák az applikáció komponenseire, a részletes komponensek listáját lejjebb találják.
- Hibásan, nem megfelelően, vagy elavult applikáció verziót töltöttek le a megadott GitHub Repository-ből.
- Hibásan, vagy nem megfelelően lettek beimportálva a megadott adatok, file-ok.
- Hibásan, vagy nem megfelelően lett felépítve a .env file.
- Másik operációs rendszer verzió próbálták meg futtatni a programot.
- Másik operációs rendszeren próbálták meg futtatni a programot.
- Nem MySQL-t használtak Xampp-on keresztül az adatbázis futtatásához.

### Használt IDE-k:

- **Visual Studio Code (VS Code)**

A Visual Studio Code (VS Code) egy ingyenes, nyílt forráskódú, könnyű, mégis erőteljes fejlesztői környezet, amelyet a Microsoft fejlesztett. A VS Code támogatja a JavaScript, Vue.js, Node.js és számos más programozási nyelv fejlesztését, így tökéletes

választás egy full stack alkalmazás létrehozásához. Az intelligens kódkiegészítés (IntelliSense), a beépített Git-integráció és a terminál támogatás lehetővé teszik a fejlesztési folyamat gyorsítását és hatékony kezelését. Az eszköz támogatja a könnyen telepíthető kiegészítőket (extensions), amelyek lehetővé teszik a fejlesztők számára a környezet testre szabását, például a Vue.js DevTools vagy az ESLint használatával.

A VS Code emellett integrált debugging eszközzel is rendelkezik, amely lehetővé teszi a Node.js és a böngésző alapú hibakeresést. A Live Share funkció lehetővé teszi a valós idejű együttműködést, ami különösen hasznos csapatmunkánál. A könnyen kezelhető felület és a gyors teljesítmény miatt a VS Code az egyik legnépszerűbb fejlesztői eszköz, amely jól működik bármilyen operációs rendszeren, beleértve a Windows, macOS és Linux rendszereket is.

- **JetBrains WebStorm**

A WebStorm egy prémium fejlesztői környezet, amelyet kifejezetten a modern JavaScript-alapú fejlesztéshez terveztek. A JetBrains által fejlesztett WebStorm mély integrációt kínál a Vue.js, Node.js, Express és más népszerű webes technológiák számára. Az intelligens kódkiegészítés, refaktorálási lehetőségek és fejlett hibakereső eszközök révén a WebStorm különösen hasznos nagyobb, komplex projektek esetében, ahol a kód minősége és karbantarthatósága kulcsfontosságú. Az IDE beépített támogatást nyújt a verziókezelő rendszerekhez, például Git, GitHub és GitLab, ami megkönnyíti a csapatmunkát és a verziókezelést.

A WebStorm egyik kiemelkedő előnye a beépített tesztelési támogatás, amely lehetővé teszi az egység- és integrációs tesztek futtatását közvetlenül a fejlesztői környezetből. A beépített REST kliens és adatbázis-kezelő eszközök segítenek a szerveroldali fejlesztések hatékony tesztelésében. Habár a WebStorm fizetős szoftver, sok fejlesztő számára megéri a befektetést a produktivitás növelése érdekében, különösen akkor, ha napi szinten dolgoznak JavaScript és TypeScript alapú alkalmazásokon.

## Technikai Információk

### Operációs Rendszer: Windows 10/11

A fejlesztésre a Windows 10 és 11 operációs rendszerek használata ideális választás, mivel széles körben támogatottak és kompatibilisek a legtöbb modern fejlesztői eszközzel. A Windows környezet lehetőséget biztosít a fejlesztők számára, hogy könnyedén telepítsenek és konfiguráljanak különböző futtatókörnyezeteket, mint például a Node.js, MySQL és XAMPP. Továbbá, olyan népszerű fejlesztői eszközök is rendelkezésre állnak, mint a Visual Studio Code, amely beépített támogatást nyújt a JavaScript és a Vue.js számára, valamint a Webstorm. A Windows támogatja a WSL (Windows Subsystem for Linux) használatát is, amely lehetővé teszi a fejlesztők számára, hogy Linux-alapú parancssori eszközöket használjanak egy Windows rendszeren belül. A széleskörű hardver- és szoftverkompatibilitás miatt a Windows megbízható választás minden fejlesztési szakaszban.

### Front End

A felhasználói felület fejlesztése során az alábbi technológiákat használjuk:

- **HTML** – A HyperText Markup Language (HTML) az alkalmazás szerkezeti alapját biztosítja. Lehetővé teszi az oldal különböző elemeinek meghatározását és a felhasználói interfész felépítését. Az HTML folyamatosan fejlődő szabványai biztosítják a kompatibilitást a modern böngészőkkel, valamint támogatják az akadálymentességet és a keresőmotor-optimalizálást (SEO). Az alaposan strukturált HTML-kód hozzájárul a könnyebb karbantarthatósághoz és a responszív dizájn kialakításához.
- **CSS** – A Cascading Style Sheets (CSS) segítségével formázzuk az oldal kinézetét, biztosítva a modern, responszív és esztétikus megjelenést. A CSS lehetővé teszi az animációk és interaktív elemek beépítését, növelve ezzel a felhasználói élményt. Az előre definiált stílusok és CSS preprocessorok, mint például a SASS vagy LESS, tovább javíthatják a kód szervezhetőségét és hatékonyságát.
- **Vue.js (v5.0.8)** – A Vue.js egy könnyen tanulható és hatékony JavaScript keretrendszer, amely segít a felhasználói interfész dinamikus kezelésében. A Vue



5.0.8 verzió számos teljesítménybeli fejlesztést tartalmaz, valamint támogatja a legújabb modern webes szabványokat. A Vue ökoszisztémája folyamatosan bővül, így könnyedén integrálható Vuex vagy Pinia állapotkezelő, valamint Vue Router az útvonalkezeléshez. A reaktív adatkezelés és a komponensalapú architektúra segít a fejlesztőknek hatékony és jól skálázható alkalmazásokat létrehozni.

- **Tailwind CSS (v3.4.13)** – Egy utility-first CSS keretrendszer, amely lehetővé teszi az egyedi stílusok gyors alkalmazását anélkül, hogy külső CSS fájlokat kellene írni. A Tailwind 3.4.13 verziója optimalizált teljesítményt és testre szabható konfigurációs lehetőségeket kínál. Az előre definiált utility osztályok használatával gyorsítható a fejlesztés, miközben a kód továbbra is tiszta és jól olvasható marad. A Tailwind segítségével könnyedén valósíthatók meg reszponzív és modern webes felületek anélkül, hogy hosszadalmas egyedi CSS szabályokat kellene írni.
- **NPM (v10.9.0)** – A Node Package Manager (NPM) egy csomagkezelő, amely lehetővé teszi a szükséges front-end függőségek kezelését és telepítését. Az NPM segítségével könnyedén integrálhatók külső könyvtárak és eszközök a fejlesztési környezetbe. A gyors verziókezelésnek köszönhetően mindig naprakészen tarthatók a projekt függőségei, minimalizálva a kompatibilitási problémákat.

## Back End

A szerveroldali logika és az API-k fejlesztése a következő technológiákkal történik:

- **JavaScript** – A teljes stack fejlesztést megkönnyítve egyetlen programozási nyelvet használunk mind a front-end, mind a back-end fejlesztésére. A JavaScript dinamikus, könnyen skálázható, és széles körű támogatással rendelkezik. A Node.js környezetben futtatva lehetővé teszi aszinkron műveletek gyors végrehajtását, amely jelentős teljesítménybeli előnyt nyújt a hagyományos szinkron rendszerekkel szemben.
- **Node.js (v20.17.0)** – A Node.js egy gyors, aszinkron működésű JavaScript futtatókörnyezet, amely lehetővé teszi szerveroldali alkalmazások fejlesztését. A 20.17.0 verzió fejlett teljesítményoptimalizálásokat és stabilitást biztosít. A V8

motorra épülve hatékonyan kezeli a nagy terhelésű, real-time adatfeldolgozási feladatokat, így ideális választás webserverekhez és API-khoz.

- **Express.js (v4.21.1)** – Az Express.js egy minimalista és rugalmas Node.js keretrendszer, amely segít az API-k és szerveroldali alkalmazások gyors fejlesztésében. A 4.21.1 verzió biztosítja a legújabb hibajavításokat és kompatibilitást a Node.js újabb verzióival. Az Express könnyen integrálható middleware-ekkel és adatbázis-kezelőkkel, ami tovább növeli a fejlesztés hatékonyságát.
- **NPM (v10.9.0)** – A backend függőségek kezelése és automatizálása NPM segítségével történik, lehetővé téve az Express.js és egyéb csomagok egyszerű telepítését és frissítését. Az NPM CLI (parancssoros felület) segítségével gyorsan konfigurálhatók a projektek, és biztosítható a stabil működés.

## Adatbázis

Az alkalmazás adatkezelése és tárolása MySQL alapú adatbázis segítségével történik:

- **XAMPP Control Panel (v3.3.0)** – Egy könnyen használható helyi fejlesztői környezet, amely tartalmazza az Apache webszervert és a MySQL/MariaDB adatbáziskezelőt. A XAMPP használata gyors beállítást és kényelmes adatbázis-kezelést biztosít. A lokális fejlesztési környezet lehetővé teszi az adatbázis-struktúra gyors tesztelését és módosítását, mielőtt az éles szerverre kerülne.
- **MySQL (10.4.32-MariaDB)** – A MariaDB a MySQL egyik legnépszerűbb nyílt forráskódú változata, amely kiváló teljesítményt, megbízhatóságot és skálázhatóságot kínál. A 10.4.32 verzió stabil és optimalizált működést biztosít az adatok gyors és biztonságos kezelésére. Az ACID kompatibilitásnak és fejlett indexelési mechanizmusoknak köszönhetően hatékonyan kezeli a nagy mennyiségű adatot és összetett lekérdezéseket.

## Adatszerkezetek

Ebben a fejezetben az applikációhoz használt adatszerkezetek kerülnek bemutatásra.



## Adatbázis dokumentációja

A vizsgaremek adatbázis az éttermi rendelési rendszer támogatására készült. A rendszer a rendelések, fizetések, foglalások és felhasználók kezelésére szolgál. Az alábbiakban részletezzük az adatbázis szerkezetét és a benne található táblákat.

---

### Táblák és oszlopok

#### 1. category (**Kategóriák**)

Tárolja az ételek kategóriáit (pl. előétel, főétel, desszert).

- id (int): Egyedi azonosító.
- (255): A kategória neve.

---

#### 2. item (**Ételek és italok**)

Tárolja az étlapon szereplő tételeket.

- id (int): Egyedi azonosító.
- name (varchar (255)): Az étel vagy ital neve.
- price (int): Az étel vagy ital ára.
- categoryId (int): A kategória azonosítója (category.id kapcsolódó mező).

---

#### 3. openinghours (**Nyitvatartás**)

Tárolja az étterem nyitvatartási adatait.

- id (int): Egyedi azonosító.
- dayName (varchar (255)): A hét napja.
- fromHour (varchar (255)): Nyitási idő.
- untilHour (varchar (255)): Zárási idő.

---

#### 4. orders (**Rendelések**)

Tárolja a leadott rendeléseket.

- id (int): Egyedi azonosító.
- tableId (int): Asztal azonosítója (tables.id kapcsolódó mező).
- itemId (int): Rendelési tétel azonosítója (item.id kapcsolódó mező).
- isDone (tinyint (1)): Kész van-e a rendelés (0 = nem, 1 = igen).
- isServed (tinyint (1)): Kiszolgálták-e a rendelést (0 = nem, 1 = igen).
- orderedAt (datetime): Rendelés időpontja.

---

#### 5. paid (**Kifizetések**)

Tárolja a kifizetett rendeléseket.

- id (int): Egyedi azonosító.



- tableId (int): Asztal azonosítója.
  - itemId (int): Rendelési tétel azonosítója.
  - paymentMethodId (int): Fizetési mód azonosítója (paymentmethods.id kapcsolódó mező).
  - paidAt (datetime): Kifizetés időpontja.
- 

## 6. paymentmethods (Fizetési módok)

Tárolja az elérhető fizetési módokat.

- id (int): Egyedi azonosító.
  - name (varchar (255)): Fizetési mód neve (pl. készpénz, bankkártya).
- 

## 7. permissionsettings (Jogosultságok)

Tárolja a különböző jogosultsági szinteket.

- id (int): Egyedi azonosító.
  - section (varchar (255)): Jogosultsági szekció neve.
- 

## 8. reservedtable (Foglalások)

Tárolja az asztalfoglalásokat.

- id (int): Egyedi azonosító.
  - name (varchar (255)): Foglalo neve.
  - numberOfCustomers (int): Foglalásban részt vevő vendégek száma.
  - tableId (int): Asztal azonosítója.
  - reservedAt (datetime): Foglalás időpontja.
  - reservedUntil (datetime): Foglalás lejáratási ideje.
- 

## 9. sessions (Munkamenetek)

Tárolja a felhasználók aktív munkameneteit.

- id (varchar (255)): Egyedi azonosító.
  - userId (int): Felhasználó azonosítója (user.id kapcsolódó mező).
  - ip (varchar (255)): Felhasználó IP címe.
  - expires (datetime): Munkamenet lejáratási ideje.
- 

## 10. tables (Asztalok)

Tárolja az étterem asztalait.

- id (int): Egyedi azonosító.
  - tableNumber (int): Asztal száma.
- 

## 11. user (Felhasználók)

Tárolja az éttermi rendszerben regisztrált felhasználókat.

- id (int): Egyedi azonosító.
- name (varchar (255)): Felhasználó neve.
- email (varchar (255)): Felhasználó e-mail címe.
- hashedPassword (varchar (255)): Jelszó hash-elve.
- permissionId (int): Jogosultsági szint (permissionsettings.id kapcsolódó mező).
- createdAt (datetime): Fiók létrehozásának időpontja.

---

### Kapcsolatok az adatbázisban

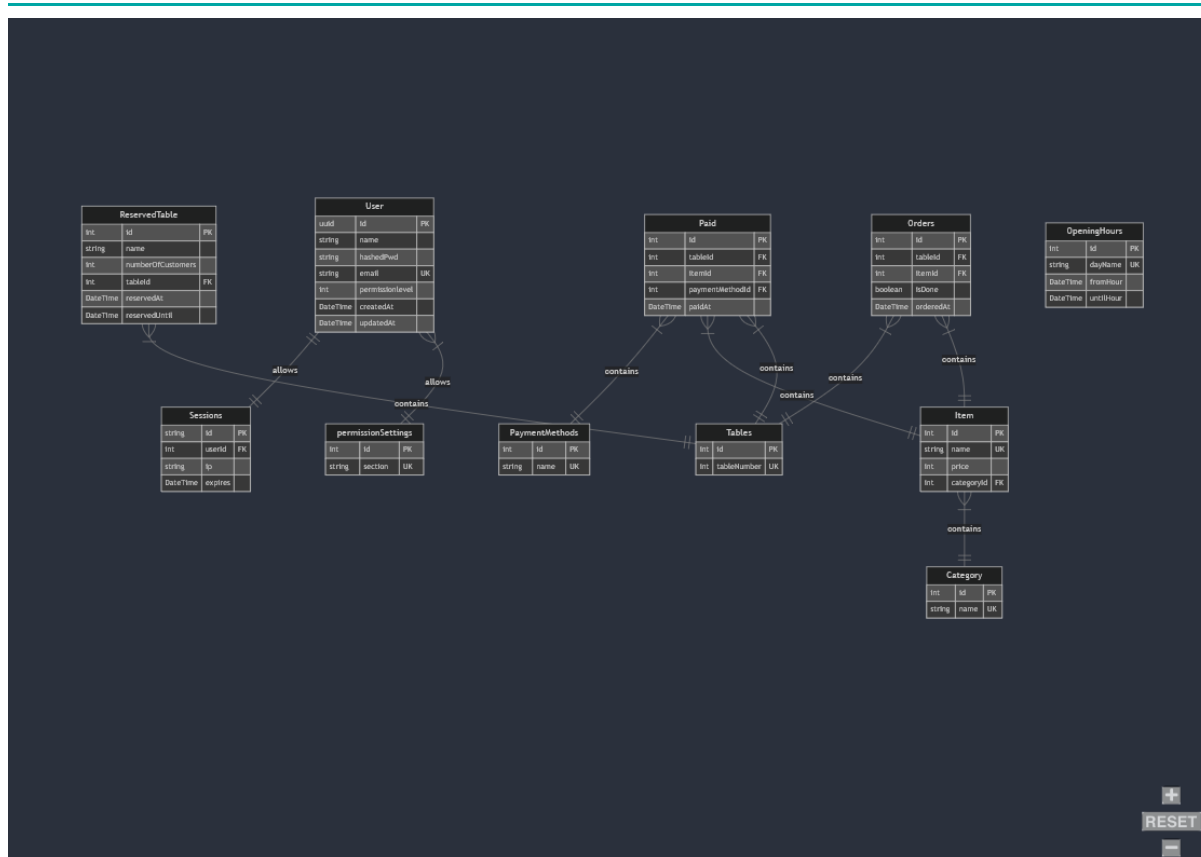
- Az item tábla a category táblára hivatkozik.
- Az orders és paid táblák az item, tables és paymentmethods táblákkal kapcsolódnak.
- A reservedtable és orders táblák az tables táblával állnak kapcsolatban.
- A user táblában lévő jogosultságokat a permissionsettings táblában tároljuk.

---

Ez az adatbázis-struktúra biztosítja az éttermi rendelési rendszer hatékony működését, lehetővé téve a rendelések, fizetések és foglalások pontos kezelését.

### ER (Entity Relationship) Diagramm

Ez az ER-diagram egy éttermi rendelési rendszert modellez. A **User** entitás kezeli a felhasználókat, jogosultságokkal és munkamenetekkel kapcsolódva. Az **Orders** és **Paid** táblák követik a rendeléseket és a fizetéseket, összekötve az **Item**, **Tables**, és **PaymentMethods** entitásokkal. Az **Item** kategóriákba sorolható, és a **ReservedTable** az asztalfoglalásokat kezeli. Az **OpeningHours** tárolja a nyitvatartást. Az adatmodell biztosítja az éttermi rendelési és foglalási folyamatok átlátható kezelését.



## Egyedi algoritmusok

Ebben a fejezetben egy-két érdekesebb, egyedi algoritmus van bemutatva. Ezek az algoritmusok szerves részét képezik az applikációnak, felépítésük a programozási alapelveknek megfelelően lett létrehozva, kialakítva, és implementálva. A továbbiakban képeket láthatnak a kódrészletről, illetve rövid leírást, hogy hogyan működik a program és mi a célja.

## authController.js

A **register** függvény egy új felhasználót hoz létre, ellenőrzi, hogy az e-mail cím már foglalt-e, majd a jelszót biztonságosan hashelve menti az adatbázisba.

```
const register = async (req, res) : Promise<...> => { Show usages ⓘ iwillsettyou
  const { name, email, password, permissionId } = req.body;

  if (!name || !email || !password || !permissionId) {
    return res.status(400).json({ message: "Minden mező megadása kötelező." });
  }

  const users :... = await User.findOne(req.body.email)

  if (users.data.length != 0) { res.status(409).send({ "message" : "Az email cím már foglalt." }) }

  else {
    const salt :any|void = await bcrypt.genSalt( rounds: 10);
    const hashedPassword = await bcrypt.hash(password, salt);

    const user :User = new User(name, hashedPassword, email, permissionId);

    try {
      const savedUser :... = await user.save();

      res.status(200).send(savedUser);
    } catch (err) {
      res.status(400).send({ message : "Hiba történt a mentés során", error : err});
    }
  }
};
```

A **login** függvény először ellenőrzi, hogy van-e érvényes munkamenet, és ha nincs, akkor a felhasználó hitelesítő adatait ellenőrzi, JSON Web Token generál számára, és azt sütikben tárolja.

```
const login = async (req, res) : Promise<...> => { Show usages ⓘ iwillsettyou
  const sessionCheck :... = await checkSession(req, res);

  const sessionId :string | Uint8Array = await uuidv4();

  if(sessionCheck.response) {
    const users :... = await User.findOne(req.body.email);
    const user = users.data[0]

    const jti :string | Uint8Array = await uuidv4();
    const token = jwt.sign(
      payload: {
        jti: jti,
        sub: user.email,
        iss: process.env.ISSUER,
        typ: "Bearer",
        preferred_username: user.name,
        sid: sessionId,
      },
      process.env.TOKEN_SECRET,
      options: {
        expiresIn: "3h",
      });

    const refreshToken = await generateRefreshToken(jti, user, sessionId);

    res.cookie("token", token, {
      httpOnly: true,
      secure: process.env.NODE_ENV === "production",
      maxAge: 3 * 60 * 60 * 1000,
    });

    res.cookie("refreshToken", refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === "production",
      maxAge: 30 * 24 * 60 * 60 * 1000,
    });

    res.send({ "message" : "Belépve érvényes munkamenet alapján." })
  } else {
```



```
const users :... = await User.findOne(req.body.email);
const user = users.data[0]

if (!user) return res.status(400).send({ message : "Érvénytelen email cím." });

const validPass = await bcrypt.compare(req.body.password, user.hashedPassword);

if (!validPass) return res.status(400).send({ message : "Érvénytelen jelszó." });

const jti :string| Uint8Array = await uuidv4();
const token = jwt.sign(
  payload: {
    jti: jti,
    sub: user.email,
    iss: process.env.ISSUER,
    typ: "Bearer",
    preferred_username: user.name,
    sid: sessionId,
  },
  process.env.TOKEN_SECRET,
  options: { expiresIn: "3h" }
);

try{
  const refreshToken = await generateRefreshToken(jti, user, sessionId);

  const expires :string = formatDateToMySQL( timestamp: Date.now() + 86400000);

  await uploadSession(req, sessionId, user.id, expires)

  res.cookie('connect.sid', `s:${sessionId}`, {
    httpOnly: true,
    maxAge: 86400000
  });

  res.cookie("token", token, {
    httpOnly: true,
    secure: process.env.NODE_ENV === "production",
    maxAge: 3 * 60 * 60 * 1000,
  });
}
```

```
res.cookie("refreshToken", refreshToken, {
  httpOnly: true,
  secure: process.env.NODE_ENV === "production",
  maxAge: 30 * 24 * 60 * 60 * 1000,
});

res.status(200).send({ "message" : "Belépve új munkamenettel." })
}
catch (error){
  res.status(500).send({ error : error })
}
}
};
```

A *logout* függvény törli a felhasználó munkamenetét, míg a *generateRefreshToken* egy új frissítő tokenet generál, amely lehetővé teszi a hozzáférési token megújítását anélkül, hogy újra be kellene jelentkezni.

```
const generateRefreshToken = async (jti, user, sid) : Promise<string> => {
  return jwt.sign(
    {
      payload: {
        iss: process.env.ISSUER,
        jti: jti,
        typ: "Refresh",
        sub: user.email,
        sid: sid,
      },
      process.env.TOKEN_SECRET,
      options: { expiresIn: "1h" }
    },
    process.env.TOKEN_SECRET,
    { expiresIn: "1h" }
  );
};
```

A *formatDateToMySQL* segédfüggvény az időbélyegeket MySQL-kompatibilis formátumba alakítja.

```
const formatDateToMySQL = (timestamp) : string => {
  const date : Date = new Date(timestamp);
  const year : number = date.getFullYear();
  const month : string = String(date.getMonth() + 1).padStart(2, '0');
  const day : string = String(date.getDate()).padStart(2, '0');
  const hours : string = String(date.getHours()).padStart(2, '0');
  const minutes : string = String(date.getMinutes()).padStart(2, '0');
  const seconds : string = String(date.getSeconds()).padStart(2, '0');

  return `${year}-${month}-${day} ${hours}:${minutes}:${seconds}`;
};
```

## Szerver oldali komponens

Az orderController.js file-ban található az alkalmazás szerver oldali komponense, amely EJs használatával készült. Nem sokkal tér el ennek a szerver oldali megvalósítása, viszont a kliens oldalon nagyobb változásokat kellett bevezetni, mint a többi, kliens oldali komponensünknél.

```
const getOrdersByTableId = async (req, res) : Promise<...> => { Show usages csan *
  const { id } = req.params;

  if (!id) {
    return res.status(400).send("Az ID megadása kötelező.");
  }

  try {
    const orders = await new Promise<(executor: (resolve, reject) : void => {
      connect.query(`
        SELECT
          o.id as id,
          o.tableId as tableId,
          o.itemId as itemId,
          o.orderedAt as orderedAt,
          i.id as itemId,
          i.name as itemName,
          i.price as itemPrice,
          t.tableNumber as tableNumber
        FROM
          orders o
          JOIN
            item i ON o.itemId = i.id
          JOIN
            tables t ON o.tableId = t.id
        WHERE
          t.id = ?
      `, [id], (err, result) : void => {
        if (err) reject(err);
        else resolve(result);
      });
    });

    if (!orders.length) {
      return res.render("orders", { message: "Nincsenek elérhető rendelések.", items: [] });
    }

    res.render("orders", { message: "Rendelések sikeresen lekérve.", items: orders });
  } catch (error) {
    res.status(500).render("orders", { message: "Hiba történt a rendelések lekérése során.", items: [] });
  }
};
```

A frontend-en: src/views/CashoutTable.ejs

```
<!DOCTYPE html>
<html lang="hu">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Rendelések</title>
  <style>
    table { width: 100%; border-collapse: collapse; }
    th, td { border: 1px solid black; padding: 8px; text-align: left; }
    th { background-color: #f2f2f2; }
  </style>
</head>
<body>
<h2>Rendelések</h2>
<p><%= message %></p>
<table>
  <thead>
    <tr>
      <th>ID</th>
      <th>Név</th>
      <th>Ár</th>
    </tr>
  </thead>
  <tbody>
    <% items.forEach(item => { %>
      <tr>
        <td><%= item.id %></td>
        <td><%= item.itemName %></td>
        <td><%= item.itemPrice %> Ft</td>
      </tr>
    <% }) %>
  </tbody>
</table>
</body>
</html>
```

---

## Teszt dokumentáció

Ebben a fejezetben néhány teszt kerül bemutatásra, mind a Backend-ből, mind a Frontend-ből. A Jest és Supertest kombinációja ideális választás a JavaScript backend teszteléséhez, mivel mindkét eszköz erőteljes funkcionalitást és egyszerűsített tesztelési folyamatot kínál. A Jest egy könnyen konfigurálható és gyors tesztelő keretrendszer, amely lehetővé teszi a szinkron és aszinkron tesztek egyszerű írását, miközben beépített eszközöket biztosít a mockoláshoz, kémkedéshez és időzítők kezeléséhez.

A Supertest pedig kifejezetten HTTP-kérések küldésére és válaszok tesztelésére lett tervezve, így ideális választás API végpontok tesztelésére. A két eszköz együtt biztosítja, hogy könnyedén írhatunk integrációs teszteket, amelyek ellenőrzik a backend különböző komponenseinek együttműködését, miközben a tesztek jól olvashatóak és gyorsan futtathatóak maradnak. Ezen kívül a Jest részletes tesztjelentései és a Supertest könnyed HTTP kérés-képességei gyorsabb hibakeresést és fejlesztést tesznek lehetővé, így biztosítva a hatékony és robusztus backend fejlesztést.

### openingHourController.test.js

Ez a fájl az **Opening Hour API** tesztelésére szolgál a **Jest** és **Supertest** segítségével. A tesztkörnyezet beállítására, az API végpontjainak működésének ellenőrzésére, valamint az adatbázis inicializálására és tisztítására szolgál.

---

### Módszertan és Módszerek

- **Supertest és Jest:** A tesztelés során a Supertest könyvtárat használjuk az API végpontok kérésének küldésére és az HTTP válaszok ellenőrzésére. A Jest segítségével pedig a teszteket futtatjuk és validáljuk az eredményeket.
- **Aszinkron tesztelés:** Mivel az API végpontok aszinkron műveleteket végeznek (adatbázis műveletek, HTTP kérések), a tesztek aszinkron módon lettek megírva,

és a `async/await` szintaxist alkalmazzuk a megfelelő válaszok megvárására és az ellenőrzések végrehajtására.

- **Tesztelési scenáriók:** A tesztek minden fontos forgatókönyvet lefednek, beleértve a sikeres nyitvatartási idő létrehozását, lekérését és törlését, biztosítva ezzel, hogy a Controller minden funkciója megfelelően működjön.

---

## Tesztkörnyezet és Beállítások

- **Kérések küldése:** A tesztkészlet POST, GET, és DELETE HTTP metódusokkal dolgozik az `openingHours` végpontokkal.
- **Válaszok ellenőrzése:** Minden teszt biztosítja, hogy a válaszok a megfelelő státuszkódot tartalmazzák, és a válaszban a szükséges adatok, például az `insertId` vagy a törlés üzenete megtalálható.
- **Hibaellenőrzés:** Ha a válasz nem a várt státuszkóddal érkezik, a tesztek hibaüzenetet generálnak, amely segít a hiba forrásának azonosításában.

## Tesztek előkészítése

- **Adatbázis inicializálás (`beforeAll`)**
  - Kapcsolódik az adatbázishoz.
  - Létrehozza a `test_vizsgaremek` tesztadatbázist és az `openinghours` táblát.
  - Előre beállított nyitvatartási adatokkal tölti fel az adatbázist.
- **Adatbázis tisztítása (`afterAll`)**
  - A tesztek lefutása után törli az `openinghours` táblát.
  - Megszünteti az adatbázis-kapcsolatot.

```
beforeAll( fn: async () :Promise<void> => { /* csan */
  await new Promise( executor: (resolve, reject) :void => {
    connect.connect((err) :void => {
      if (err) reject(err);
      else resolve();
    });
  });

  await new Promise( executor: (resolve, reject) :void => {
    connect.query("CREATE DATABASE IF NOT EXISTS test_vizsgaremek", (err) :void => {
      if (err) reject(err);
      else resolve();
    });
  });

  await new Promise( executor: (resolve, reject) :void => {
    connect.query("USE test_vizsgaremek", (err) :void => {
      if (err) reject(err);
      else resolve();
    });
  });

  await new Promise( executor: (resolve, reject) :void => {
    connect.query(
      CREATE_TABLE_IF_NOT_EXISTS_openinghours (
        id INT AUTO_INCREMENT PRIMARY KEY,
        dayName VARCHAR(255),
        fromHour TIME,
        untilHour TIME
      ), (err) :void => {
        if (err) reject(err);
        else resolve();
      });
  });

  await new Promise( executor: (resolve, reject) :void => {
    connect.query("INSERT INTO openinghours (dayName, fromHour, untilHour)" +
      VALUES ('Monday', '09:00:00', '17:00:00'), ('Tuesday', '09:00:00', '17:00:00')", (err) :void => {
      if (err) reject(err);
      else resolve();
    });
  });
});

afterAll( fn: async () :Promise<void> => { /* csan */
  await new Promise( executor: (resolve, reject) :void => {
    connect.query("DROP TABLE IF EXISTS openinghours", (err) :void => {
      if (err) reject(err);
      else resolve();
    });
  });

  await new Promise( executor: (resolve, reject) :void => {
    connect.end((err) :void => {
      if (err) reject(err);
      else resolve();
    });
  });
});
```

---

## Főbb Funkciók és Tesztelt Funkciók

- **POST /api/openingHours** – Új nyitvatartási idő létrehozása
  - **Sikeres nyitvatartási idő létrehozása:** A teszt ellenőrzi, hogy a rendszer képes új nyitvatartási idő hozzáadására egy POST kéréssel, és helyes választ ad. A nyitvatartási idő tartalmazza a napot, az induló és befejező időpontot.
    - **Várt válasz:** 200 OK státuszkód, valamint a válaszban lévő data objektumban egy insertId mező.
- **GET /api/openingHours** – Minden nyitvatartási idő lekérése
  - **Nyitvatartási idő lekérése:** A teszt azt ellenőrzi, hogy a rendszer sikeresen visszaadja az összes nyitvatartási időt egy GET kéréssel.
    - **Várt válasz:** 200 OK státuszkód, valamint a válaszban lévő data objektumban található nyitvatartási idők listája.
- **DELETE /api/openingHours/:id** – Nyitvatartási idő törlése az adott ID alapján
  - **Sikeres törlés:** A teszt biztosítja, hogy a rendszer képes törölni egy adott nyitvatartási időt az id alapján, és a válasz megfelelő üzenetet küld.
    - **Várt válasz:** 200 OK státuszkód, valamint a válaszban lévő message mezőben szerepelnie kell a "Időpont sikeresen törölve." üzenetnek.



```
describe('Opening Hour Controller', blockFn: () :void => {
  describe('POST /api/openingHours', blockFn: () :void => {
    it('should create a new opening hour', fn: async () :Promise<void> => {
      const openingHour :{...} = {
        day: 'Wednesday',
        from: '09:00:00',
        until: '17:00:00'
      };
      const res = await request(app).post('/api/openingHours').send(openingHour);
      if (res.statusCode !== 200) {
        console.error('Error response:', res.body);
      }
      expect(res.statusCode).toEqual( expected: 200);
      expect(res.body).toHaveProperty( expectedPath: 'data');
      expect(res.body.data).toHaveProperty( expectedPath: 'insertId');
    });
  });

  describe('GET /api/openingHours', blockFn: () :void => {
    it('should get all opening hours', fn: async () :Promise<void> => {
      const res :string[] = await request(app).get('/api/openingHours');
      if (res.statusCode !== 200) {
        console.error('Error response:', res.body);
      }
      expect(res.statusCode).toEqual( expected: 200);
      expect(res.body).toHaveProperty( expectedPath: 'data');
    });
  });

  describe('DELETE /api/openingHours/:id', blockFn: () :void => {
    it('should delete an opening hour by the given id', fn: async () :Promise<void> => {
      const openingHourId :number = 1; // Replace with a valid opening hour ID
      const res = await request(app).delete(`/api/openingHours/${openingHourId}`);
      if (res.statusCode !== 200) {
        console.error('Error response:', res.body);
      }
      expect(res.statusCode).toEqual( expected: 200);
      expect(res.body).toHaveProperty( expectedPath: 'message', expectedValue: 'Időpont sikeresen törölve.');
```

## Következtetés

Ez a tesztkészlet biztosítja, hogy az Opening Hour Controller API végpontjai, amelyek a nyitvatartási időket kezelik, helyesen működnek. A tesztek figyelmet fordítanak a sikeres műveletek mellett a válaszok épségére, így biztosítva, hogy minden API végpont helyesen kezeli a nyitvatartási időkhöz létrehozását, lekérését és törlését. Az izolált tesztelés és az aszinkron működés biztosítja, hogy a tesztkészlet megbízható és gyors legyen.

## authController.test.js

Ez a fájl a **Jest** és a **mocking** technikák segítségével teszteli az **authController** működését, amely a felhasználói regisztrációt, bejelentkezést és kijelentkezést kezeli. A tesztkörnyezet beállítása során a User modellt, a bcrypt jelszókezelő könyvtárat, a jsonwebtoken token generálót, az uuid generátort és a sessionHandler middleware-t **mock**-oljuk, hogy a tesztek izoláltak legyenek, és a tényleges adatbázis vagy külső szolgáltatások ne befolyásolják őket.

### Módszertan és Módszerek

- **Mocking:** Az összes külső függőség (például User, bcrypt, jsonwebtoken, uuid, sessionHandler) mock-olása biztosítja, hogy a tesztek izoláltak legyenek, és ne függjenek az adatbázistól vagy más külső rendszerektől.
- **Jest és Supertest:** A tesztek a **Jest** keretrendszer segítségével készülnek, amely biztosítja a könnyű tesztelést és az aszinkron kód kezelését. A **Supertest** és az integrációs tesztek segítségével az API végpontok viselkedése ellenőrizhető.
- **Tesztelési scénáriók:** A tesztek minden fontos forgatókönyvet lefednek, beleértve a hibakezelést, sikeres regisztrációt és bejelentkezést, valamint a session kezelést.

A mock-olás bemutatása:

```
describe('Auth Controller', blockFn: () : void => {
  let req, res;

  beforeEach(fn: () : void => {
    req = {
      body: {},
      params: {},
      cookies: {},
      session: {
        destroy: jest.fn()
      }
    };

    res = {
      status: jest.fn().mockReturnThis(),
      json: jest.fn(),
      send: jest.fn(),
      cookie: jest.fn()
    };
  });
});
```

## Főbb Funkciók és Tesztelt Funkciók

- **register funkció tesztelése**

- **Hiányzó mezők ellenőrzése:** A teszt ellenőrzi, hogy ha bármelyik kötelező mező (pl. név, email, jelszó) hiányzik, akkor a válasz 400 státuszkóddal és megfelelő hibaüzenettel tér vissza.
- **Már regisztrált email:** Ha a regisztrált email már létezik az adatbázisban, akkor 409 státuszkóddal és a "email már foglalt" üzenettel válaszol.
- **Sikeres regisztráció:** Ha minden adat helyes, a teszt ellenőrzi, hogy a felhasználó sikeresen regisztrálódik, és a megfelelő adatokat (felhasználó id, név, email) kapja vissza.

```
describe('blockName: 'register', blockFn: () :void => {
  it('testName: 'should return 400 if any field is missing', fn: async () :Promise<void> => {
    req.body = { name: '', email: '', password: '', permissionId: '' };
    await authController.register(req, res);
    expect(res.status).toBeCalledWith( expected: 400);
    expect(res.json).toBeCalledWith( expected: { message: "Minden mező megadása kötelező." });
  });

  it('testName: 'should return 409 if email is already taken', fn: async () :Promise<void> => {
    req.body = { name: 'Test', email: '__test__@example.com', password: 'password', permissionId: 1 };
    User.findOne.mockResolvedValue( value: { data: [] });
    await authController.register(req, res);
    expect(res.status).toBeCalledWith( expected: 409);
    expect(res.send).toBeCalledWith( expected: { message: "Az email cím már foglalt." });
  });

  it('testName: 'should return 200 and save user if registration is successful', fn: async () :Promise<void> => {
    req.body = { name: 'Test', email: '__test__@example.com', password: 'password', permissionId: 1 };
    User.findOne.mockResolvedValue( value: { data: [] });
    bcrypt.genSalt.mockResolvedValue( value: 'salt');
    bcrypt.hash.mockResolvedValue( value: 'hashedPassword');
    User.prototype.save.mockResolvedValue( value: { id: 1, name: 'Test', email: '__test__@example.com' });
    await authController.register(req, res);
    expect(res.status).toBeCalledWith( expected: 200);
    expect(res.send).toBeCalledWith( expected: { id: 1, name: 'Test', email: '__test__@example.com' });
  });
});
```

- **login funkció tesztelése**

- **Session ellenőrzés:** A teszt figyeli, hogy ha a session ellenőrzés nem sikerül, akkor a válasz 400 státuszkóddal és "Session check failed" üzenettel érkezik.
- **Érvénytelen email vagy jelszó:** Ha az email nem található, vagy a megadott jelszó nem egyezik a mentett jelszóval, a rendszer megfelelő 400 státuszkóddal és a "Érvénytelen email cím" vagy "Érvénytelen jelszó" üzenetekkel válaszol.
- **Sikeres bejelentkezés:** Sikeres bejelentkezés esetén a teszt ellenőrzi, hogy a rendszer helyesen generálja le a JWT token-t és a megfelelő cookie-kat beállítja.

```
describe('login', { blockFn: () :void => {
  it('testName: 'should return 400 if session check fails'', fn: async () :Promise<void> => {
    req.body = { email: 'idkbro@idk.com', password: 'password' };
    checkSession.mockResolvedValue({ value: false }); // Mock session check failure
    await authController.login(req, res);
    expect(res.status).toHaveBeenCalledWith( expected: 400);
    expect(res.send).toHaveBeenCalledWith( expected: { message: "Session check failed." });
  });

  it('testName: 'should return 400 if email is invalid'', fn: async () :Promise<void> => {
    req.body = { email: 'invalid@example.com', password: 'password' };
    User.findOne.mockResolvedValue({ value: { data: [] } }); // No user found
    checkSession.mockResolvedValue({ value: { response: false } }); // Mock session check failure
    await authController.login(req, res);
    expect(res.status).toHaveBeenCalledWith( expected: 400);
    expect(res.send).toHaveBeenCalledWith( expected: { message: "Érvénytelen email cím." });
  });

  it('testName: 'should return 400 if password is invalid'', fn: async () :Promise<void> => {
    req.body = { email: '___test___@example.com', password: 'invalidPassword' };
    User.findOne.mockResolvedValue({ value: { data: [{ email: '___test___@example.com', hashedPassword: 'hashedPassword' }] } });
    bcrypt.compare.mockResolvedValue({ value: false }); // Password does not match
    checkSession.mockResolvedValue({ value: { response: false } }); // Mock session check failure
    await authController.login(req, res);
    expect(res.status).toHaveBeenCalledWith( expected: 400);
    expect(res.send).toHaveBeenCalledWith( expected: { message: "Érvénytelen jelszó." });
  });

  it('testName: 'should return 200 and set cookies if login is successful'', fn: async () :Promise<void> => {
    req.body = { email: '___test___@example.com', password: 'password' };
    User.findOne.mockResolvedValue({ value: { data: [{ email: '___test___@example.com', hashedPassword: 'hashedPassword' }] } });
    bcrypt.compare.mockResolvedValue({ value: true });
    uuidv4.mockReturnValue('uuid');
    jwt.sign.mockReturnValue('token');
    await authController.login(req, res);
    expect(res.cookie).toHaveBeenCalledWith( expected: 'token', 'token', expect.any(Object));
    expect(res.status).toHaveBeenCalledWith( expected: 200);
    expect(res.send).toHaveBeenCalledWith( expected: { message: "Belépve új munkamenettel." });
  });
});
```

- **logout funkció tesztelése**

- **Sikeres kijelentkezés:** A teszt biztosítja, hogy ha a kijelentkezés sikeresen végrehajtódik, akkor a megfelelő státuszkóddal és üzenettel válaszol.
- **Hiba a kijelentkezés során:** Ha valamilyen hiba lép fel a session törlése közben, a válasz 500 státuszkóddal és a hiba részletezésével érkezik.

```
describe('logout', () => {
  it('testName: 'should return 200 and set cookies if login is successful'', { fn: async () => {
    req.body = { email: '___test___@example.com', password: 'password' };
    User.findOne.mockResolvedValue({ data: [{ email: '___test___@example.com', hashedPassword: 'hashedPassword' }] });
    bcrypt.compare.mockResolvedValue({ value: true });
    uuidv4.mockReturnValue('uuid');
    jwt.sign.mockReturnValue('token');
    await authController.login(req, res);
    expect(res.cookie).toHaveBeenCalledWith({ expected: 'token', 'token', expect.any(Object) });
    expect(res.status).toHaveBeenCalledWith({ expected: 200 });
    expect(res.send).toHaveBeenCalledWith({ expected: { message: "Belépve új munkamenettel." } });
  });

  it('testName: 'should return 500 if an error occurs during logout'', { fn: async () => {
    req.cookies['connect.sid'] = 's:sessionId';
    deleteSession.mockRejectedValue(new Error('Error'));
    await authController.logout(req, res);
    expect(res.status).toHaveBeenCalledWith({ expected: 500 });
    expect(res.send).toHaveBeenCalledWith({ expected: { message: "Hiba kijelentkezéskor.", error: expect.any(Error) } });
  });
});
```

Ez a tesztkészlet biztosítja, hogy az **authController** minden funkciója helyesen működik, figyelve a felhasználói regisztráció, bejelentkezés, és kijelentkezés megfelelő működésére, miközben ellenőrzi a hibakezelést és az adatbázis-függőségeket. A mockolt komponensek használata biztosítja a tesztek izoláltságát, gyorsaságát és megbízhatóságát.

## Fejlesztési lehetőségek

Jelenlegi éttermi rendszerünk egyik legfontosabb hiányossága, hogy a foglalási oldalon nincs lehetőség az asztalok helyének módosítására. A foglalásokat jelenleg az asztalok fix helyeihez rendeljük, de ez nem biztosít kellő rugalmasságot a rendszer használói számára. Például előfordulhat, hogy egy vendég szeretne egy nagyobb asztalt, amely több szék köré van elrendezve, de mivel az asztalok elhelyezkedése nem módosítható a rendszerben, ez nem lehetséges. Ez különösen problémás lehet csúcsidőszakokban, amikor több foglalás is egyes asztalokhoz van rendelve, és a rendelkezésre álló helyek nem biztosítanak elegendő rugalmasságot a vendégek igényeinek megfelelően.

A jelenlegi rendszer nem veszi figyelembe azt a fontos igényt sem, hogy az asztalok helyét az étterem tulajdonosa, vagy a foglalási menedzser könnyen át tudja alakítani, hogy a vendégek számára ideális környezetet biztosítson. Az asztalok mozgathatóságának bevezetése lehetővé tenné az étterem számára, hogy rugalmasabban kezelje a helyeket, és jobban igazodjon a vendégek igényeihez.

A PlateMate jelenlegi verziója még nem rendelkezik többnyelvű támogatással, ami korlátozhatja az étterem alkalmazottainak hatékony munkavégzését, különösen, ha többnyelvű személyzet dolgozik egy adott helyszínen. Egy olyan rugalmas nyelvválasztási rendszer bevezetése, amely lehetőséget ad a felhasználóknak a különböző nyelvek közötti váltásra, nagymértékben javítaná a rendszer használhatóságát és hatékonyságát.

Ez a funkció nem csupán az alkalmazottak kényelmét szolgálná, hanem hozzájárulna a gyorsabb és pontosabb rendeléskezeléshez, valamint a kommunikációs akadályok csökkentéséhez. A rendszer automatikusan alkalmazkodhatna az egyes felhasználók nyelvi preferenciáihoz, így minden dolgozó a számára legkényelmesebb nyelven érhetné el az információkat.

A többnyelvű támogatás egy fontos lépés lehet a PlateMate rendszer továbbfejlesztésében, amely hozzájárulhat az éttermi személyzet hatékonyabb munkavégzéséhez, minimalizálhatja a félreértéseket, és javíthatja az étterem belső kommunikációját.

### III. Felhasználói dokumentáció

Ebben a fejezetben a felhasználót segítő információk közlése lesz a fő szempont, egészen a telepítéstől, a használaton keresztül a helytelen használatból adódott hibák elhárításáig. Többek között bemutatjuk a felhasználók, rendelések, fizetések és foglalások kezelését, valamint a jogosultságokat és a munkameneteket. A dokumentáció lépésről lépésre ismerteti a funkciókat, biztosítva a gördülékeny működést és a felhasználói élmény optimalizálását.

#### A program funkciói

A **PlateMate** éttermi rendszer célja, hogy a programot használó éttermeknek egy gördülékeny, könnyű használatot nyújtson az éttermi rendelések, foglalások, számla rendezések lebonyolításában. A program többek között képes: foglalásokat kezelni, rendeléseket felvenni az adott számú asztalhoz, egyidőben (real-time) megjeleníteni az éppen felvett rendeléseket a konyha számára, majd kiszolgálás után az összes elfogyasztott terméket asztal szerint kifizetésre bocsájtani az étterem által megadott ár lista szerint.

A program lehetőséget nyújt 4 féle munkakört ellátó alkalmazottaknak fiók létrehozásra, így lehetővé téve a párhuzamos munkavégzést akár egy nagyobb méretű étterem számára is.

#### A 4 munkakör:

- **Adminisztrátor**
- **Séf**
- **Pincér**
- **Szakács**

Mindegyik munkakörhöz tartozó fiók (kivételet képez az adminisztrátor) csak a saját munkaköréhez tartozó oldalakat és funkciókat tudja elérni az alkalmazásban. Az adminisztrátor mindegyik munkakörnek eléri a funkcióit, továbbá ezzel a munkakörrel lehet csak felvenni új felhasználót, így az új munkások nem tudnak maguktól regisztrálni, hanem szükség van egy adminisztrátorra, aki csinál nekik egy fiókot. A szoftver bejelentkezés nélkül nem használható.

## Szükséges technikai eszközök

Mivel a program alapvetően Windows 10/11 -es operációs rendszerrel rendelkező számítógépekre (PC), illetve Android-os és iOS-es mobiltelefonokra/tabletekre lett kifejlesztve, ezért a program helyes működése csak kizárólag az ilyen eszközökön van garantálva. Bármilyen más eszköz használata nem a program helytelen működésének veszélyét vonja magával.

### Technikai adatok Windows PC-hez

Ebben a részben részletesebben is olvashatnak a hardveres, illetve szoftveres elvárásairól a programnak való futtatásához. A weblapon való látogatáshoz bármilyen alapértelmezett böngésző megfelelő, de lehet többek között: Firefox, vagy Chromium alapú is.

### Hardveres igények

- **CPU**
  - Intel Core i3-7100 3.9GHz vagy jobb, vagy
  - AMD Ryzen 3 1200 3.1GHz vagy jobb
- **GPU**
  - Intel HD Graphics 5500 vagy jobb, vagy
  - AMD Radeon Graphics (Ryzen 7000) vagy jobb
- **RAM:** 4GB 1600Mhz vagy jobb
- **Tárhely:** ~100MB szabad tárhely

Az applikáció ezekkel a hardverekkel rendelkező gépeken lett kipróbálva, és tesztelve, így legalacsonyabb hardveres követelményeket ezeken az adatokon kívül nem tudunk szolgáltatni, viszont a projekt és az alkalmazás természetéből, felépítéséből, és kivitelezéséből fakadóan nagy valószínűséggel ennél alacsonyabb hardveres specifikációkkal ellátott eszközökön is kifogástalanul futna.



## Szoftveres igények

A futtatás során a [Technikai Információk](#) részben leírt specifikációkkal teszteltük és futtattuk az applikációt. Azokkal a verziókkal mi biztosítjuk, hogy az applikáció kifogástalanul működni fog. Nagy valószínűséggel működni fog a PlateMate az újabb verziókon is, viszont, ha nem, úgy kérjük használják a megadott, régebbi verziót.

### Az applikáció elvárásai:

- Windows 10/11
- NPM megadott vagy a legfrissebb verziója
- NODE megadott vagy a legfrissebb verziója
- VUE megadott vagy legfrissebb verziója
- Xampp és MySQL megadott vagy legfrissebb verziója

## Telepítés és indítás

Ebben a fejezetben végig vezetjük a felhasználót, hogy hogyan telepítse és indítsa el a programot, a következő fejezetben pedig megtanítjuk használni azt.

### Adatbázis telepítése és futtatása

A projekthez elengedhetetlen, hogy legyen egy adatbázis, ami tartalmazza, tárolja az étteremmel kapcsolatos adatokat, és azokat lehet dinamikusan használni és kezelni. Éppen ezért mi az Xampp mellett döntöttünk, hiszen ez egy könnyen kezelhető, felhasználóbarát alkalmazás, ami lehetőséget nyújt MySQL szerver futtatására.

Ez a dokumentáció részletes útmutatót nyújt arról, hogyan importáljunk egy SQL fájlt egy adatbázisba az XAMPP használatával.

### 1. XAMPP Telepítése és Indítása

1. Töltse le a XAMPP legfrissebb verzióját a hivatalos weboldalról: <https://www.apachefriends.org>
2. Futtassa a telepítőt, majd kövesse az utasításokat.
3. A telepítés során hagyja meg az alapértelmezett komponenseket, beleértve az **Apache** és **MySQL** modult.
4. A telepítés befejezése után indítsa el a **XAMPP Control Panel**-t.

5. A **XAMPP Control Panel**-ben kattintson az **Apache** és **MySQL** melletti **Start** gombra.
6. Ellenőrizze, hogy mindkét modul zökkenőmentesen elindult (zöld "Running" jelzés megjelenik és ott is marad).

## 2. Adatbázis Létrehozása phpMyAdmin Felületen

1. Nyissa meg a böngészőben a következő címet: <http://localhost/phpmyadmin/>  
Az alapértelmezett bejelentkezési adatok: **user:** root, **password:** {üres}
2. A **phpMyAdmin** felületen kattintson az **Új** lehetőségre a bal oldali menüben.
3. Írja be az adatbázis nevét (vizsgaremek) az **Adatbázis neve** mezőbe.
4. Válassza ki a **kollációt** (utf8mb4\_hungarian\_ci).
5. Kattintson a **Létrehozás** gombra.

## 3. SQL Fájl Importálása

1. A phpMyAdmin bal oldali listájában válassza ki az imént létrehozott adatbázist.
2. A felső menüsoron kattintson az **Importálás** fülre.
3. Kattintson a **Fájl kiválasztása** gombra, és válassza ki a „PlateMate\db\vizsgaremek.sql” -t.
4. Görgessen lejjebb, majd kattintson az **Importálás végrehajtása** gombra.
5. Ha az importálás sikeres, a phpMyAdmin egy zöld értesítést jelenít meg a sikeres műveletről.

## 4. Hibaelhárítás

Ha az importálás során hiba lép fel, az alábbi problémák merülhetnek fel:

- **Fájlméret korlát:** Ha az SQL fájl túl nagy, szerkeszteni kell a php.ini fájlt és módosítani az upload\_max\_filesize és post\_max\_size értékeit.
- **Adatbázis már létezik:** Győződjön meg arról, hogy az importált SQL fájl nem próbál meg létrehozni egy már meglévő adatbázist.

## 5. További Lépések

- Az importált adatokat a phpMyAdmin **Szerkezet** vagy **Böngészés** füle alatt ellenőrizheti.



- Ha szüksége van további módosításokra, az **SQL** fül alatt futtathat egyedi lekérdezéseket.
- Ellenőrizze az adatbázis kapcsolat beállításait a kódodban (pl. config.php vagy .env fájl).

Ezzel sikeresen importálta az SQL fájlot a XAMPP segítségével! Ha bármilyen további kérdése van, ellenőrizze a **phpMyAdmin** dokumentációját vagy a XAMPP hivatalos fórummal lépjen kapcsolatba.

## Frontend és Backend telepítése és futtatása

A projekt fő fájljait ezen lépésekkel tudja beimportálni.

Ez a dokumentáció részletes útmutatót nyújt egy Node.js és Vue.js alapú projekt klónozásához, telepítéséhez és futtatásához. A projekt mappa struktúrája a következő:

```
PlateMate /
```

```
| -- server/      # Backend (Node.js, Express)
```

```
| -- client/      # Frontend (Vue.js)
```

### 1. Projekt Klónozása vagy Letöltése

A projekt letöltése két módon történhet: **Git repository klónozásával** vagy **ZIP fájl letöltésével**.

#### 1.1 Git Repository Klónozása

1. Nyisson meg egy terminált vagy parancssort adminisztrátor módban.
2. Navigáljon arra a könyvtárra, ahova a projektet szeretné klónozni, például:

```
cd c:/code/
```

3. Futtassa az alábbi parancsot a repository klónozásához:

```
git clone https://github.com/Pollak-Projects/PlateMate
```

4. Lépjen be a projekt mappájába:

```
cd PlateMate
```



## 1.2 ZIP Fájl Letöltése

1. Nyissa meg a projekt GitHub oldalát (<https://github.com/Pollak-Projects/PlateMate>)
2. Kattintson a **Code** gombra, majd válassza a **Download ZIP** opciót.
3. Csomagolja ki a ZIP fájlt egy kívánt helyre.
4. Nyisson meg egy terminált, és navigáljon az így létrejött mappába, például:

```
cd c:/code/unzippedProjects/PlateMate
```

## 2. Függőségek Telepítése

A projekt két részből áll: **backend (server)** és **frontend (client)**. Mindkét részhez külön kell telepíteni a függőségeket.

### 2.1 Backend Telepítése

1. Navigáljon a server mappába:

```
cd server
```

2. Telepítse a szükséges Node.js csomagokat:

```
npm install
```

### 2.2 Frontend Telepítése

1. Nyisson meg egy új terminált vagy lépjen vissza a fő könyvtárba:

```
cd ../client
```

2. Telepítse a frontend függőségeket:

```
npm install
```

## 3. Projekt Indítása

A backend és a frontend külön folyamatként futtatható.

### 3.1 Backend Indítása

1. Navigáljon a server mappába:

```
cd server
```

2. Indítsa el a szerveret:

```
node .
```



3. Ha sikeresen elindult, az API a <http://localhost:3000> címen érhető el (a port száma változhat a beállításoktól függően).

### 3.2 Frontend Indítása

1. Nyisson meg egy új terminált, és navigáljon a client mappába:

```
cd client
```

2. Indítsa el a Vue.js alkalmazást:

```
npm run dev
```

3. Ha sikeresen elindult, a frontend a <http://localhost:5173> címen érhető el (a port száma változhat a lokális változóktól függően).

### 4. Hibaelhárítás

Ha bármilyen probléma merül fel, próbálja meg az alábbi lépéseket:

- **Hibás csomagok vagy telepítési gondok:** Próbálja meg újratelepíteni a függőségeket: törölje a node\_modules mappát, majd futtassa:

```
npm install
```

Ezt a server és client mappákban is el kell végezni.

- **Port ütközés:** Ha egy port már foglalt, próbáljon meg egy másikat beállítani az .env fájlban vagy a konfigurációs fájlokban, vagy futtassa a:

```
npx kill-port (port száma)
```

majd próbálja meg újra futtatni

Ezzel sikeresen telepítette és elindította a projektet!



## A program bemutatása

A program amint már említve lett felül, egy éttermi rendszer, ami az éttermi dolgozók munkájának megkönnyítésére lett kifejlesztve. A program alkalmas az étterem adminisztratív, főbb részeinek ellátására, illetve rendelkezik egy fiókos rendszerrel, amellyel az összes alkalmazott egyidőben tudja használni az applikációt párhuzamosan a munkatársaikkal, így lehetővé téve nagyobb és kisebb cégek számára is egyaránt a használatot.

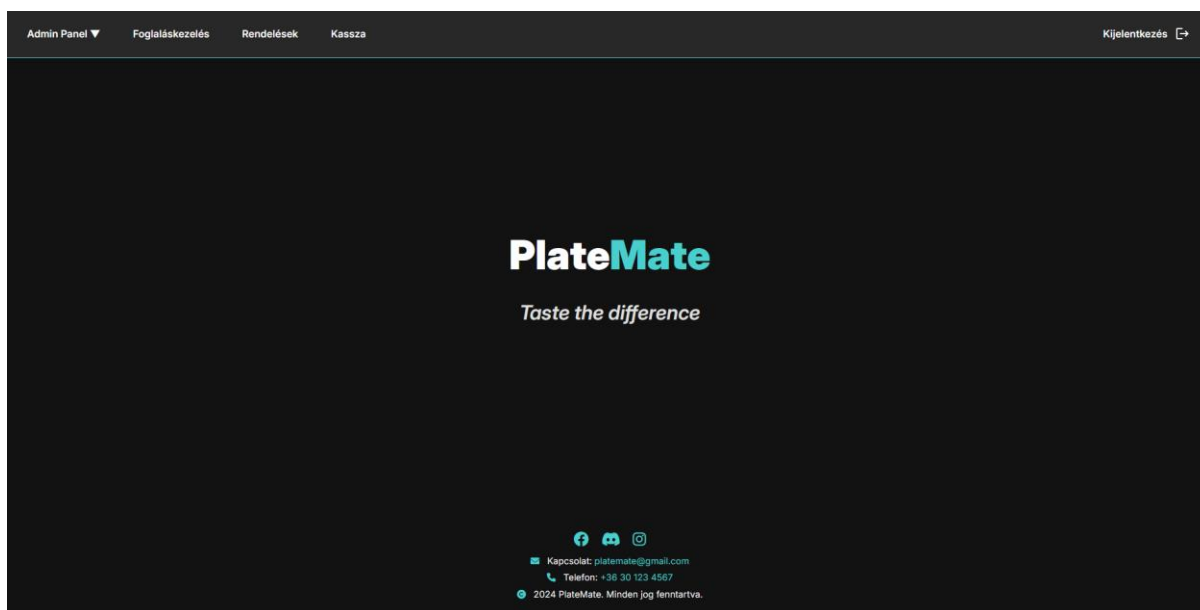
## Az alkalmazás főbb részei

Ez a rész tartalmazza az összes funkciót, amit már feljebb [A program funkciói](#) részben már lett említve, csak sokkal részletesebben jellemezve, és hogy hogyan lehet használni.

## Kezdőlap

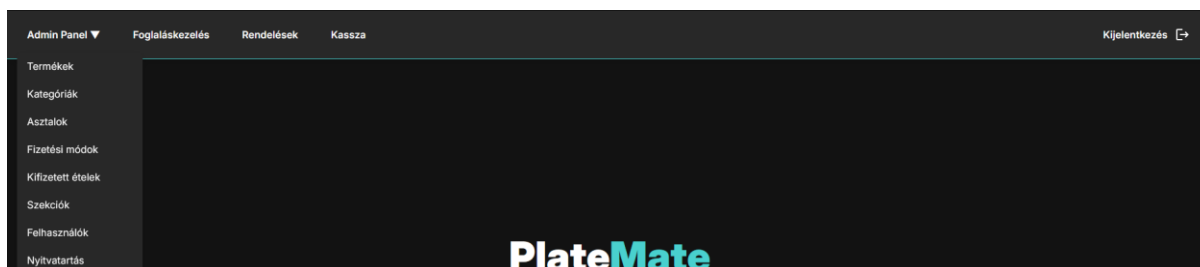
A kezdőlapról a navigációs menün, és a hamburger menün keresztül (csak adminisztrátoroknak) lehet elérni a program összes funkcióját. Ezeket használva könnyen navigálhat a felhasználó az oldalak között, és érheti el az ő szekcióihoz tartozó oldalakat.

### Kezdőlap



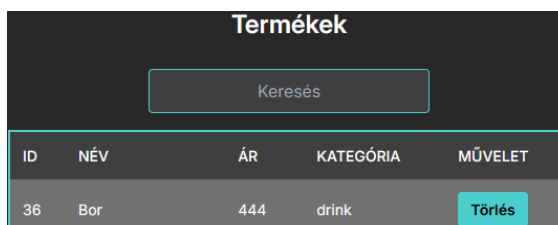
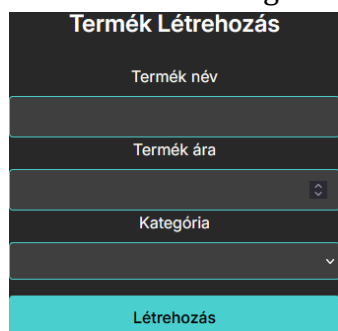
## Admin Panel (Lenyíló Menü, csak adminisztrátornak fog megjelenni)

Ez a lenyíló menüsor csak az adminisztrátoroknak fog megjelenni, itt érhetik el a külön nekik szóló adminisztrátori oldalakat.



- **Termékek:**

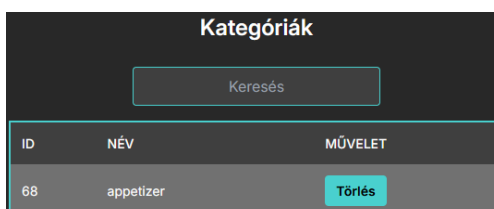
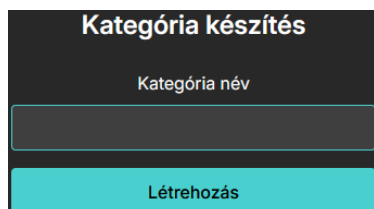
- A termékek adatainak kitöltése után a gombra kattintva akár azonnal is megtekinthetők, kitörölhetők, név szerint listázhatóak az ételek. Létrehozásnál a beviteli mezőkbe a termék árához csak számot írhat be, a többihez nincs megkötés.



ID	NÉV	ÁR	KATEGÓRIA	MŰVELET
36	Bor	444	drink	Törölés

- **Kategóriák:**

- Az új kategória nevének megadása után (nincs formai megkötés) és a „Létrehozás” gombra nyomás után, szintén azonnal megtekinthetők a kategóriák a listázás föl alatt, név szerint kikereshetők.



ID	NÉV	MŰVELET
68	appetizer	Törölés

### Asztalok:

- Egy még nem létező asztalszám hozzáadása a rendszerhez a beviteli mezőn keresztül. Létrehozás után szintén megtekinthetők az asztalszámok a listázás fül alatt, és törölhetők.

#### Asztal Hozzáadás

Asztal szám

Létrehozás

#### Asztalok

ID	ASZTALSZÁM	MŰVELET
1	1	Törölés

### Fizetési módok:

- Új fizetési mód felvétele megtehető a bevitelmező kitöltése után (nincs formai megkötés), majd törölhető, illetve megtekinthetők a listázás lapon.

#### Fizetési mód készítés

Neve

Létrehozás

#### Fizetési módok

ID	NÉV	MŰVELET
23	kártya	Törölés

### Kifizetett ételek:

- A már kifizetett és teljesített ételeknek a megtekintése asztalokra és dátumra bontva.

Kifizetett Termékek					
ASZTALSZÁM	FOGYASZTOTT TERMÉK	TERMÉK ÁRA	FIZETÉSI MÓD	FIZETÉS IDEJE	MŰVELET
8 2	Bor	444	kártya	2025-03-06 12:17	Törölés

### Szekciók:

- Az étteremhez új munkakör hozzáadása, illetve törlése. Beviteli mezőnél megkötés nincs.

#### Szekció készítés

Szekció név

Létrehozás

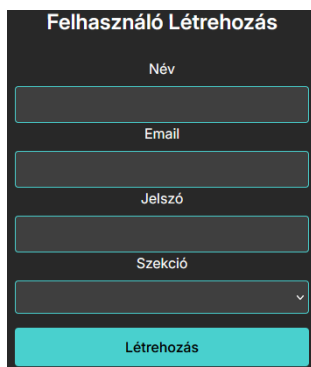
#### Szekciók

ID	NÉV	MŰVELET
1	admin	Törölés



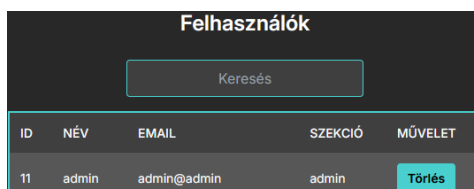
- **Felhasználók:**

- Új felhasználók (pl.: adminisztrátorok, pincérek, kasszások) hozzáadása.  
Formai megkötés csak az email-nél van, annak egy valós email cím formátumban kell szerepelnie.



A form for creating a new user. It has a title 'Felhasználó Létrehozás'. Below the title are four input fields: 'Név', 'Email', 'Jelszó', and 'Szekció'. The 'Szekció' field is a dropdown menu. At the bottom is a red button labeled 'Létrehozás'.

- Felhasználók megtekintése és törlése megtehető a listázás lapon.

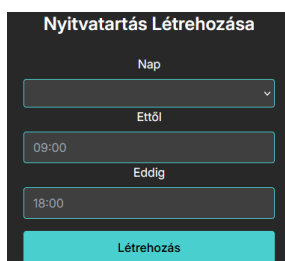


A table showing a list of users. The title is 'Felhasználók'. There is a search bar labeled 'Keresés'. The table has five columns: ID, NÉV, EMAIL, SZEKCIÓ, and MŰVELET. There is one row with the following data: ID: 11, NÉV: admin, EMAIL: admin@admin, SZEKCIÓ: admin, MŰVELET: Törlés.

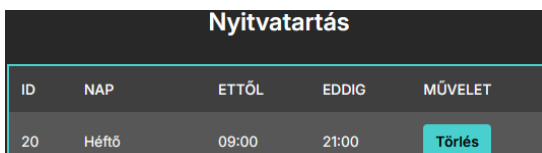
ID	NÉV	EMAIL	SZEKCIÓ	MŰVELET
11	admin	admin@admin	admin	Törlés

- **Nyitvatartás:**

- Az üzlet nyitvatartási idejének beállítása és módosítása.



A form for creating a new opening hours entry. It has a title 'Nyitvatartás Létrehozása'. Below the title are four input fields: 'Nap' (a dropdown menu), 'Ettől', 'Eddig', and 'Létrehozás' (a red button).



A table showing a list of opening hours entries. The title is 'Nyitvatartás'. The table has five columns: ID, NAP, ETTŐL, EDDIG, and MŰVELET. There is one row with the following data: ID: 20, NAP: Héftő, ETTŐL: 09:00, EDDIG: 21:00, MŰVELET: Törlés.

ID	NAP	ETTŐL	EDDIG	MŰVELET
20	Héftő	09:00	21:00	Törlés



## Foglaláskezelés

- Új foglалások létrehozása és rögzítése. Formai megkötés a név mezőnél nincs, a fők számának viszont számnak kell lennie.

**Asztalfoglalás**

1 2 Pult 9 10  
3 4 11 12  
5 6 13 14  
7 8 15 16

**Adatok**

Név

Fők száma

mm / dd / yyyy

Ettől

Eddig

Foglalás

- Létező foglалások törlése.
- Asztalok foglaltságának áttekintése, keresés név alapján.

**Foglalások**

Keresés

ID	NÉV	VENDÉGEK	ASZTALSZÁM	ETTŐL	EDDIG
11	asd	5	1	2024-11-03 17:00	2024-11-03 18:00



## Rendelések

- Új rendelések kezelése és továbbítása a konyhának.
- A már felszolgált, éppen készülő, elkészült rendelések megtekintése, keresés név alapján.
- Rendelések állapotának frissítése.
- Rendelések törlése.

### Termékek listája

Keresés...

NÉV	MŰVELET
Bor	Hozzáadás
Pizza	Hozzáadás
Suti	Hozzáadás
cinanimin rolls	Hozzáadás

### Hozzáadott termékek

Keresés...

NÉV	MŰVELET
-----	---------

Asztalszám

7

Leadás

### Készülő termékek

Keresés

ID	ASZTALSZÁM	TERMÉK	FELVÉVE	MŰVELET	MŰVELET
178	15	Pizza	2025-02-04 08:12	Kész	Törlés

### Elkészült termékek

Keresés

ASZTALSZÁM	TERMÉK	FELVÉVE	MŰVELET	MŰVELET	MŰVELET
14	Bor	2025-02-03 14:47	Vissza	Felszolgálva	Törlés

### Felszolgált termékek

Keresés

ID	ASZTALSZÁM	TERMÉK	FELVÉVE	MŰVELET	MŰVELET
99	12	Bor	2024-12-10 10:14	Vissza	Törlés

## Kassza

- Asztal kiválasztása után a program feldobja ahhoz az asztalhoz éppen kifizetetlen rendeléseket, és a fizetésmód kiválasztása után a kifizetés gomb továbbítja az összeget a terminálba, vagy a kasszába (nincs implementálva a továbbítás).

**Elfogyasztott termékek**

ID	NÉV	ÁR
99	Bor	444
100	Pizza	333
101	Pizza	333
102	Suti	3333

Asztalszám

12

Fizetési mód

kártya

Végösszeg: 4443

Kifizetés

## Első beléptetés az alkalmazásba

Mivel az alkalmazásba nem lehet külön regisztrálni, csak egy adminisztrátor tud új felhasználót csinálni, ezért van egy alapértelmezett fiók, aminek adminisztrátori jogosultsága van, így be tudnak lépni, tesztelni, új felhasználókat létrehozni.

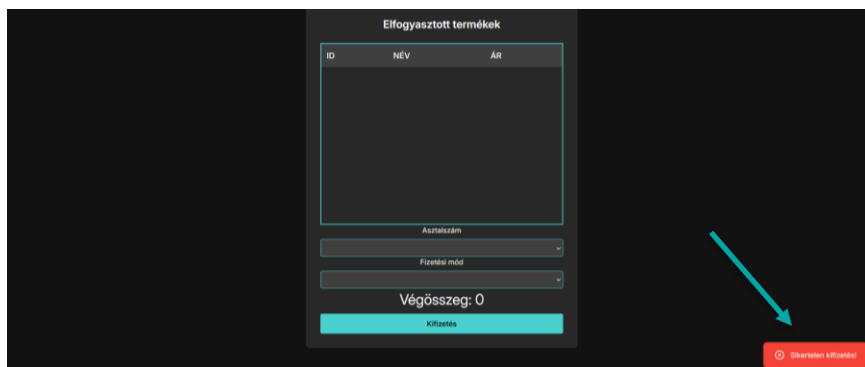
Email: admin@admin Jelszó: admin

## Helytelen használat esetei

Helytelen használat akkor alakul ki, amikor használat közben a felhasználó, ebben az esetben az éttermi dolgozó olyan műveletet akar végrehajtani a szoftverrel, amire nem lett kitalálva, illetve hibásan, rossz módon kívánja végrehajtani azokat a műveleteket, amire ki lett találva. Ezekből az esetekből, ha nem lennének kezelve, akár a szoftver leállása is fakadhatna. Az alkalmazásban ezek az esetek mind kezelve vannak kivétel nélkül. Ha a szoftver mégis leáll, **az úgy lett kitalálva**.

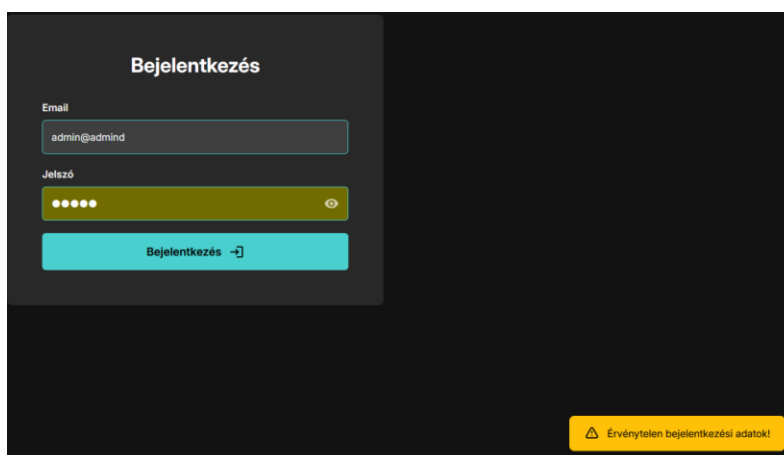
## Hibajelzések

Ha bármilyen adatot helytelenül adnak meg az alkalmazásban, akkor minden esetben egy hasonló hibaüzenet fogja fogadni a felhasználót

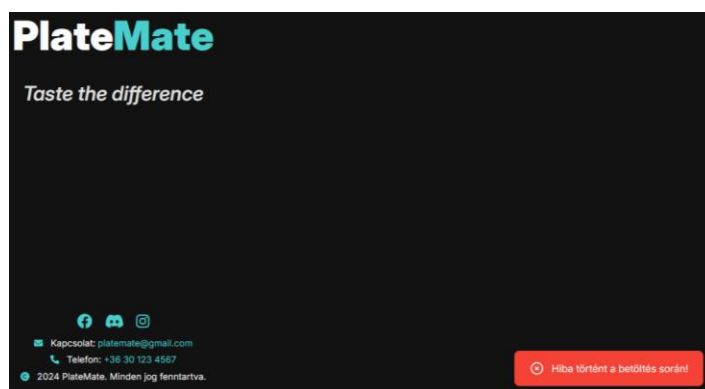


A felhasználót nem mindig egy pontos hibaüzenet fogja fogadni. Ez szintén így lett kitalálva.

Helytelen bejelentkezési adatokkal a következő hibaüzenet fogadja a felhasználót:



Lejárt vagy nem meglévő cookie-k estén a következő hibaüzenet fogja fogadni a felhasználót:



Ez az eset csak akkor tud életbe lépni, ha a felhasználó szándékosan kitörli a cookie-jait, amely esetben manuálisan vissza kell route-olnia a „/login” -ra



(localhost:5173/login). Ez szintén direkt nem lett kezelve, hiszen ha a felhasználó direkt kitörlni a cookie-jait egy aktív session-ben, akkor valószínűleg nem akarja a célnak megfelelően használni a szoftvert, így ebben az esetben a programnak nem kell eleget tennie a rosszindulatú tevékenységnek.

## **Információkérés**

Ha a program bármely részével kapcsolatban további bármilyen kérdése, felvetése van, a következő email címen keresztül léphet a PlateMate csapatával kapcsolatba: [bmate20050911@gmail.com](mailto:bmate20050911@gmail.com)

## IV. Összefoglalás, köszönetnyilvánítás

A PlateMate fejlesztése során egy innovatív éttermi rendszert hoztunk létre, amelynek célja a vendéglátóipar digitalizációjának támogatása és az éttermek mindennapi munkájának megkönnyítése volt. Mivel a projekt egy iskolai feladat része volt, a valós piaci bevezetés nem volt elsődleges célunk, azonban tudatosan törekedtünk arra, hogy egy olyan megoldást tervezzünk és kivitelezzünk, amely egy működő rendszer alapjául szolgálhatna. Tisztában vagyunk azzal, hogy a PlateMate jelenlegi formájában még fejlesztésre szorul, de a projektet nem csupán egy vizsgafeladatnak tekintjük, hanem hosszú távon is szeretnénk továbbfejleszteni és új funkciókkal bővíteni.

Az első fejezetben megfogalmazott szakmai célok teljesítését illetően utólag elmondhatjuk, hogy sikerült egy alapvetően funkcionális és felhasználóbarát rendszert kialakítani. A rendeléskezelés, a felhasználói interfész tervezése és az adatkezelés fő szempontok voltak a fejlesztés során.

A legnagyobb kihívást a rendszer stabilitásának biztosítása és a funkcionalitás összehangolása jelentette. Emellett az elérhető erőforrások korlátozottsága miatt bizonyos funkciókat csak alapverzióban tudtunk implementálni. Továbbá a csapatmunka összehangolása is nehézséget jelentett, mivel a fejlesztés során technikai problémák, különböző fejlesztői megközelítések és a közös munkafolyamatok összehangolása is kihívásokat okozott. Ezeket folyamatos kommunikációval és kompromisszumokkal igyekeztünk áthidalni.

A projekt során rengeteget tanultunk a modern web- és mobilalkalmazás-fejlesztési technológiákról, az adatbázis-optimalizálásról, valamint a szoftverfejlesztési folyamatok megtervezéséről. Különösen sokat fejlődtünk a frontend- és backend-fejlesztésben, valamint a Node.js és Vue.js használatában, amelyek kulcsfontosságú technológiákká váltak számunkra. Ezek az ismeretek nemcsak a PlateMate megvalósításában, hanem a jövőbeli projektjeink során is hasznosak lesznek.

A PlateMate jövőjét illetően tisztában vagyunk azzal, hogy a rendszer jelenlegi állapotában még számos fejlesztési lehetőséget rejt magában. Bár az eddig megvalósított funkciók működőképesek, további finomhangolás és bővítés szükséges ahhoz, hogy egy valóban piacképes megoldássá váljon. A projekt nem csupán egy vizsgafeladatnak készült, hanem hosszú távon is látunk benne lehetőséget. Szeretnénk a későbbiekben

továbbfejleszteni, új funkciókkal kiegészíteni, és akár egy éles környezetben is tesztelni annak érdekében, hogy valódi felhasználói igényeket is kiszolgálhasson.

Ezúton szeretnénk kifejezni köszönetünket mindazoknak, akik segítettek a PlateMate megvalósításában. Külön köszönet illeti oktatóinkat, akik szakmai útmutatásukkal hozzájárultak a projekt sikerességéhez. Hálásak vagyunk a csoporttársainknak és barátainknak is, akik értékes visszajelzéseikkel segítettek a rendszer fejlesztésében. Végül, de nem utolsó sorban, szeretnénk megköszönni családtagjainknak a támogatást, amit a projekt készítése során nyújtottak.

Bízunk benne, hogy a PlateMate projekt tanulságai hozzájárulnak további szakmai fejlődésünkhöz, és hogy a jövőben hasznosíthatjuk a megszerzett tudást.



## V. Irodalomjegyzék

A PlateMate többek között nem valósulhatott volna meg a következő oldalak segítségével, amelyek segítettek és lehetővé tették a munkánk gördülékeny haladását.

### Linkek

- Űrlap-elemek megvalósítása HTML-ben  
<http://infojegyzet.hu/webszerkesztes/html/urlapok/>
- Záródolgozat témák, szempontok, ötletek  
<http://infojegyzet.hu/webszerkesztes/zarodolgozat/>
- SQL szintaxis segédlet  
<https://www.w3schools.com/sql/>
- HTML szintaxis segédlet  
<https://www.w3schools.com/html/>
- JavaScript szintaxis segédlet  
<https://www.w3schools.com/js/>
- VUE hivatalos dokumentáció  
<https://vuejs.org/guide/>
- XAMPP Control Panel  
<https://www.apachefriends.org/hu/index.html>
- Node JS  
<https://nodejs.org/en/download>
- NPM parancsok dokumentációja  
<https://docs.npmjs.com/cli/v11/commands/>
- JavaScript: The Good Parts by Douglas Crockford  
<https://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742>
- HTML & CSS: The Complete Reference, Fifth Edition  
<https://www.amazon.com/HTML-CSS-Complete-Reference-Fifth/dp/0071496297>