1. **Vizsgázó neve:** Karita Árpád Ferenc
2. **Csoport tagjai:** Borbély Alex és Karita Árpád Ferenc
3. **Szoftveres fejlesztői környezet bemutatása: Visual Studio Code**
4. **Futtatás módja: Domain alapú futtatás**
5. **Vizsgaremek címe:**

,,Fundelio”

1. **Vizsgaremek témája: Kereskedés, befektetés szimuláció/játékos környezetben.**
2. **Témaválasztás indoklása: Mindkettőnknek van tapasztalata a kereskedés világában, és nem találtunk hasonló oldalt, ami jó**
3. **Funkciók leírása:**

**1. Backend funkciók (Szerveroldali funkciók)**

A backend a **Node.js**, **Express.js**, és **MSSQL** adatbázis segítségével van felépítve. Az API végpontok JSON formátumban válaszolnak a kliens oldali kérésre. Az autentikáció és az adatkezelés nagy része az **JWT (JSON Web Token)** technológiára épít, ami biztosítja a felhasználói adatok védelmét.

**a. Regisztráció funkció**

**API végpont**: /api/register

* A felhasználó a regisztrációs űrlapon megadja a nevét, email címét és jelszavát.
* A backend ellenőrzi, hogy az email cím már létezik-e az adatbázisban. Ha igen, hibaüzenetet küld vissza.
* Ha az email cím nem foglalt, akkor a jelszót **bcryptjs** segítségével hasheli, majd elmenti az adatokat a Felhasznalo táblába.
* Az új felhasználó egy **alapértelmezett egyenleggel** és **kriptovaluta** adatokkal (például: btcusdt: 0) kerül a FelhasznaloEgyenleg táblába, hogy a felhasználónak legyen kezdőtőkéje és kriptovaluta.
* Válaszként egy üzenetet küld vissza, hogy a regisztráció sikeres volt, és átirányítja a felhasználót a bejelentkezési oldalra.

**b. Bejelentkezés funkció**

**API végpont**: /api/login

* A felhasználó a bejelentkezési űrlapon megadja az email címét és a jelszavát.
* A backend ellenőrzi, hogy az email cím létezik-e az adatbázisban. Ha nem, akkor hibát jelez.
* Ha létezik, akkor **bcryptjs** segítségével ellenőrzi, hogy a megadott jelszó megegyezik-e a tárolt jelszóval.
* Ha a jelszó helyes, akkor a backend generál egy **JWT (JSON Web Token)** tokent, amelyet a frontend tárolni fog a **localStorage**-ban.
* A válaszban a backend visszaküldi a **token**-t, a felhasználó **nevét** és **email címét**.
* Sikeres bejelentkezés után a felhasználót átirányítja a főoldalra.

**c. Felhasználói adatok lekérése és frissítése**

**API végpont**: /api/userdata

* Ez a végpont biztosítja, hogy a bejelentkezett felhasználók hozzáférjenek a saját pénzügyi adataikhoz.
* A backend megkeresi a felhasználó adatokat a **FelhasznaloEgyenleg** táblában, és visszaküldi a válaszban az egyenleget, devizát, valamint a kriptovaluta és részvény mennyiségeket.
* A **POST** kérés lehetővé teszi, hogy a felhasználó frissítse az adatait (pl. egyenleg, kriptovaluták, részvények).

**d. Piaci adatok lekérése (Binance API és Twelve Data API)**

* **Binance WebSocket API**:
  + A backend egy **WebSocket kapcsolatot** létesít a **Binance API**-val, hogy folyamatosan frissítse a kriptovaluták árfolyamait, mint például a **BTC**, **ETH**, **DOGE**, **XRP** stb.
  + Az árfolyamokat a backend tárolja, és a frontend folyamatosan lekérheti őket az élő adatok lekérésére szolgáló végponton (/api/live/:symbol).
* **Twelve Data API**:
  + A backend a **Twelve Data API**-t használja a részvények árfolyamának lekérésére (például **SPY**, **MSFT**, **NVDA**).
  + A backend időszakosan lekéri az árakat, és azokat az API-n keresztül elérhetővé teszi a frontend számára.

**e. WebSocket kapcsolat a Binance API-hoz**

* A backend folyamatosan figyeli a **Binance WebSocket** adatfolyamát a kriptovaluták árfolyamainak frissítése céljából.
* A WebSocket adatfolyam frissíti az árakat és tárolja őket, hogy a frontend a legfrissebb adatokat használhassa.

**f. API végpontok**

* /api/register: A regisztrációs végpont.
* /api/login: A bejelentkezési végpont, amely token-t generál.
* /api/userdata: A felhasználó adatainak lekérése és frissítése.
* /api/live/:symbol: Az élő kriptovaluta árfolyam lekérése.
* /api/stocks: A részvények árfolyamának lekérése.

**2. Frontend funkciók (HTML, CSS, JavaScript)**

A frontend a felhasználói interakciókat és adatmegjelenítést kezeli. A frontend oldalak HTML, CSS és JavaScript segítségével épültek fel. Az oldalakat úgy alakították ki, hogy a felhasználók könnyedén tudjanak regisztrálni, bejelentkezni és kezelni a portfóliójukat.

**a. Regisztráció (HTML - regisztracio.html)**

* A felhasználó megadja a nevét, email címét és jelszavát.
* A frontend **userdata.js** segítségével a regisztrációs adatokat elküldi a backend /api/register végpontjára.
* Ha sikeres, akkor a felhasználót átirányítja a bejelentkezési oldalra.

**b. Bejelentkezés (HTML - bejelentkezes.html)**

* A felhasználó megadja a felhasználónevét és jelszavát.
* Az űrlap elküldi az adatokat a backend /api/login végpontjára.
* A backend válasza után a **JWT tokent** elmenti a frontend **localStorage**-ban, majd átirányítja a felhasználót a főoldalra.

**c. Profil oldal (HTML - profil.html)**

* A profil oldal megjeleníti a felhasználó nevét, email címét, egyenlegét, és más pénzügyi adatokat.
* A frontend a **userdata.js**-t használja az adat lekérésére a backendtől, és az adatokat dinamikusan jeleníti meg.
* Ha a felhasználó nincs bejelentkezve, akkor egy üzenet jelenik meg, amely felhívja a figyelmet a bejelentkezés szükségességére.

**d. Funkcionális JavaScript (userdata.js)**

A userdata.js kezelni a regisztrációt, bejelentkezést, kijelentkezést és a felhasználói adatokat:

* **Regisztráció**: A regisztrációs űrlap adatainak elküldése a backend /api/register végpontra.
* **Bejelentkezés**: A felhasználói adatok elküldése a backend /api/login végpontra, és a **token** tárolása a localStorage-ban.
* **Kijelentkezés**: A token és felhasználói adat eltávolítása a localStorage-ból, és átirányítás a bejelentkezési oldalra.
* **Profil megjelenítése**: A bejelentkezett felhasználó adatait (email, név, egyenleg) megjeleníti, ha azok elérhetőek a localStorage-ból.

**3. Adatbázis**

A projekt egy **MSSQL adatbázist** használ a felhasználói és pénzügyi adatok tárolására:

* Felhasznalo: A felhasználó személyes adatait tartalmazza, mint például a nevét, email címét és jelszavát.
* FelhasznaloEgyenleg: A felhasználó pénzügyi adatait tartalmazza, beleértve az egyenlegét, devizáját, kriptovalutáit és részvényeit.

**4. Biztonság és autentikáció**

A **JWT (JSON Web Token)** használata biztosítja, hogy a felhasználók autentikációja és adatkezelése biztonságos legyen. Az autentikációt az **authenticateToken** middleware végzi, amely ellenőrzi a bejelentkezett felhasználó jogosultságát, és csak akkor engedélyezi a hozzáférést a személyes adatokhoz.

**5. Piaci adatok:**

A rendszer a Binance WebSocket API és a Twelve Data API segítségével valós időben frissíti a kriptovaluták és részvények árfolyamait.

1. **A projektben melyik feladatrészt dolgozza ki?**

(…amely egyaránt tartalmaz frontend és backend részt, és ezek rövid leírása)

(A dokumentáció oldalszámai megjelölésével.)

(…melyek azok a részek, melyek minden csapattagnál ugyan az, és ezek kifejlesztése és dokumentálása hogy oszlik meg. Ha a fejlesztés és dokumentálás közös munka, azt is írd le!)

**Borbély Alex:**

**Frontend funckiók, backend integrálása frontendbe, dinamikus funckiók (26-31 oldal)**

**Karita Árpád Ferenc: Backend server fejlesztése, backend funkciók integrálása frontendbe. (2-26 oldal)**

1. **Adatbázis felépítése és kapcsolata a megvalósított funkciókkal**

A Fundelio alkalmazás adatbázisa a következő fő entitásokból áll:

1. **Felhasználó**
2. **FelhasználóEgyenleg**
3. **FelhasználóRészvények**
4. **GlobalSettings**
5. **Tranzakciók**

Minden entitás egy-egy táblának felel meg, és a köztük lévő relációk határozzák meg, hogy a különböző funkciók milyen adatokat olvasnak és írnak.

**10.1. Felhasználó (dbo.Felhasználó)**

| **Oszlop** | **Típus** | **Szerep a funkciókban** |
| --- | --- | --- |
| FelhasználóID (PK) | INT, NOT NULL | Egyedi azonosító – minden műveletnél ezzel dolgozunk |
| Név | NVARCHAR(100), NOT NULL | Felhasználó megjelenítése profiloldalon |
| Email | NVARCHAR(255), NOT NULL | Bejelentkezéshez és egyediség ellenőrzéséhez |
| Jelszó | NVARCHAR(255), NOT NULL | Regisztrációkor bcrypt-hashed, loginkor validáljuk |
| RegisztrációDátuma | DATETIME, NOT NULL | Statisztikai célokra, adminfelületen látható |

* **Regisztráció**: a POST /api/auth/register végpont új rekordot szúr be ebbe a táblába, bcrypt-pé hashed jelszóval.
* **Bejelentkezés**: a POST /api/auth/login lekérdezi az Email alapján, majd összeveti a tárolt jelszóhash-t.

**10.2. FelhasználóEgyenleg (dbo.FelhasználóEgyenleg)**

| **Oszlop** | **Típus** | **Szerep a funkciókban** |
| --- | --- | --- |
| EgyenlegID (PK) | INT, NOT NULL | Rekord egyedi azonosítója |
| FelhasználóID (FK) | INT, NOT NULL | Kapcsolat a felhasználóval – portfólióhoz tartozik |
| Egyenleg | FLOAT, NOT NULL | A maradék játékpénz – frissül minden tranzakciónál |
| Deviza | NVARCHAR(10), NULL | Pl. "HUF" – átváltáskor használatos |
| CryptoMennyiség | NVARCHAR(MAX), NULL | JSON-formátum: { "BTC": 0.5, "ETH": 2.1, … } |
| RészvényMennyiség | NVARCHAR(MAX), NULL | JSON-formátum: { "MSFT": 10, "NVDA": 5, … } |

* **Portfólió betöltése**: a frontend GET /api/market/portfolio hívás lekéri ennek a táblának a sorát, és a JSON-mezőkből építi fel a „Crypto” és „Stocks” listákat.
* **Tranzakció végrehajtás**: minden buy vagy sell művelet után a backend:
  1. Validálja a FelhasználóEgyenleg.Egyenleg értékét.
  2. Módosítja az Egyenleg mezőt a tranzakció típusa és ára szerint.
  3. A JSON mezőben frissíti a vásárolt/eladott eszköz mennyiségét.

**10.3. FelhasználóRészvények (dbo.FelhasználóRészvények)**

| **Oszlop** | **Típus** | **Szerep a funkciókban** |
| --- | --- | --- |
| FelhasználóStockID (PK) | INT, NOT NULL | Rekord egyedi azonosítója |
| FelhasználóID (FK) | INT, NOT NULL | Kapcsolat a Felhasználó táblával |
| Szimbólum | NVARCHAR(10), NOT NULL | Részvény ticker (pl. MSFT, SPY) |
| Mennyiség | DECIMAL(18,8), NOT NULL | Birtokolt részvény darabszám |

* **Részvény oldal**: a GET /api/market/portfolio végpont betölti a FelhasználóRészvények-ből az összes sorát.
* **Eladás**: a POST /api/market/trade endpoint sell esetén:
  1. Lekérdezi a jelenlegi Mennyiség értéket.
  2. Ellenőrzi, hogy a kért eladási mennyiség nem nagyobb‑e ennél.
  3. Csökkenti a Mennyiség mezőt, illetve ha nullára csökken, rekordot töröl.

**10.4. GlobalSettings (dbo.GlobalSettings)**

| **Oszlop** | **Típus** | **Szerep a funkciókban** |
| --- | --- | --- |
| ID (PK) | INT, NOT NULL | Általában csak egy rekord (ID = 1) |
| currency | NVARCHAR(10), NULL | Alap deviza (például "HUF") |
| cryptoQuantity | NVARCHAR(MAX), NULL | JSON: induló kriptó mennyiségek |
| stockQuantity | NVARCHAR(MAX), NULL | JSON: induló részvény mennyiségek |

* **Alapbeállítások**: a szerver indításakor a server.js betölti a GlobalSettings rekordot, és a memóriában tárolja az induló pénz‑ és eszközmennyiségeket.
* **Új felhasználó regisztrációja**: a POST /api/auth/register során ezzel inicializáljuk a FelhasználóEgyenleg rekord cryptoQuantity és stockQuantity mezőit.

**10.5. Tranzakciók (dbo.Tranzakciók)**

| **Oszlop** | **Típus** | **Szerep a funkciókban** |
| --- | --- | --- |
| TranzakcióID (PK) | INT, NOT NULL | Egyedi tranzakcióazonosító |
| FelhasználóID (FK) | INT, NOT NULL | Az elkövető felhasználó hivatkozása |
| Szimbólum | NVARCHAR(10), NOT NULL | Eszköz ticker (BTC, MSFT, stb.) |
| Mennyiség | DECIMAL(18,8), NOT NULL | Tranzakció darabszáma |
| Ár | DECIMAL(18,8), NOT NULL | Egy egység ára a tranzakció pillanatában |
| Típus | NVARCHAR(10), NOT NULL | “buy” vagy “sell” |
| Dátum | DATETIME, NOT NULL | Tranzakció végrehajtásának időpontja |

* **Audit-log**: minden sikeres kereskedéskor – vásárlásnál és eladásnál – új rekord jön létre itt.
* **Tranzakciós előzmények**: a frontend a GET /api/market/transactions végponttal kérdezi le a Tranzakciók táblát, és listázza a korábbi műveleteket.

**10.6. Működési példa: új felhasználó regisztrációja**

1. **Felhasználó létrehozása**
   * INSERT INTO dbo.Felhasználó …
2. **Egyenleg inicializálása**
   * Globális beállítások (cryptoQuantity, stockQuantity) betöltése
   * INSERT INTO dbo.FelhasználóEgyenleg(FelhasználóID, Egyenleg, Deviza, CryptoMennyiség, RészvényMennyiség) VALUES(…, 10000, 'HUF', '[…]', '[…]');

**10.7. Összefoglalás**

A táblák és mezőik gondos kialakítása lehetővé teszi, hogy az alkalmazásban minden funkció – regisztráció, bejelentkezés, portfólió‑betöltés, kereskedés, tranzakciós előzmények megjelenítése – egyszerű SQL‑lekérdezésekkel és frissítésekkel oldódjon meg, miközben az ER‑kapcsolatok garantálják az adatok integritását és könnyű bővíthetőségét.

A Vizsgaremeket mindenki a Git-en keresztül adja le, mindenkitől kérek meghívót az  
[sztedit@gmail.com](mailto:sztedit@gmail.com) vagy [szalaine.torok.edit@ganziskola.hu](mailto:szalaine.torok.edit@ganziskola.hu) és a [mehes.jozsef@ganziskola.hu](mailto:mehes.jozsef@ganziskola.hu) címekre. Ezt a dokumentumot is kitöltve töltsétek fel!

Ha sikerült hostolnod az alkalmazást, a README.md fájlban add meg az alkalmazás konfigurációit és futtatási lépéseit, a generált url-t, vagy ha van, akkor a saját domain nevet.

Az eredetiségi nyilatkozatot (Az iskolai meghajtó közös mappájában megtalálod) a végső leadáskor kinyomtatva, aláírva kell majd a kinyomtatott dokumentációhoz mellékelni.

A repository maradhat privát, és hozzáadhatod a vizsgáztatót, mint kollaborátort. Ehhez:

* Lépj be a repository-ba, és kattints a **"Settings"** fülre.
* A bal oldalon válaszd ki a **"Collaborators and Teams"** lehetőséget.
* Add meg a vizsgáztató GitHub-felhasználónevét vagy email címét, és kattints az **"Invite a collaborator"** gombra.

Az érintett személy meghívót kap, amelyet el kell fogadnia a hozzáférés érdekében.