Haladó programozás II. C#

Beadandó feladat - Funkcionális Specifikáció

1. Bevezetés

Ez a dokumentum egy oktatási céllal készülő kriptovaluta adás-vétel szimulátor backend funkcionális specifikációját tartalmazza. A rendszer lehetőséget biztosít a felhasználók számára, hogy egy virtuális pénztárcával kriptovalutákat vásároljanak, tartsanak meg, illetve adjanak el a piaci árfolyam változásának figyelembevételével.

A rendszer kizárólag backend funkcionalitásokat tartalmaz, a felhasználói interfész **Swagger-en keresztül** lesz elérhető. Tényleges bejelentkezés és autentikáció megvalósítás nem szükséges.

Leadási követelmények:

Connection string:

Az adatbázis eléréshez a connection string a következő legyen:

Server=(local);Database=CryptoDb_NEPTUN;Trusted_Connection=True;TrustServerCertificate=True;

A NEPTUN helyett mindenki a saját Neptun kódját használja.

Adatbázis kezdeti tartalma:

A tesztelés céljából a leadott kód mellé kérünk **VALÓSNAK TŰNŐ** teszt adatokat, amikkel az adatbázist a teszteléshez fel lehet tölteni. Ez lehet egy initdb.sql script, amit az adatbázison le lehet futtatni, vagy egy szerverbe épített szolgáltatás, ami észreveszi az üres adatbázist és feltölti adatokkal.

2. Technológiai környezet

- **Backend**: C# .NET 9
- **Adatbázis**: Microsoft SQL Server (MSSQL)
- **ORM**: Entity Framework Core
- **API dokumentáció**: Swagger

3. Főbb Funkcionalitások

3.1 Felhasználókezelés

A rendszer több felhasználós, így minden felhasználó egyedi azonosítóval, névvel és e-mail címmel rendelkezik. A felhasználók regisztrálhatnak és bejelentkezhetnek, valamint jelszavukat is módosíthatják.

CRUD műveletek:

- **Felhasználó regisztráció** (POST /api/users/register)
    - **Input**: { username, email, password }
    - **Validálás**: Az e-mail címnek egyedinek kell lennie.
- **Felhasználó adatainak lekérdezése** (GET /api/users/{userId})
- **Felhasználó adatainak frissítése** (PUT /api/users/{userId})
- **Felhasználó törlése** (DELETE /api/users/{userId})

3.2 Felhasználó Pénztárcája

A rendszer indításakor minden felhasználó kap egy virtuális pénztárcát egy kezdeti egyenleggel. A pénztárca tartalmazza a felhasználó egyenlegét és az általa birtokolt kriptovalutákat.

CRUD műveletek:

- **Pénztárca lekérdezése** (GET /api/wallet/{userId}) – Lekérdezi a felhasználó aktuális egyenlegét és kriptovalutáit.
- **Pénztárca egyenleg frissítése** (PUT /api/wallet/{userId}) – Az egyenleg manuális módosítása.
- **Pénztárca törlése** (DELETE /api/wallet/{userId})

3.3 Kriptovaluták kezelése

A rendszer **15 féle kriptovalutát** kezel, melyeknek előre definiált kezdő árfolyamuk és mennyiségük van. Az árfolyamok folyamatosan változnak, és minden kriptovalutának külön azonosítója van.

CRUD műveletek:

- **Kriptovaluták listázása** (GET /api/cryptos) – Listázza az összes elérhető kriptovalutát és azok aktuális árfolyamát.
- **Egy adott kriptovaluta lekérdezése** (GET /api/cryptos/{cryptoId})
- **Új kriptovaluta hozzáadása** (POST /api/cryptos)
- **Kriptovaluta törlése** (DELETE /api/cryptos/{cryptoId})

3.4 Kriptovaluta adás-vétel

A felhasználó a pénztárcájában lévő egyenlegből vásárolhat kriptovalutákat, illetve eladhatja azokat az aktuális piaci árfolyamon.

Műveletek:

- **Kriptovaluta vásárlása** (POST /api/trade/buy)
- **Kriptovaluta eladása** (POST /api/trade/sell)
- **Felhasználó portfóliójának lekérdezése** (GET /api/portfolio/{userId})

3.5 Árfolyam változás kezelése

A kriptovaluták árfolyamának változása dinamikusan történik egy **háttérszolgáltatáson keresztül**, amely külön szálon fut.

Műveletek:

- **Automatikus árfolyam frissítés**:
    - Egy időzített háttérszolgáltatás (Background Service) rendszeresen frissíti az árfolyamokat (pl. 30-60 másodpercenként).
    - Az árfolyamok kiszámításánál alkalmazhatók szabályok, mint például trendalapú mozgások vagy külső API-kból való lekérdezés *(opcionális)*.
- **Manuális árfolyam frissítés**:
    - Manuálisan is frissíthetők az árfolyamok:
        - **PUT /api/crypto/price**
        - **Input**: { cryptoId, newPrice }
- **Árfolyamváltozási naplózás**:
    - Minden árfolyamváltozás naplózásra kerül:
        - **GET /api/crypto/price/history/{cryptoId}**

3.6 Nyereség/Veszteség Számítása

A felhasználók nyomon követhetik portfóliójuk értékének változását az árfolyamok alapján.

Műveletek:

- **Profit/Veszteség kiszámítása egy adott pillanatban** (GET /api/profit/{userId})
    - (aktuális érték - vásárlási ár) * mennyiség minden kriptovalutára.
    - Az összegzés megmutatja a teljes portfólió nyereségét vagy veszteségét.
- **Profit/Veszteség részletes lebontásban** (GET /api/profit/details/{userId})
    - JSON válasz tartalmazza az egyes kriptovalutákra lebontott eredményeket.

3.7 Tranzakciók Naplózása

Minden vásárlás és eladás tranzakció naplózásra kerül, hogy a felhasználók visszanézhessék korábbi kereskedéseiket.

Műveletek:

- **Tranzakciók listázása** (GET /api/transactions/{userId})
    - A felhasználó korábbi tranzakcióinak visszakeresése.
    - Az adatok időrendi sorrendben tárolódnak.
- **Tranzakció részleteinek lekérdezése** (GET /api/transactions/details/{transactionId})

- Egy adott tranzakció részletes megjelenítése *(árfolyam, mennyiség, időbélyeg)*.

4. Záró gondolatok

A rendszer egy **oktatási célú kriptovaluta kereskedési szimulátor**, amely lehetőséget biztosít a felhasználók számára a piaci mozgások megismerésére.