**BGÉSZC GANZ ÁBRAHÁM** **KÉT TANÍTÁSI NYELVŰ TECHNIKUM**

**Fundelio**

**Vizsgaremek**

Készítette: Karita Árpád Ferenc  
 Borbély Alex

Témavezető: Szalainé Török Edit

Méhes József

Budapest, 2025

Tartalomjegyzék

### 1.

[Tartalomjegyzék 1](#_Toc197539897)

[1. 1](#_Toc197539898)

[1. Bevezetés 4](#_Toc197539899)

[1.1. Témaválasztás indoklása 4](#_Toc197539900)

[1.2. Célkitűzés 4](#_Toc197539901)

[1.3. A célcsoport és felhasználói kör 5](#_Toc197539902)

[1.4. A projekt történeti és piaci háttere 5](#_Toc197539903)

[2. Fejlesztői Dokumentáció 6](#_Toc197539904)

[2.1. Fejlesztőkörnyezet: Hardver- és szoftverkövetelmények 6](#_Toc197539905)

[2.2. Választott technológiák és eszközök indoklása 7](#_Toc197539906)

[2.3. A projekt felépítésének áttekintése 8](#_Toc197539907)

[2.3.1. Backend struktúra és működés 8](#_Toc197539908)

[2.3.2. Frontend struktúra és kommunikáció 9](#_Toc197539909)

[2.4. Az adatbázis architektúrája 10](#_Toc197539910)

[2.4.1. Adatbázis táblák, mezők és kapcsolatok 10](#_Toc197539911)

[2.4.2. Kapcsolatok és ER‑diagram 12](#_Toc197539912)

[2.5. Modulok, komponensek és a kódban megoldott algoritmusok 13](#_Toc197539913)

[2.5.1. Backend algoritmusok és moduláris felépítés 13](#_Toc197539914)

[2.5.2. Frontend interakciók és token kezelési logika 27](#_Toc197539915)

[A fájl működése lépésről lépésre 29](#_Toc197539916)

[2.6. Biztonsági megoldások és hitelesítési mechanizmusok 33](#_Toc197539917)

[2.7. Rendszerintegráció, API-k és külső szolgáltatások 33](#_Toc197539918)

[2.8. Hibakezelés és naplózás 34](#_Toc197539919)

[2.9. Tranzakciók kezelése 34](#_Toc197539920)

[2.10 Tranzakciólista lekérdezése 38](#_Toc197539921)

[3. Tesztelési Dokumentáció 44](#_Toc197539922)

[3.1. A tesztelési stratégia és célok 44](#_Toc197539923)

[3.2. Manuális tesztek és funkcionális ellenőrzések 44](#_Toc197539924)

[3.3. Teljesítménytesztelés, stresszteszt és terheléses vizsgálatok 45](#_Toc197539925)

[3.4. Tesztelési eredmények és hibakeresési folyamat 45](#_Toc197539926)

[3.5. Tesztelésben észlelt hibák 46](#_Toc197539927)

[4. Fejlesztési Lehetőségek és Jövőbeni Bővítések 47](#_Toc197539928)

[4.1. Új funkciók és integrációk 47](#_Toc197539929)

[4.2. Rendszer skálázhatósága és optimalizációs lehetőségek 47](#_Toc197539930)

[4.3. UX/UI fejlesztések és felhasználói visszajelzések beépítése 48](#_Toc197539931)

[4.4. Biztonsági fejlesztések és rendszeres auditok 48](#_Toc197539932)

[**5.** Felhasználói Dokumentáció 49](#_Toc197539933)

[6. Összefoglalás, Értékelés és Köszönetnyilvánítás 54](#_Toc197539934)

[**6.1.** Összefoglalás 54](#_Toc197539935)

[6.2. Értékelés 54](#_Toc197539936)

[6.3. Köszönetnyilvánítás 55](#_Toc197539937)

[**8.** Függelék 56](#_Toc197539938)

[8.2. Diagramok és Adatbázis Sémák 56](#_Toc197539939)

[9. Fundelio – Főoldal HTML Oldal Dokumentáció 58](#_Toc197539940)

[9.1 Főoldal felépítése 58](#_Toc197539941)

[9.3.1 Navigációs Sáv (Navbar) 58](#_Toc197539943)

[9.3.2 Oldalsáv 59](#_Toc197539944)

[9.4. Tartalmi Rész 60](#_Toc197539945)

[9.5. Dinamikus Funkciók 60](#_Toc197539946)

[9.6. Stílus és Téma Kezelés 60](#_Toc197539947)

[9.7. Felhasználói Élmény és Cél 60](#_Toc197539948)

[10. CSS Stíluslapok 62](#_Toc197539949)

1. Bevezetés

1.1. Témaválasztás indoklása

A Fundelio projekt indokolása több szempontból is releváns. A fő célkitűzés az volt, hogy egy kockázatmentes, de élethű pénzügyi szimulációs platformot hozzunk létre, amely lehetőséget ad a felhasználóknak arra, hogy gyakorolják a kereskedési folyamatokat. Ezt az ötletet a következő tényezők motiválták:

* **Oktatási célok:** A pénzügyi piacok működésének megismertetése a felhasználókkal.
* **Biztonságos tanulási környezet:** Mivel a kereskedés valódi pénz kockázatával járna, a játékszintű megoldás segít megismerni a piac dinamikáját anélkül, hogy pénzügyi kockázatot vállalnánk.
* **Személyes érdeklődés:** A projekt mögött személyes tapasztalatok és érdeklődés álltak a kriptovaluták, valamint a részvénykereskedelem iránt.
* **Piaci igény:** A meglévő, angol nyelvű és bonyolult felületek helyett egy könnyen érthető, magyar nyelvű platform készítése.

A választott téma tehát egyszerre szolgál oktatási, kísérleti és piackutatási célokat, valamint lehetőséget ad arra, hogy a fejlesztők kipróbálják a modern webfejlesztési technológiákat.

1.2. Célkitűzés

A Fundelio projekt célja, hogy:

* **Könnyen használható felületet:** Olyan webes alkalmazást fejlesszünk, amely intuitív felhasználói élményt nyújt, és nem támaszkodik bonyolult műszaki kifejezésekre, így a kezdők is könnyen megérthetik.
* **Kockázatmentes kereskedés:** A felhasználók játékpénzzel léphetnek be a kereskedési folyamatba, ezáltal kockázat nélkül gyakorolhatnak.
* **Valós idejű adatok integrációja:** A Binance és Twelve Data API-kon keresztül megoldott valós idejű árfolyam adatok alapján működik a rendszer.
* **Oktatási érték:** A projekt nem csupán a technikai megoldásokat mutatja be, hanem a pénzügyi gondolkodásmód fejlesztését is segíti, a kockázatkezelés, stratégiaalkotás és türelem fontosságát hangsúlyozva.
* **Modularitás és skálázhatóság:** A kód és az architektúra úgy készült, hogy a jövőben könnyen bővíthető legyen új funkciókkal, kriptovalutákkal vagy részvényekkel, továbbá a rendszer könnyen integrálható további platformokkal.

1.3. A célcsoport és felhasználói kör

A Fundelio platform több rétegből álló felhasználói csoportot céloz meg:

* **Kezdők:** Olyan felhasználók, akik először ismerkednek a kereskedés világával, és szeretnék megérteni a piac működését valós adatok alapján.
* **Diákok és oktatók:** Iskolai vagy egyetemi környezetben érdeklődők, akik pénzügyi tanulmányokat folytatnak, valamint oktatók, akik interaktív módszerekkel kívánják bemutatni a piac folyamatát.
* **Haladó kereskedők:** Bár a platform elsődlegesen a kezdőknek készült, a haladó felhasználók is hasznosíthatják a rendszerben található statisztikákat és valós idejű árfolyam adatokat a stratégia finomításában.
* **Pályakezdők fejlesztők:** A projekt fejlesztői dokumentációja segít abban, hogy azok, akik a szoftvermérnöki karrierjük elején járnak, a modern webfejlesztési technológiákat és a rendszerintegrációs megoldásokat gyakorolják.

1.4. A projekt történeti és piaci háttere

A Fundelio projekt elindítása nem véletlen: a pénzügyi piacok digitalizációja és a blockchain-technológiák térnyerése egyben új kihívásokat és lehetőségeket is teremtett a kereskedési szimulációk terén. A platform létrejöttét megelőzően a csapattagok különböző valós piacokon szerzett tapasztalataik, mint például sikeres és sikertelen befektetések, inspirációt szolgáltattak a projekt kialakításához.

A piacon több hasonló alkalmazás is elérhető volt (pl. TradingView, eToro, Binance demo mód), de ezek gyakran angol nyelvűek, túlbonyolítottak vagy nem kizárólag oktatási célokra lettek tervezve. Így a Fundelio egyedülállóan ötvözi a valós idejű adatok feldolgozását, a játékos tanulási élményt és az egyszerű, de hatékony felületet.

2. Fejlesztői Dokumentáció

Ebben a részben részletesen ismertetjük a projekt technikai hátterét, a választott technológiákat, a kód architektúráját és a főbb algoritmusokat, amelyek biztosítják a rendszer működését.

2.1. Fejlesztőkörnyezet: Hardver- és szoftverkövetelmények

**Hardver**

* **Fejlesztési gép:**
  + Processzor: Legalább Intel i5 vagy ekvivalens AMD processzor
  + Memória: Minimum 8 GB RAM (ajánlott 16 GB a zökkenőmentes fejlesztéshez)
  + Tároló: SSD meghajtó a gyorsabb indítás és adatfeldolgozás érdekében
* **Tesztkörnyezet:**
  + Különböző platformokon (Windows, macOS, Linux) való fejlesztésre optimalizált konfigurációk
  + Mobil eszközök az alkalmazás reszponzivitásának teszteléséhez

**Szoftver**

* **Fejlesztői eszközök:**
  + **IDE:** Visual Studio Code
  + **Verziókövetés:** Git alapú rendszer, GitHub repository használata
  + **Környezeti menedzsment:** Node.js (minimum v14 vagy újabb), npm csomagkezelők
* **Szerveroldali környezet:**
  + Node.js futtatókörnyezet az Express alapú backend számára
  + Microsoft SQL Server a FundelioDB adatbázis futtatásához
* **Frontend eszközök:**
  + Böngészők: Chrome, Firefox, Safari, Edge (a kompatibilitás tesztelése érdekében)
  + JavaScript, HTML5, CSS3 alapú fejlesztési eszközök

2.2. Választott technológiák és eszközök indoklása

A Fundelio projekt fejlesztése során számos modern technológiát alkalmaztunk annak érdekében, hogy egy jól moduláris, bővíthető és biztonságos rendszert hozzunk létre. A választásunkat a következő szempontok alapján határoztuk meg:

* **Közösségi támogatás és dokumentáció:**  
  A Node.js és Express rendkívül széleskörű dokumentációval, aktív közösséggel és számos elérhető moduláris megoldással rendelkezik, ami megkönnyíti a fejlesztést.
* **Teljesítmény és skálázhatóság:**  
  A Node.js aszinkron működése és a modern adatbázis-kezelő rendszer (MSSQL) lehetővé teszi a magas terhelés hatékony kezelését.
* **Biztonság:**  
  Az olyan modulok, mint a bcryptjs a jelszavak hashelésére, és a jsonwebtoken az autentikációhoz biztosítják a korszerű biztonsági intézkedéseket, amelyek alapvetőek a pénzügyi alkalmazásoknál.
* **Valós idejű adatkommunikáció:**  
  A WebSocket alapú kommunikáció a Binance API-val lehetővé teszi az élő adatfolyamok kezelését, míg az axios segít a REST API hívásokban, így a rendszer folyamatosan naprakész marad.

**Alternatívák és miért nem választottuk őket**

* **Python + Flask/Django**: Bár gyors fejlesztést tesz lehetővé, a Python GIL korlátozza az egyidejű I/O-t, és külön nyelvi ismeretekre van szükség a frontendhez képest.
* **Java + Spring Boot**: Magasabb memória- és erőforrásigény, nagyobb konfigurációs overhead, lassabb prototípus-készítés.
* **Blazor (ASP.NET Core)**: Teljes C#/.NET-alapú full-stack fejlesztést tesz lehetővé, a csapat rendelkezik elegendő C#/.NET tapasztalattal, azonban a Node.js egyszerűbb telepítést és gyorsabb prototípus-készítést tesz lehetővé, valamint kisebb memória- és erőforrásigénye van konténer környezetben.

2.3. A projekt felépítésének áttekintése

A Fundelio architektúrája két fő részből áll: a backend (szerveroldal) és a frontend (felhasználói interakciók). Az alábbiakban részletesen ismertetjük mindkét réteget.

2.3.1. Backend struktúra és működés

A backend fő fájlja a **server.js**, amely a Node.js és Express keretrendszert használva szolgáltatja ki az API-kat, kezeli a felhasználói hitelesítést és a valós idejű adatfolyamokat. A szerver architektúrája a következő főbb modulokból áll:

1. **Külső modulok importálása:**  
   Az első szakaszban történik az olyan külső modulok betöltése, mint az express, ws, path, cors, bcryptjs, jsonwebtoken, nodemailer, mssql, exec és axios. Ezek a modulok biztosítják a HTTP-szerver futtatását, WebSocket kapcsolatot az élő adatfolyamokhoz, valamint az adatbázis és más külső források elérését.
2. **Alapbeállítások és konfiguráció:**
   * Az Express app inicializálása
   * Port beállítása, mely általában 3000, de környezeti változóktól függően változhat
   * A titkos kulcs definiálása a JWT tokenek aláírásához
3. **MSSQL adatbázis-kapcsolat:**  
   A server.js konfigurálja az adatbázis elérését az MSSQL kliens segítségével, amely a FundelioDB nevű adatbázishoz kapcsolódik. A konfiguráció tartalmazza a host címét, adatbázis nevét, hitelesítési adatokat és egyéb szükséges paramétereket.
4. **Middleware-ek és statikus fájlok kiszolgálása:**  
   Az Express alkalmazás egy sor middleware-t használ:
   * **CORS Middleware:** A cross-origin kérések engedélyezésére
   * **Statikus fájlok kezelése:** A /Pages, /css és /js könyvtárakból kerülnek kiszolgálásra a HTML, CSS és JavaScript fájlok.
   * **Token-alapú hitelesítés:** Egy olyan middleware, amely ellenőrzi az API-khoz érkező kérések JWT tokenjét.
5. **RESTful API végpontok:**
   * **Regisztrációs végpont (/api/register):** Ellenőrzi az új felhasználó adatait, hasheli a jelszót, és elmenti az adatbázisba.
   * **Bejelentkezési végpont (/api/login):** Ellenőrzi a beérkezett adatokat, és sikeres azonosítás esetén JWT token-t generál.
   * **Kriptovaluta ár lekérdezés:** Egy dedikált API, amely a Binance WebSocket adatfolyamából olvassa ki az aktuális árfolyamokat.
   * **Részvényár lekérdezés:** Az axios segítségével hívja a Twelve Data API-t, és visszaadja az aktuális részvényárakat.
6. **Globális beállítások betöltése:**  
   A rendszer indulásakor lefutó szekció, amely betölti a GlobalSettings táblából az alapértelmezett pénznemet, illetve az induló játékpénzek mennyiségét, így biztosítva az alkalmazás egyedi konfigurációját.
7. **Szerverindítás:**  
   A konfigurációk és a globális beállítások betöltése után indul el a szerver, biztosítva a teljes rendszer elérhetőségét.

2.3.2. Frontend struktúra és kommunikáció

A frontend a statikus HTML, CSS, JavaScript fájlok kombinációjából áll, melyeket a szerver kiszolgál. A **userdata.js** egy kulcsfontosságú JavaScript fájl, mely az alábbi feladatokat látja el:

1. **Regisztráció és bejelentkezés kezelése:**  
   Az űrlapok elküldésekor a JavaScript kód összegyűjti a felhasználói adatokat, majd fetch API hívások segítségével POST kéréseket küld a backend regisztrációs és bejelentkezési végpontjaira.
   * Az űrlap adatainak validálása
   * A jelszó megfelelő hashelése (a backend gondoskodik a bcryptjs által végzett hash-elésről)
   * A válaszként érkező JWT token mentése a böngésző localStorage-jébe
2. **Kijelentkezés mechanizmusa:**  
   A felhasználó a kijelentkezés gombra kattintva törli a token-t és a felhasználói adatokat a localStorage-ből, majd automatikusan átirányítja magát a bejelentkezési felületre.
3. **Profiladatok betöltése és megjelenítése:**  
   A DOMContentLoaded esemény után a script betölti a localStorage-ben eltárolt felhasználói információkat, és azokat megjeleníti a megfelelő HTML elemekben, így minden oldalon friss adatokat biztosítva.
4. **Egyéb interakciók:**  
   Olyan kisebb, de fontos funkciók, mint például a hibakezelő üzenetek, felhasználói visszajelzések megjelenítése, valamint a kérések státuszának indikációja (pl. loading spinner-ek) is szerepelnek a kódban.

2.4. Az adatbázis architektúrája

A Fundelio adatbázisa SQL Server alatt fut, és a következő fő táblákat tartalmazza. Minden tábla kulcsa és kapcsolatai világosan lekövetik a felhasználók, a tranzakciók, az egyenlegek és a globális beállítások közötti viszonyt.

2.4.1. Adatbázis táblák, mezők és kapcsolatok

1. **dbo.Felhasználó**  
   A rendszer felhasználóinak alapadatait tartalmazza.

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| FelhasználóID | INT (PK, NOT NULL) | Egyedi felhasználói azonosító |
| Név | NVARCHAR(100), NOT NULL | Felhasználó teljes neve |
| Email | NVARCHAR(255), NOT NULL | Egyedi e‑mail cím |
| Jelszó | NVARCHAR(255), NOT NULL | Bcrypt hashelt jelszó |
| RegisztrációDátuma | DATETIME, NOT NULL | Regisztráció ideje |

1. **dbo.FelhasználóEgyenleg**  
   A játékpénzes egyenleget és devizát tartalmazza felhasználónként.

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| EgyenlegID | INT (PK, NOT NULL) | Egyedi egyenleg rekordazonosító |
| FelhasználóID | INT (FK → Felhasználó) | Hivatkozás a Felhasználóra |
| Egyenleg | FLOAT, NOT NULL | Játékpénzes egyenleg összege |
| Deviza | NVARCHAR(10), NULL | Alapértelmezett devizanem (pl. HUF) |
| CryptoMennyiség | NVARCHAR(MAX), NULL | JSON vagy lista formátumban a kriptók mennyiségei |
| RészvényMennyiség | NVARCHAR(MAX), NULL | JSON vagy lista formátumban a részvények mennyiségei |

1. **dbo.FelhasználóRészvények**  
   Felhasználói részvény‑pozíciók tárolása (ha külön szeretnénk a balance‑tól).

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| FelhasználóStockID | INT (PK, NOT NULL) | Rekordazonosító |
| FelhasználóID | INT (FK → Felhasználó) | Hivatkozás a Felhasználóra |
| Szimbólum | NVARCHAR(10), NOT NULL | Részvény ticker (pl. MSFT, NVDA) |
| Mennyiség | DECIMAL(18,8), NOT NULL | Birtokolt részvénymennyiség |

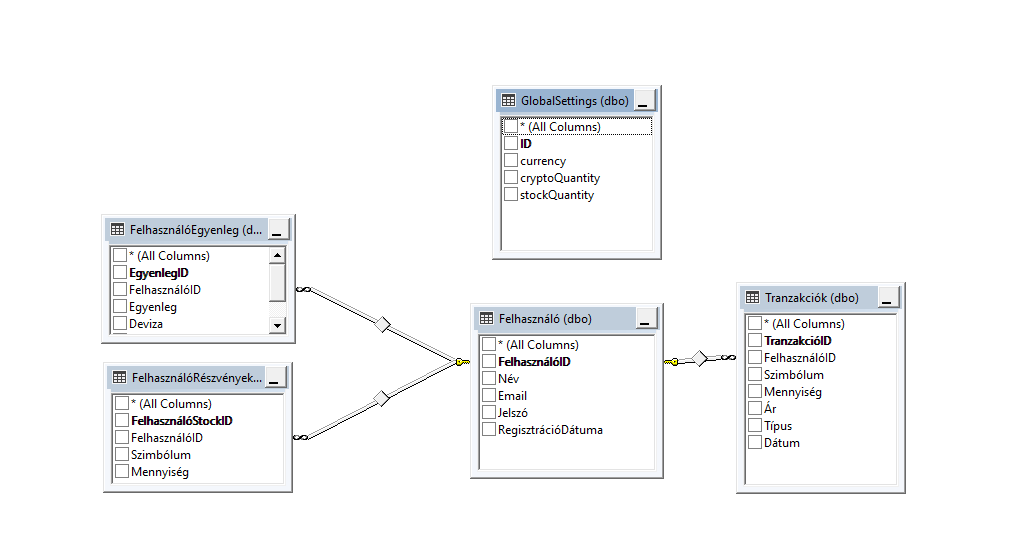
1. **dbo.GlobalSettings**  
   A rendszer általános konfigurációs paraméterei.

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| ID | INT (PK, NOT NULL) | Rekordazonosító (pl. mindig 1) |
| currency | NVARCHAR(10), NULL | Alapértelmezett deviza (pl. HUF, USD) |
| cryptoQuantity | NVARCHAR(MAX), NULL | Induló játék‑kripto mennyiség (JSON lista) |
| stockQuantity | NVARCHAR(MAX), NULL | Induló játék‑részvény mennyiség (JSON lista) |

1. **dbo.Tranzakciók**  
   Minden vásárlási/​eladási esemény naplózása.

| **Oszlop** | **Típus** | **Leírás** |
| --- | --- | --- |
| TranzakcióID | INT (PK, NOT NULL) | Egyedi tranzakcióazonosító |
| FelhasználóID | INT (FK → Felhasználó) | A tranzakciót indító felhasználó |
| Szimbólum | NVARCHAR(10), NOT NULL | Érintett eszköz ticker (BTC, SPY, stb.) |
| Mennyiség | DECIMAL(18,8), NOT NULL | Vétel/eladás mennyisége |
| Ár | DECIMAL(18,8), NOT NULL | Egy egység aktuális ára |
| Típus | NVARCHAR(10), NOT NULL | „buy” vagy „sell” |
| Dátum | DATETIME, NOT NULL | Tranzakció végrehajtásának időpontja |

2.4.2. Kapcsolatok és ER‑diagram

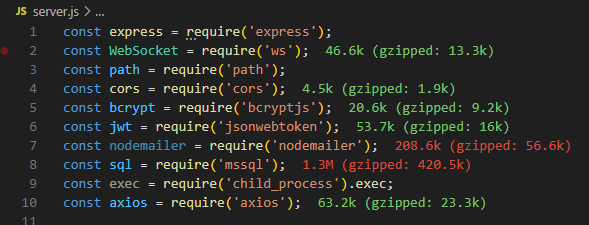
* Felhasználó 1– FelhasználóEgyenleg\*:  
  Egy felhasználóhoz egy vagy több egyenleg rekord tartozhat (általában csak egy, de külön tarthatjuk devizák szerint).
* Felhasználó 1– FelhasználóRészvények\*:  
  A felhasználó több különböző részvénypozíciót tarthat.
* Felhasználó 1– Tranzakciók\*:  
  Minden tranzakció a FelhasználóID-re hivatkozik.
* **GlobalSettings**:  
  Általában csak egy rekord (ID = 1), amelyből minden modul betölti az induló mennyiségeket, devizát.
* 

2.5. Modulok, komponensek és a kódban megoldott algoritmusok

Ebben a szakaszban részletezzük az egyes modulok és komponensek implementációs részleteit, továbbá ismertetjük azokat a speciális algoritmusokat, amelyek a rendszer működését garantálják.

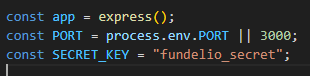
2.5.1. Backend algoritmusok és moduláris felépítés

**Külső modulok betöltése és inicializálása:**  
A server.js elején történik az összes szükséges külső modul importálása, melyek között szerepel:



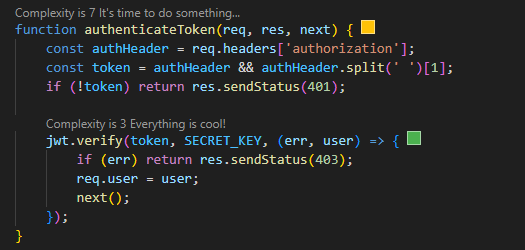
* **express – HTTP szerver és API-k létrehozása.**
* **ws – WebSocket kapcsolat a Binance élő árfolyamadatainak figyeléséhez.**
* **path – Fájlrendszer-útvonalak kezeléséhez, főleg statikus fájlok betöltésére.**
* **cors – Cross-Origin Resource Sharing engedélyezés frontend számára.**
* **bcryptjs – Jelszavak biztonságos tárolásához szükséges hash-elés.**
* **jsonwebtoken – Token-alapú azonosításhoz (JWT).**
* **nodemailer – (jelenleg nem használva, de valószínűleg jelszó-emlékeztető vagy regisztrációhoz e-mail küldéshez van bekészítve).**
* **mssql – Microsoft SQL Server adatbázis eléréséhez.**
* **exec – Shell parancsok futtatása a szerverről.**
* **axios – REST API-k hívására (pl. részvényadatokhoz).**

**Alapbeállítások és konfiguráció**

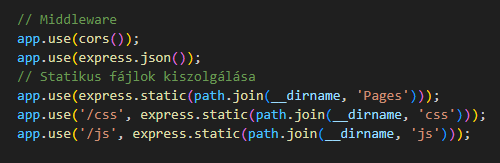


Ez inicializálja az Express appot, beállítja a portot (környezetfüggően vagy 3000-re), valamint definiál egy titkos kulcsot a JWT aláíráshoz.

**Token validációs middleware:**  
A middleware egy olyan funkció, amely minden védett API végpont előtt fut le. Ellenőrzi a kérés fejlécében található JWT token érvényességét, majd az ellenőrzés után a felhasználói adatokat csatolja a kérés objektumához, így biztosítva, hogy a további feldolgozás során azonosítani tudjuk a kérelmezőt.  
*Példa:*



**Middleware és statikus fájlok**



Ez biztosítja a frontend fájlok kiszolgálását. A /Pages mappából jönnek a HTML oldalak, a css és js mappák pedig külön route-on keresztül érhetőek el.

**5. Regisztrációs végpont**



**URL**  
POST /api/register

**Leírás**  
Új felhasználó létrehozása az adatbázisban, majd alapértelmezett pénzügyi adatok (kezdő egyenleg, üres kripto‑ és részvényállomány) beállítása.

**Kérés**

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| name | String | A felhasználó teljes neve |
| email | String | Egyedi e‑mail cím |
| password | String | Jelszó (plain text, bcrypt-tal hash‑elve) |

Pl:   
 {

"name": "Kiss Péter",

"email": "peter.kiss@example.com",

"password": "mySecurePassword123"

}

**Működési folyamat**

1. **Adatbázis-kapcsolat létrehozása**  
   A dbConfig beállításokkal mssql pool példányt hozunk létre.
2. **E‑mail egyediségének ellenőrzése**  
   Lekérdezzük a Felhasználó táblát, hogy létezik‑e már rekord az adott e‑mail címmel.
   * Ha igen → HTTP 400, { message: 'Ez az e-mail már foglalt.' }
3. **Jelszó hashelése**  
   A bcrypt.hash(password, 10) függvénnyel biztonságos hash-t készítünk.
4. **Új felhasználó beszúrása**  
   A Felhasználó táblába kerül a Név, Email és a hash-elt Jelszó.  
   A beszúrást követően lekérjük az újonnan generált FelhasználóID-t.
5. **Kezdő egyenleg beállítása**  
   A FelhasználóEgyenleg táblába írjuk a következő értékeket:
   * FelhasználóID: az előző lépésben kapott azonosító
   * Egyenleg: 10000
   * Deviza: "USD"
   * CryptoMennyiség: üres objektum ({} JSON-ként)
   * RészvényMennyiség: üres objektum ({} JSON-ként)
6. **Válasz küldése**  
   Siker esetén HTTP 201, { message: 'Sikeres regisztráció!' }  
   Hiba esetén HTTP 500, { message: 'Szerverhiba regisztráció közben.' }

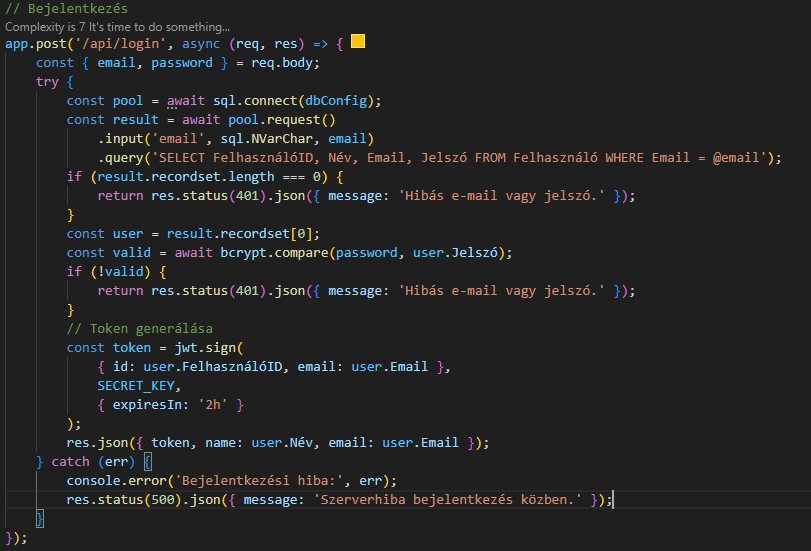
**Válaszok**

| **HTTP státusz** | **Törzs** | **Magyarázat** |
| --- | --- | --- |
| **201** | { message: 'Sikeres regisztráció!' } | Felhasználó sikeresen létrehozva |
| **400** | { message: 'Ez az e-mail már foglalt.' } | Email cím már foglalt |
| **500** | { message: 'Szerverhiba regisztráció közben.' } | Váratlan hiba a szerveren |

**Megjegyzések**

* A jelszó sosem kerül plaintextben az adatbázisba, mindig hash-elt formában tároljuk.
* A kezdő egyenleg és üres portfólió értékek az üzleti logika részei; szükség esetén konfigurálhatóak.
* Hibakezeléskor minden váratlan kivételt naplózunk (console.error), de a kliensnek általános hibaüzenetet küldünk vissza.

**6. Bejelentkezés**



**URL**  
POST /api/login

**Leírás**  
Felhasználó hitelesítése e‑mail és jelszó alapján, majd JWT token generálása.

**Kérés**

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| email | String | Regisztrált e‑mail cím |
| password | String | Plain text jelszó |

Pl:

{

"email": "peter.kiss@example.com",

"password": "mySecurePassword123"

}

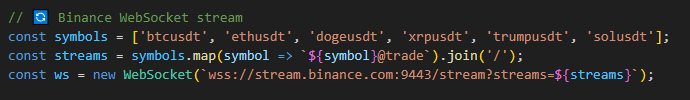
**Működési folyamat**

1. **Adatbázis-kapcsolat létrehozása**  
   A dbConfig beállításokkal mssql pool példányt hozunk létre.
2. **Felhasználó lekérdezése e‑mail alapján**  
   A Felhasználó táblából lekérdezzük a FelhasználóID, Név, Email és a hash-elt Jelszó mezőket.
   * Ha nincs találat → HTTP 401, { message: 'Hibás e-mail vagy jelszó.' }
3. **Jelszó ellenőrzése**  
   A bcrypt.compare(password, user.Jelszó) függvénnyel validáljuk a beérkező jelszót a tárolt hash ellen.
   * Sikertelen egyezés → HTTP 401, { message: 'Hibás e-mail vagy jelszó.' }
4. **JWT token generálása**  
   A jwt.sign segítségével hozunk létre egy 2 óráig érvényes tokent, ami tartalmazza:
   * id: FelhasználóID
   * email: felhasználó e‑mail címe  
     A titkos kulcsot a SECRET\_KEY változó tartalmazza.
5. **Válasz küldése**  
   Sikeres hitelesítés esetén JSON válaszban visszaküldjük:
   * token: a JWT
   * name: a felhasználó neve
   * email: a felhasználó e‑mail címe
6. **Hibakezelés**  
   Bármely váratlan kivétel esetén logoljuk a hiba részleteit (console.error), és HTTP 500-as választ adunk: { "message": "Szerverhiba bejelentkezés közben." }
7. **Válaszok**

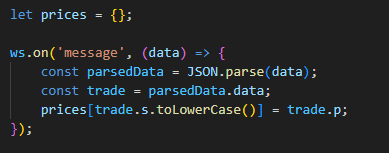
| **HTTP státusz** | **Törzs** | **Magyarázat** |
| --- | --- | --- |
| **200** | { "token": "...", "name": "Kiss Péter", "email": "peter.kiss@…" } | Sikeres bejelentkezés, visszaadja a JWT-t és felhasználói adatokat |
| **401** | { "message": "Hibás e-mail vagy jelszó." } | Hibás e‑mail vagy jelszó |
| **500** | { "message": "Szerverhiba bejelentkezés közben." } | Váratlan szerverhiba |

1. **Megjegyzések**
   * A kliensoldalnak a token tárolásáról (pl. localStorage vagy httpOnly süti) és a további kérések során az Authorization: Bearer <token> fejléc beállításáról kell gondoskodnia.
   * A token lejárta után újra be kell jelentkezni, vagy opcionálisan refresh token mechanizmust lehet implementálni.
   * A hibaüzenetek biztonsági okokból nem árulnak el részleteket a hitelesítési folyamatról.

**7. Binance WebSocket stream**

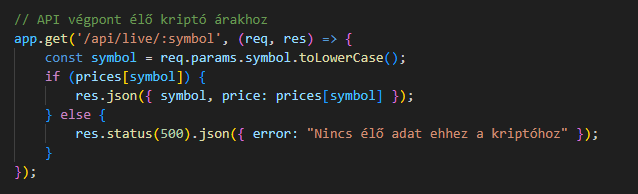


* Élő adatfolyam indul Binance-ről a megadott kriptovaluták kereskedési eseményeiről (BTCUSDT, ETHUSDT stb.).
* Minden kereskedési eseménynél (trade) elmenti az aktuális árat a prices objektumba.

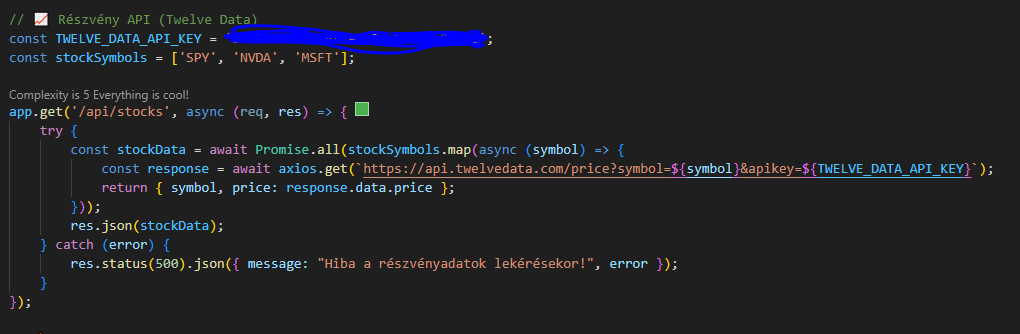


Ezzel élő kriptóárakat tarthatsz karban a szerveren.

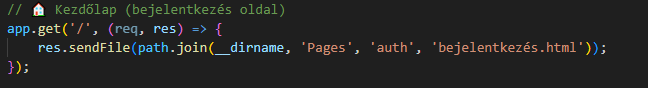
**8. API élő kriptóár lekérdezéshez**



Ez az API végpont adja vissza egy adott kriptó élő árát a prices objektumból.

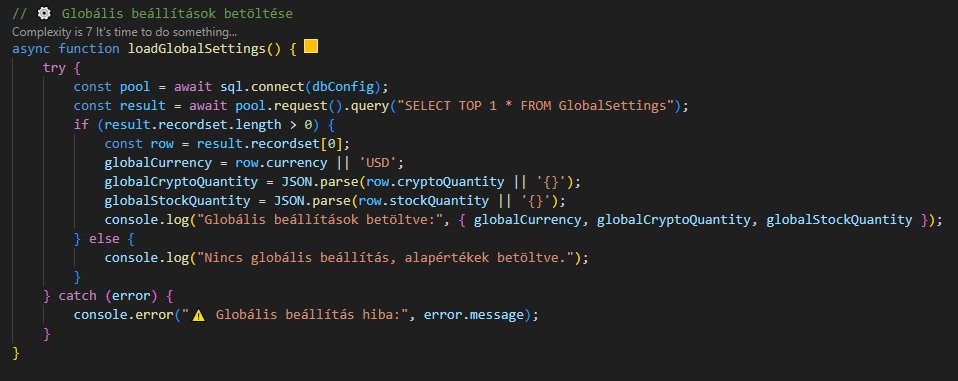
**9. Részvény árfolyamok lekérdezése**

* Lekéri az SPY, NVDA, MSFT árfolyamát a Twelve Data API segítségével.
* A válasz tartalmazza az árakat, amelyeket a frontend megjeleníthet.

**10. Kezdőlap**

A kezdőoldal automatikusan a bejelentkezési HTML-t tölti be.

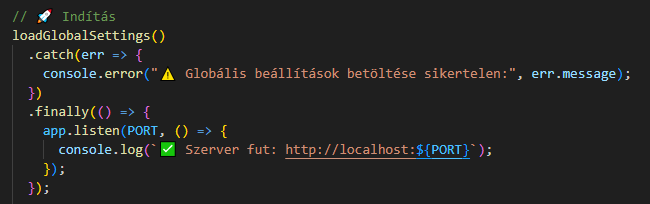
**11. Globális beállítások betöltése**



Ez az induláskor lefutó funkció lekérdezi a GlobalSettings táblát és beállítja:

* globalCurrency – az alapértelmezett pénznem
* globalCryptoQuantity – minden kriptóhoz tárolt érték (pl. mennyi áll rendelkezésre a játékban)
* globalStockQuantity – ugyanígy részvényekhez

**12. Szerver indítása**



A szerver csak a beállítások betöltése után indul el.

**13. Tranzakciók kezelése**

**URL**  
POST /api/transactions

**Hitelesítés**  
A végpont védett: a authenticateToken middleware-rel ellenőrzött, érvényes JWT token szükséges.

**Leírás**  
Kriptopénz vásárlási („Vétel”) vagy eladási („Eladás”) tranzakció rögzítése, majd a felhasználó USD-egyenlegének és kriptoállományának frissítése.

**Kérés**

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| type | String | Tranzakció típusa: "Vétel" vagy "Eladás" |
| amount | Number | Kriptopénz mennyisége (tizedesekkel, pl. 1.23456789) |
| currency | String | (Jelenleg nem használatos, a rendszer csak USD-alapú műveleteket támogat.) |
| crypto | String | Kriptoeszköz szimbóluma, pl. "BTC", "ETH". Vétel/Eladás esetén kötelező mező |
| price | Number | Egy kriptoegység USD-értéke a tranzakció időpontjában (tizedes pontosság) |

Pl:

{

"type": "Vétel",

"amount": 0.5,

"currency": "USD",

"crypto": "BTC",

"price": 60000.00

}

**Működési folyamat**

1. **Tranzakció rekord beszúrása**
   * Beszúrjuk a dbo.Tranzakciók táblába a felhasználó azonosítóját (req.user.id), a kriptoeszköz szimbólumát, a mennyiséget, az árat és a típust.
   * A beszúrás után lekérdezzük a generált TranzakcióID-t.
2. **Jelenlegi egyenleg és kriptoállomány lekérése**
   * A dbo.FelhasználóEgyenleg táblából lekérdezzük az Egyenleg (USD) és CryptoMennyiség (JSON-formátumú) mezőket.
   * Hibakezelés: ha nincs adat, kivételt dobunk.
3. **Egyenleg- és kriptofrissítés**
   * A CryptoMennyiség mezőt JSON-ként parse-oljuk ({} az üres alap).
   * **Vétel** esetén:
     + Kiszámoljuk a cost = amount \* price összeget.
     + Ha az USD-egyenleg kisebb, mint a cost → HTTP 400, { message: 'Nincs elég USD kriptó vásárláshoz.' }.
     + Ellenkező esetben csökkentjük az egyenleget és növeljük az adott kripto mennyiségét.
   * **Eladás** esetén:
     + Ellenőrizzük, hogy van‑e elég kriptó (cryptoObj[crypto] >= amount), különben HTTP 400, { message: 'Nincs elég kriptód eladásra.' }.
     + Ha van, kiszámoljuk a proceeds = amount \* price, növeljük az egyenleget és csökkentjük a kriptoállományt.
   * **Egyéb típus** esetén HTTP 400, { message: 'Ismeretlen tranzakciótípus.' }.
4. **Egyenleg mentése az adatbázisban**
   * Frissítjük a FelhasználóEgyenleg táblát az új Egyenleg és CryptoMennyiség (stringesített JSON) értékekkel.
5. **Válasz küldése**
   * Siker esetén HTTP 201, törzsében:

{

"success": true,

"transactionId": 123,

"balance": 7000.5,

"crypto": { "BTC": 0.5, "ETH": 0 }

}

1. **Hibakezelés**
   * Bármely, a fenti lépések során keletkező kivételt logolunk (console.error) és HTTP 500-at küldünk a kliensnek: { "message": "Hiba a tranzakció során." }
2. **Válaszok** 
   * HTTP státusz Törzs Magyarázat
   * 201 { success: true, transactionId, balance, crypto } Tranzakció sikeresen rögzítve, frissített egyenleg és portfólió
   * 400 { message: 'Nincs elég USD kriptó vásárláshoz.' }
   * { message: 'Nincs elég kriptód eladásra.' }
   * { message: 'Ismeretlen tranzakciótípus.' } Üzleti logikai hiba (nem elegendő forrás, vagy érvénytelen típus)
   * 500 { message: 'Hiba a tranzakció során.' } Váratlan szerverhiba

**Megjegyzések**

* A currency mező fogadása jelenleg csak helyőrző — az egyenleg mindig USD-ben történik.
* A crypto és price mezők kötelezőek mind vétel, mind eladás esetén.
* A pontos tizedespontosság miatt a mennyiségeket és árakat DECIMAL(18,8) típusú paraméterrel kezeljük az SQL-ben.
* A kliensoldalon a kapott transactionId felhasználható későbbi tranzakció-lekérdezésekhez vagy visszaigazolásokhoz.

2.5.2. Frontend interakciók és token kezelési logika

A **userdata.js** felelős a felhasználói űrlapok kezeléséért és a kliensoldali adatok kezeléséért:

* **Regisztrációs folyamat:**
  + Az űrlap adatainak összegyűjtése JavaScript eseménykezelő segítségével
  + A fetch API használata a POST kérés elküldésére, a backend regisztrációs végpontjára
  + Sikeres válasz esetén a felhasználót értesítő üzenetek megjelenítése és átirányítása a bejelentkezési oldalra
* **Bejelentkezési folyamat:**
  + Az email és jelszó adatok bekérése
  + A felhasználói token és adatok elmentése a localStorage-ben, hogy más oldalakon is elérhetőek legyenek
* **Kijelentkezési logika:**
  + A token és felhasználói adatok törlése és az oldal újratöltése a bejelentkezési felületre való navigációval
* **Profiladatok betöltése:**
  + Az oldal betöltésekor a mentett adatok automatikus megjelenítése, ezzel biztosítva a felhasználói élmény következetességét

**userdata.js**

**Áttekintés – Mi a userdata.js célja?**

A userdata.js egy **frontend JavaScript fájl**, amely a következő kulcsfontosságú felhasználói funkciókat valósítja meg:

1. **Regisztráció**
2. **Bejelentkezés**
3. **Kijelentkezés**
4. **Profilbetöltés**

A fájl **aszinkron módon** kommunikál a **Node.js + Express backenddel**, pontosabban a /api/register és /api/login végpontokkal. Ezen kívül kezeli a **felhasználói tokent és adatokat a localStorage-ben**, így más oldalak is hozzáférhetnek az éppen bejelentkezett felhasználóhoz.

A fájl működése lépésről lépésre

**1. Regisztráció kezelése**Ez a kód figyel egy registerForm nevű űrlapot. Ha megtalálja, akkor az alábbiakat teszi, amikor a felhasználó rákattint a „Regisztráció” gombra:

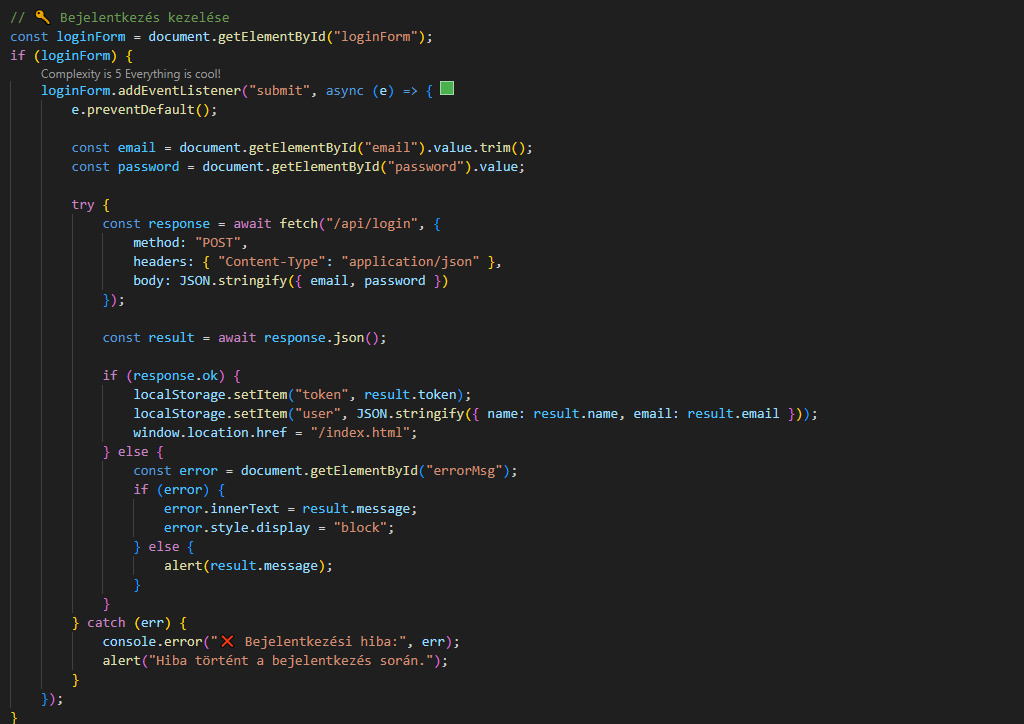
**Működés:**

1. **Űrlap adatainak összegyűjtése** – name, email, password
2. **Fetch POST** kérés a szerver felé:
   * Cél: api/register
   * Küldött adatok: { name, email, password }
3. **Sikeres válasz esetén:**
   * Felhasználó értesítése (alert)
   * Átnavigálás a bejelentkezési oldalra
4. **Hiba esetén:**
   * Hibaüzenet megjelenítése az errorMsg ID-jű elemben
   * Konzolos naplózás

**Biztonsági megjegyzés:**

A jelszó **nem kerül elmentésre localStorage-be**, csak a szerver kapja meg.

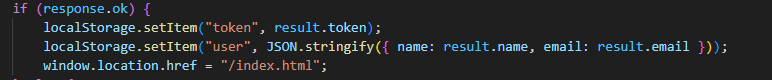
**2. Bejelentkezés kezelése**

Ez a blokk figyel egy loginForm-ot, és hasonló módon kezeli, mint a regisztrációt:

**Működés:**

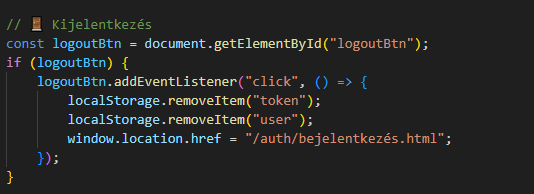
1. Beolvassa a beírt e-mail címet és jelszót.
2. Küld egy **POST** kérést a backend /api/login végpontjára.
3. A szerver válasza kétféle lehet:
   * **Sikeres**: visszakapja a token-t, name-et és email-t.
   * **Sikertelen**: hibaüzenet megjelenítése (errorMsg, vagy alert).

**Mit tárolunk?**



Ez azért fontos, mert más oldalak (pl. /btc.html) ebből olvassák ki a bejelentkezett felhasználót, és a JWT tokent – ezzel tudják azonosítani az adott usert a szerver felé küldött kérésekben.

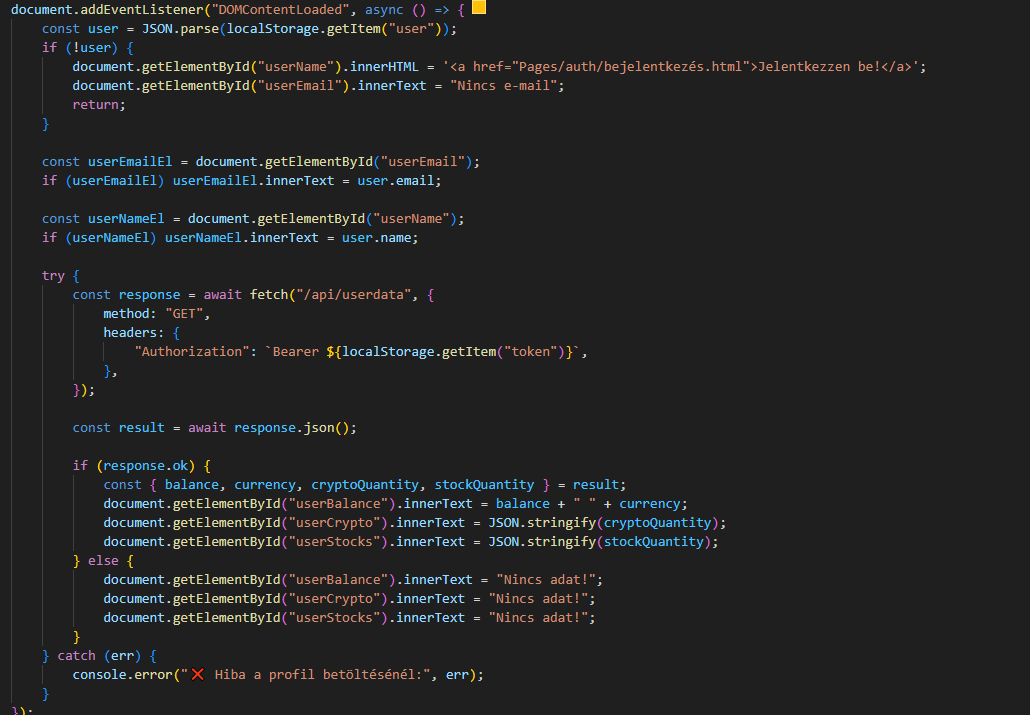
**3. Kijelentkezés logika**



**Működés:**

* Ellenőrzi, hogy van-e kijelentkezés gomb.
* Ha rákattintanak, akkor:
  + Törli a tokent és a felhasználói adatokat a localStorage-ből.
  + Átnavigál a bejelentkező oldalra.

Ez egy **gyors és hatékony módja** annak, hogy kijelentkeztessük a felhasználót anélkül, hogy bármit a szerverhez kéne küldenünk (mivel a token törlésével megszűnik a jogosultság).

**4. Profiloldal betöltése**Ez a rész azt csinálja:

* Megvárja, amíg betölt az oldal.
* Megnézi, van-e user mentve a localStorage-ben.
* Ha van, akkor beállítja a userEmail és userName mezőket az oldalon.

Ezt használhatod a profiloldaladon, vagy bárhol, ahol szeretnéd **kiírni a felhasználó adatait**.

**Miért jó ez a fájl?**

* Teljesen backend-orientált, nincs többé localStorage-alapú „házi adatbázis”.
* A JWT token alapján működik, így bővíthető role-alapú védelemmel is.
* Egyszerű, könnyen karbantartható, érthető.
* Token és felhasználóadat elérhető más fájlok számára is.
* HTML oldalakhoz van optimalizálva, nem kell hozzá React vagy más frontend framework.

**Javasolt bővítések / finomítások (opcionális)**

1. **Token lejárat figyelés**  
   Lehetőség van a JWT-ben tárolt lejárati idő (exp) ellenőrzésére, így automatikusan kijelentkeztethetjük a felhasználót.

2.6. Biztonsági megoldások és hitelesítési mechanizmusok

A Fundelio projekt kiemelkedő figyelmet fordít a biztonságra, különösen azért, mert pénzügyi szimulációs rendszerről van szó. Az alábbi intézkedések garantálják az adatok védelmét:

* **Jelszó Hash-elés:**  
  A bcryptjs segítségével minden felhasználó jelszavát egyszeri, erős hash-elési algoritmussal titkosítjuk, így az adatbázisban soha nem tárolódik sima szöveges formában a jelszó.
* **JWT Tokenek:**  
  Minden sikeres bejelentkezéskor a rendszer egy időkorlátos (exp) token-t generál, amely az API-khoz történő hozzáférést szabályozza.
* **HTTPS használata:**  
  A szerver konfigurálása során javasolt HTTPS protokoll használata, hogy a kliensek és a szerver közötti kommunikáció titkosított legyen.
* **Rate Limiting és IP blokkolás:**  
  Az olyan támadások ellen, mint az erőltetett jelszó kitalálás, rate limiting mechanizmusokat alkalmazunk, melyek korlátozzák a túl gyakori kéréseket.
* **Adatbázis Biztonság:**  
  Az MSSQL konfigurációjában a megfelelő jogosultságkezelést és auditálást alkalmazzuk, hogy az adatokhoz csak hitelesített felhasználók férhessenek hozzá.

2.7. Rendszerintegráció, API-k és külső szolgáltatások

A Fundelio több külső szolgáltatással integrálódik, amelyek révén a rendszer valódi időben képes az aktuális piac mozgását követni:

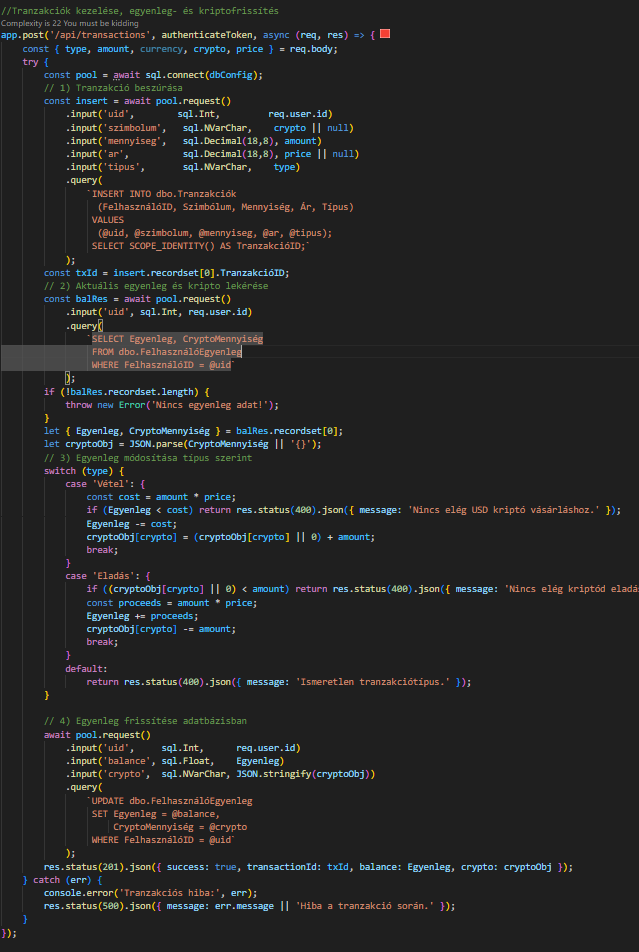
* **Binance WebSocket API:**  
  A Binance API-val történő WebSocket kapcsolat lehetővé teszi, hogy a rendszer a kriptovaluták (BTC, ETH, DOGE stb.) élő árfolyam adatait valós időben frissítse. A szerver egy dedikált szervizt futtat, amely minden egyes kereskedési eseményt (trade) fogad, és az aktuális árat a memóriában, az ún. prices objektumban tárolja, majd egy dedikált API végpont szolgáltatja az adatokat a frontendre.
* **Twelve Data API:**  
  A részvényárfolyamok eléréséhez az axios modul segítségével hívjuk meg a Twelve Data API-t, és a kapott adatok alapján építjük fel az élő árfolyam adatokat, melyeket szintén a frontend lekérdezhet.
* **Nodemailer:**  
  Habár jelenleg nem aktív a rendszerben, a nodemailer modul későbbi integrációra készül, hogy e-mail alapú értesítéseket, jelszó-emlékeztetőt vagy egyéb kommunikációt támogasson.

2.8. Hibakezelés és naplózás

A rendszer megfelelő hibakezelési mechanizmusokkal és naplózási lehetőségekkel rendelkezik:

* **Middleware alapú hibakezelés:**  
  Az Express hibakezelő middleware-je biztosítja, hogy minden keletkező kivétel esetén részletes hibaüzenet kerüljön visszaküldésre, ugyanakkor a felhasználó számára barátságos formában legyenek az értesítések megjelenítve.
* **Naplózás:**  
  A server.js-ben és a kliensoldali kódban is alkalmazott konzol logok segítik a fejlesztőket az esetleges hibák és problémák gyors azonosításában. A naplózási rendszer kiterjeszthető külső log-kezelő eszközök integrációjával is (pl. Winston).

2.9. Tranzakciók kezelése



**URL**  
POST /api/transactions

**Hitelesítés**  
authenticateToken middleware szükséges: a kérés fejléceiben érvényes JWT (Authorization: Bearer <token>).

**Leírás**  
Kriptopénz-vásárlási vagy eladási tranzakció rögzítése, majd a felhasználó USD-egyenlegének és kriptoállományának frissítése.

**Kérés**

| **Mező** | **Típus** | **Leírás** |
| --- | --- | --- |
| type | String | Tranzakció típusa: "Vétel" vagy "Eladás" |
| amount | Number | Kriptómennyiség (tizedespontossággal, pl. 0.12345678) |
| currency | String | (Jelenleg csak "USD" támogatott, de nem használjuk közvetlenül a logikában.) |
| crypto | String | Kriptoeszköz szimbóluma (pl. "BTC", "ETH"); kötelező mező mindkét típusnál |
| price | Number | Egy egység kripto USD-értéke a tranzakció pillanatában (tizedespontossággal) |

Pl: {

"type": "Vétel",

"amount": 0.5,

"currency": "USD",

"crypto": "BTC",

"price": 60000.00

}

**Folyamat**

1. **Tranzakció beszúrása**  
   Beszúrásra kerül a dbo.Tranzakciók táblába a felhasználó azonosítója, kripto szimbólum, mennyiség, ár és típus. A lekérdezés visszaadja az újonnan generált TranzakcióID-t.
2. **Aktuális egyenleg és kriptoállomány lekérése**  
   A dbo.FelhasználóEgyenleg táblából kiolvasásra kerül az USD-egyenleg (Egyenleg) és a JSON-formátumú CryptoMennyiség. Ha nincs rekord, kivétel dobódik.
3. **Üzleti logika – egyenleg és kriptofrissítés**
   * A CryptoMennyiség JSON-parsolása JavaScript-objektummá (alapértelmezetten {}).
   * **Vétel**:
     + Költség: amount \* price
     + Ha az egyenleg kisebb, mint a költség → HTTP 400, { message: 'Nincs elég USD kriptó vásárláshoz.' }
     + Egyenleg csökkentése, kripto-mennyiség növelése.
   * **Eladás**:
     + Ha nincs elég kripto (cryptoObj[crypto] < amount) → HTTP 400, { message: 'Nincs elég kriptód eladásra.' }
     + Bevétel: amount \* price
     + Egyenleg növelése, kripto-mennyiség csökkentése.
   * **Egyéb típus** → HTTP 400, { message: 'Ismeretlen tranzakciótípus.' }
4. **Egyenleg mentése**  
   Az új Egyenleg és CryptoMennyiség (JSON-string) értékekkel frissítjük a FelhasználóEgyenleg táblát.
5. **Válasz küldése**  
   Siker esetén HTTP 201, törzsében:

{

"success": true,

"transactionId": 123,

"balance": 9500.50,

"crypto": { "BTC": 0.5, "ETH": 1.2 }

}

1. **Hibakezelés**  
   Bármely váratlan hiba esetén logoljuk (console.error) és HTTP 500-at küldünk:

{ "message": "Hiba a tranzakció során." }

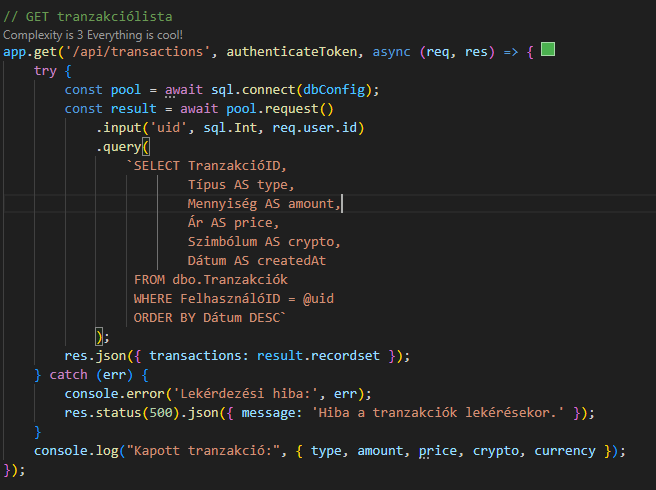
**Válaszok**

| **HTTP státusz** | **Törzs** | **Jelentés** |
| --- | --- | --- |
| **201** | { success: true, transactionId, balance, crypto } | Sikeres tranzakció, frissített egyenleg és portfólió |
| **400** | { message: 'Nincs elég USD kriptó vásárláshoz.' } { message: 'Nincs elég kriptód eladásra.' } { message: 'Ismeretlen tranzakciótípus.' } | Üzleti logikai hiba (nem elegendő forrás vagy érvénytelen típus) |
| **500** | { message: 'Hiba a tranzakció során.' } | Váratlan szerverhiba |

**Megjegyzések**

* A currency mező bevitele csak a jövőbeli bővítéshez van bent; jelenleg minden művelet USD-ben történik.
* A mennyiségeket és árakat SQL-oldalon DECIMAL(18,8) típusra mappingoljuk a pontosság érdekében.
* A kliensoldalon a visszakapott transactionId használható későbbi lekérdezésekhez vagy visszaigazolásokhoz.

2.10 Tranzakciólista lekérdezése



**URL**  
GET /api/transactions

**Hitelesítés**  
authenticateToken middleware: az Authorization: Bearer <token> fejlécben kapott JWT-ből req.user.id-t használjuk.

**Kérés**

* **Body**: nincs
* **Header**:
  + Authorization: Bearer <token>

**Folyamat**

1. **Adatbázis-kapcsolat létrehozása**  
   A dbConfig-ból mssql pool használatával csatlakozunk.
2. **Tranzakciók lekérdezése**

SELECT

TranzakcióID,

Típus AS type,

Mennyiség AS amount,

Ár AS price,

Szimbólum AS crypto,

Dátum AS createdAt

FROM dbo.Tranzakciók

WHERE FelhasználóID = @uid

ORDER BY Dátum DESC;

(– @uid a tokenből vett req.user.id.)

1. **Válasz összeállítása**  
   A result.recordset-et beágyazzuk: { "transactions": [ /\* … \*/ ] }
2. **Hibakezelés**  
   Sikertelen lekérdezés vagy kapcsolat → console.error, HTTP 500: { "message": "Hiba a tranzakciók lekérésekor." }

**Válaszok**

| **Kód** | **Tartalom** |
| --- | --- |
| 200 | { "transactions": [ { transactionId, type, amount, price, crypto, createdAt }, … ] } |
| 500 | { "message": "Hiba a tranzakciók lekérésekor." } |

**Frontend integráció (tervezett)**

* **Jelenleg nincs UI**: A listázó oldal még nem hívja ezt az endpointot, csak a backend-kódban került definiálásra.
* **Integrálási lépések**:
  1. A tranzakciólistáért a kliensoldalon fetch('/api/transactions', { headers: { Authorization: 'Bearer ' + token } }) hívás.
  2. A visszaérkező transactions tömböt tábla vagy kártyák formájában renderelni (pl. React-ben <Table> komponens).
  3. A createdAt mezőt igény szerint formázni (pl. YYYY.MM.DD HH:mm).
* **Bővíthetőség**:
  1. Szűrés dátum intervallumra, típusra (Vétel/Eladás), vagy kriptószimbólumra egyszerű query-paraméterekkel (?type=Vétel stb.).
  2. Oldaltördelés (pagination) bevezetése a backendben (OFFSET … FETCH) és frontendben.
* Ezekkel a lépésekkel néhány perc alatt beköthető a tranzakciólista a felhasználói felületbe.

**2.11 Crypto, illetve részvény oldalak backend működése**

**Cél**

* Felhasználói egyenleg és eszköz‑mennyiség lekérése
* Valós idejű árfolyam‑megjelenítés (Chart.js)
* Vásárlás/eladás indítása a háttér‑API-kon keresztül
* Könnyen újrahasználható bármely kripto- vagy részvényoldalon

**Scope**

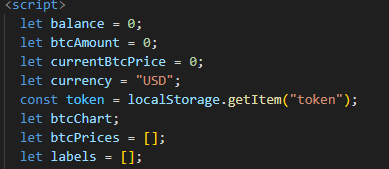
* **Kriptoeszközök:** pl. btcusdt, ethusdt stb.
* **Részvények:** pl. AAPL, GOOG – azonos logika, csak endpoint-URL és DOM‑elemek szimbolikája változik.

**Endpoint-ok és adatmodellek**

| **Funkció** | **HTTP-módszer** | **URL** | **Visszatérő JSON mezők** |
| --- | --- | --- | --- |
| Felhasználói adatok | GET | /api/userdata | { balance, currency, cryptoQuantity, stockQuantity } |
| Élő árfolyam | GET | /api/live/:symbol | { price } |
| Tranzakció létrehozása | POST | /api/transactions | { success, balance, crypto, stocks } |

**Megjegyzés:** :symbol lehet tetszőleges instrumentum‑azonosító (pl. btcusdt, AAPL).

**Főbb változók és DOM-kapcsolat**

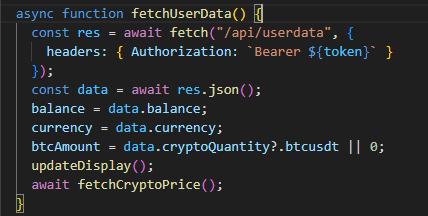


* **DOM-elemek:**
  + **#balance – egyenleg kiírás**
  + **#[symbol]Amount – eszköz‑mennyiség (pl. #btcAmount)**
  + **#[symbol]Price – árfolyam (pl. #btcPrice)**
  + **#[symbol]AmountInput – bevitt mennyiség mező**
  + **#totalCost – tranzakció költsége**

**Ha részvényoldalon használjuk, cseréljük a DOM‑ID-ket (#stockPrice, #AAPLAmount, stb.).**

**Működési lépések**

1. **Főoldali inicializáció (window.onload)**
   * **fetchUserData() → beállítja balance, currency, assetAmount → updateDisplay()**
   * **createChart() → üres line chart létrehozása**
   * **updateChart() + setInterval(updateChart, 5000) → 5 másodpercenként árajánlat‑ és grafikonfrissítés**
2. **Felhasználói adatok lekérése**



1. **Árfolyam lekérése és költség számítása**
   * fetchPrice() meghívja /api/live/:symbol → currentPrice
   * updateDisplay() frissíti az árfolyam‑szöveget
   * updateCost() kiszámolja: currentPrice \* bevitt mennyiség
2. **Vásárlás/eladás**
   * **buyAsset()** és **sellAsset():**



1. **Grafikon Frissítése**
   * Új pont (currentPrice, timestamp) hozzáadása a chart-hoz
   * Ha több mint N pont, a legrégebbi levétele (sliding window)
2. **Eltérések a stock oldalakon**
3. **assetSymbol és mennyiségi mező**
   * Kripto: const assetSymbol = "btcusdt" és data.cryptoQuantity?.[symbol]
   * Részvény: const assetSymbol = "MSFT" és data.stockQuantity?.[assetSymbol]
4. **Árfolyam‑lekérdezés**
   * Kripto: fetch("/api/live/:symbol") egy objektumot ({ price }) ad vissza.
   * Részvény: fetch("/api/stocks") egy tömböt ad ([{ symbol, price }, …]), amit data.find(item => item.symbol === assetSymbol)-pel szűrünk ki.
5. **Tranzakciós hívás testének (body) mezői**
   * Mindkét esetben POST /api/transactions, de a backend a crypto mezőt használja általános azonosítóként – itt is ezt töltjük assetSymbol-lel, illetve a válaszban data.stock?.[assetSymbol]-ből olvassuk vissza az új állományt (feltéve, hogy a backend támogatja a stocks kulcsot).
6. **Visszaállítás („reset”)**
   * Kripto: API-hívással (deposit típusú tranzakció) állítjuk vissza a kezdőösszeget.
   * Részvény: egyszerűen lokálisan (balance = 10000; assetAmount = 0;), nincs backend‑hívás.
7. **Chart konfiguráció**
   * Label: "BTC árfolyam (USD)" helyett `MSFT árfolyam (${currency})`
   * További: kripto esetén prices és labels tömbök globálisan voltak, itt hiányzott a tömb inicializálása (ügyelj arra, hogy let prices = [], let labels = [] is definiálva legyen).
8. **Frissítési intervallum**
   * Kripto: 5 másodpercenként (5000 ms)
   * Részvény: 100 000 ms (100 másodperc) — tetszőlegesen módosítható.

**Összefoglalva**, a stock‑oldalra átültetés során:

* symbol és a hozzá tartozó mennyiségi mező neve (stockQuantity vs cryptoQuantity),
* az árfolyam‑endpoint és annak formátuma (/api/stocks tömb vs /api/live objektum),
* a reset logika (lokális vs API),
* valamint a chart címkézése és frissítési gyakorisága tér el.

3. Tesztelési Dokumentáció

A tesztelési dokumentáció célja, hogy bemutassa, a Fundelio projekt hogyan reagál különféle körülmények között, és milyen módszerekkel ellenőriztük a rendszer megbízhatóságát.

3.1. A tesztelési stratégia és célok

A tesztelési megközelítésünk ötvözi a manuális és az automatizált tesztelést, hogy minden esetben biztosítsuk a rendszer stabil és hibamentes működését. Célunk, hogy:

* **Funkcionális stabilitás:**  
  Minden API végpont, felhasználói funkció és élő adatfolyam kezelés hibátlanul működjön különböző környezetekben.
* **Terheléses tesztelés:**  
  A rendszer megbízhatóan működjön nagy számú egyidejű felhasználó esetén, és megfelelően skálázódjon.
* **Biztonsági tesztek:**  
  Ellenőrizzük a token alapú hitelesítés és a jelszó hash-elés hatékonyságát, valamint a rate limiting megoldások működését.

3.2. Manuális tesztek és funkcionális ellenőrzések

**Felhasználói műveletek tesztelése:**

* **Regisztráció:**  
  Teszteljük, hogy az űrlap helyes adatokat vár, és hibás email vagy jelszó esetén megfelelő hibaüzenetet kap a felhasználó.
* **Bejelentkezés:**  
  Ellenőrizzük, hogy a bejelentkezési adatok helyes kombinációja esetén a rendszer megfelelő JWT token-t generál, és a hibás adatok esetén a felhasználót figyelmezteti.
* **Token validáció:**  
  A védett útvonalak lekérésénél teszteljük, hogy a token nélkül vagy érvénytelen tokennel érkező kérések elutasításra kerülnek.
* **Statikus fájlok kiszolgálása:**  
  Ellenőrizzük, hogy a HTML, CSS és JavaScript fájlok megfelelően töltődnek be minden támogatott böngészőben.

**Adatbázis műveletek ellenőrzése:**

* **Új felhasználó regisztrációja:**  
  Minden új regisztrációs kísérlet után ellenőrizzük, hogy az adatbázis megfelelő módon frissült-e, és az email cím egyediségét betartottuk.
* **Adatfrissítések és tranzakciók:**  
  Teszteljük, hogy a Tranzakciók tábla eseményei helyesen frissülnek az élő adatfolyam adatai alapján, és hogy a Globális beállítások megfelelően töltődnek be.

3.3. Teljesítménytesztelés, stresszteszt és terheléses vizsgálatok

Az élő piaci adatfolyamok és a valós idejű API hívások miatt kiemelten fontos, hogy a rendszer nagy terhelés alatt is megfelelően működjön. A tesztelési eljárások során:

* **Terhelési tesztek:**  
  Többszámos egyidejű bejelentkezést, regisztrációt és API hívást szimuláltunk, hogy felmérjük a szerver válaszidejét.
* **Stressztesztek:**  
  A rendszer határait feltérképeztük, például mikor kezd csökkenni a válaszidő, vagy milyen hibaüzenetek jelennek meg extrém körülmények között.
* **Memóriahasználat és erőforrás-kezelés:**  
  A szerver erőforrás-fogyasztását mértük, hogy a memória szivárgások és a túlterheltség esetén időben beavatkozhassunk.

3.4. Tesztelési eredmények és hibakeresési folyamat

A tesztelési fázis során rögzített eredményeket részletesen dokumentáltuk, így az esetleges hibák gyors elhárítása lehetséges:

* **Hibák kategorizálása:**  
  Azonosított hibákat súlyosság, előfordulási gyakoriság és hatás alapján osztályoztuk.
* **Hibakeresési logok:**  
  A fejlesztői naplókban szereplő konzol logok és a szerver oldali hibajelentések alapján határoztuk meg a problémák okát.
* **Javítási ciklusok:**  
  A tesztelési eredmények alapján rendszeresen frissítettük a kódot, és az új verziókat elmentettük, hogy az előzőleg azonosított hibák ne térjenek vissza.

3.5. Tesztelésben észlelt hibák

A tesztelés alatt számos hibába ütközött programunk. Itt fel fogom sorolni azokat, amelyikeket a legnehezebb volt kijavítani.

* **Routing hibák**

Tesztelésünk során a publikálás végett számos routing hibát észleltünk, mivel a publikálási oldal máshogy érzékeli azokat a routingokat, amiket a HTML ismer.

* **Light-Dark CSS**

A Light-Dark css váltás közötti hibák megnehezítették a tesztelési folyamatot, mivel mindkettő oldalt le kellett tesztelni, hogy ugyanúgy nézzenek ki, illetve ne vesszenek el funkciók egy-egy css hiba által.

* **Adatbázis kapcsolat**

Az adatbázis kapcsolat során rettentő sok hibába akadtunk, mivel nem mindig akart kapcsolódni a kiszolgáló az Azure által kiszolgált adatbázisunkhoz.

4. Fejlesztési Lehetőségek és Jövőbeni Bővítések

A Fundelio projekt folyamatos fejlesztésre és bővítésre ad lehetőséget, amelyet az alábbiakban ismertetünk részletesen.

4.1. Új funkciók és integrációk

* **Több kereskedési eszköz integrálása:**  
  A jelenlegi rendszerbe hatféle kriptovaluta és háromféle részvény van integrálva. A jövőben további eszközök (pl. árupiac, devizakereskedés) hozzáadása lehetőséget ad a felhasználók számára, hogy szélesebb körű tapasztalatokat szerezzenek.
* **Grafikonok és statisztikák megjelenítése:**  
  Érdemes fejleszteni egy részletes analitikai modult, amely diagramok és grafikonok segítségével vizualizálja a kereskedési adatokat, így segítve a felhasználói döntéshozatalt.
* **Játékosított elemek beépítése:**  
  A szintlépés, ranglista vagy kihívások integrálása növelheti a felhasználói elkötelezettséget, miközben interaktív tanulási élményt kínál.
* **Mobil alkalmazás fejlesztése:**  
  A webes felület mellett egy natív vagy cross-platform mobil alkalmazás fejlesztése lehetővé tenné, hogy a felhasználók útközben is hozzáférjenek a rendszer funkcióihoz.

4.2. Rendszer skálázhatósága és optimalizációs lehetőségek

* **Cache megoldások:**  
  A kriptovaluta és részvény árfolyam adatok többszöri lekérésének optimalizálása cache használatával, például Redis vagy egyéb in-memory adatbázissal, amely csökkenti az API hívások számát.
* **Load Balancing:**  
  Több szerver használata és megfelelő terheléselosztó beépítése a rendszer terhelhetőségének növelése érdekében.
* **Adatbázis optimalizálás:**  
  Az MSSQL adatbázis rendszerben megfelelő indexek, eljárások és tárolt eljárások alkalmazása a lekérdezések sebességének növelése érdekében.

4.3. UX/UI fejlesztések és felhasználói visszajelzések beépítése

* **Modern, reszponzív design:**  
  A felhasználói felület továbbfejlesztése érdekében érdemes modern frontend keretrendszereket (pl. React, Vue.js) részben integrálni, hogy a design még letisztultabb és reszponzívabb legyen.
* **Interaktív visszajelzések:**  
  Hibakezelő üzenetek, loading spinner-ek, és más vizuális elemek bevezetése, amelyek segítik a felhasználót a műveletek során.
* **Felhasználói visszajelzés gyűjtése:**  
  Online kérdőívek, elégedettségi felmérések és közösségi fórumok integrálása, amelyek segítségével a jövőbeni fejlesztéseket a felhasználók igényei alapján tudjuk alakítani.

4.4. Biztonsági fejlesztések és rendszeres auditok

* **JWT token lejárati idő figyelése:**  
  Az aktuális token lejárati idő implementálása és automatikus kijelentkeztetés, ha a token már nem érvényes.
* **Többrétegű hitelesítés:**  
  Olyan extra biztonsági rétegek bevezetése, mint a kétfaktoros hitelesítés (2FA), hogy a felhasználói fiókok még jobban védettek legyenek.
* **Rendszeres biztonsági auditok:**  
  Független szakértők által végzett auditok, amelyek feltárják az esetleges sebezhetőségeket, és javaslatokat tesznek azok kiküszöbölésére.
* **Rate Limiting és IP blokkolás:**  
  Az extrém esetekre, például DoS (Distributed Denial-of-Service) támadások esetére kidolgozott védelmi mechanizmusok alkalmazása.

**5.** Felhasználói Dokumentáció

**1. Kezdőlap**

- Főmenü: „Főoldal”, „Piac”, „Portfólió”, „Profil”.

- Jobb oldalon: „Bejelentkezés” és „Regisztráció” gombok.

-Bal oldalon: Újdonságok sidebar

Fő tartalom:

- Rövid bemutató szöveg: „Üdvözlünk a Fundelio-n!”

Interakciók:

- Cikkekre kattintva → Átirányít az adott cikkre

- Cryptok lenyíló menüpont, bármelyikre kattintva → Átirányít az adott crypto oldalára

- Részvények lenyíló menüpont, bármelyikre kattintva → Átirányít az adott részvény oldalára

- Profilra kattintva → Átirányít a profil oldalra

- Főoldalra kattintva → Átirányít a főoldalra

- Hold/Nap ikonra kattintva → Változtatjuk az oldal tematikáját, sötétre, vagy világosra.

- Főmenü pontok: ha nincs bejelentkezve, felugró ablak: „Kérjük, jelentkezz be!”

**2. Regisztráció oldal**

Regisztráció menüpont

Itt található 3 mező, a Teljes név, az Email cím, és a Jelszó.

-Adjuk meg a teljes nevünket

-Adjunk meg egy valós email címet a további hibaelhárítás érdekében. Tartalmazzon @-ot, illetve gmail.com formátumú legyen. pl:tesztelő@gmail.com

-Adjunk meg egy jelszavat.

Rákattinthatunk a regisztráció gombra, ami ha sikeres, akkor egy felugró ablak jelenik meg, hogy sikeres regisztráció.

-Található egy bejelentkezés gomb, ami a bejelentkezés oldalra navigál minket.

**3. Bejelentkezés oldal**

Bejelentkezés menüpont:

- Mindenképp a regisztrált e-mail címet kell megadnunk, különben nem tudunk belépni az oldalra. Ha ez meg van, akkor

- Beírjuk a jelszót. Itt is a regisztrált jelszót kell megadnunk.

-Ha beírtunk mindent jól, rányomhatunk a bejelentkezés gombra, és ha minden jó, akkor a főoldalra irányít minket.

-Található egy “regisztráció” gomb is, erre kattintva a regisztrációs oldalra léptet minket.

**4. Kripto oldalak**

- 6 fajta kriptovaluta érhető el: Bitcoin, Dogecoin, Ethereum, Solana, Trumpcoin, és XRP.

- Felül található a navbar:

- Cryptok lenyíló menüpont, bármelyikre kattintva → Átirányít az adott crypto oldalára

- Részvények lenyíló menüpont, bármelyikre kattintva → Átirányít az adott részvény oldalára

- Profilra kattintva → Átirányít a profil oldalra

- Főoldalra kattintva → Átirányít a főoldalra

- Hold/Nap ikonra kattintva → Változtatjuk az oldal tematikáját, sötétre, vagy világosra.

Oldal közepén:

Kereskedés menüpont, az adott oldal kriptovalutájának tudjuk nézni az árfolyamát USD árban.

Található egy rendelkezésre álló összeg felirat, ez mutatja hogy mennyi a rendelkezésre álló összegünk.

Található egy textbox, amibe tudunk összeget írni, hogy mennyit szeretnénk vásárolni/eladni az adott részvényből.

Vásárlás gomb: Az összeget beírva megvásárlunk x mennyiségű részvényt.

Eladás gomb: Az összeget beírva eladunk x mennyiségű részvényt.

**5. Részvény piac oldal**

- 3 fajta részvény érhető el: SPY, MSDT, és NVDA.

- Felül található a navbar:

- Cryptok lenyíló menüpont, bármelyikre kattintva → Átirányít az adott crypto oldalára

- Részvények lenyíló menüpont, bármelyikre kattintva → Átirányít az adott részvény oldalára

- Profilra kattintva → Átirányít a profil oldalra

- Főoldalra kattintva → Átirányít a főoldalra

- Hold/Nap ikonra kattintva → Változtatjuk az oldal tematikáját, sötétre, vagy világosra.

Oldal közepén:

Kereskedés menüpont, az adott oldal részvényének tudjuk nézni az árfolyamát USD árban.

Található egy rendelkezésre álló összeg felirat, ez mutatja hogy mennyi a rendelkezésre álló összegünk.

Található egy textbox, amibe tudunk összeget írni, hogy mennyit szeretnénk vásárolni/eladni az adott kriptovalutából.

Vásárlás gomb: Az összeget beírva megvásárlunk x mennyiségű kriptovalutát.

Eladás gomb: Az összeget beírva eladunk x mennyiségű kriptovalutát.

**6.Profil oldal**

- Felül található a navbar:

- Cryptok lenyíló menüpont, bármelyikre kattintva → Átirányít az adott crypto oldalára

- Részvények lenyíló menüpont, bármelyikre kattintva → Átirányít az adott részvény oldalára

- Profilra kattintva → Átirányít a profil oldalra

- Főoldalra kattintva → Átirányít a főoldalra

- Hold/Nap ikonra kattintva → Változtatjuk az oldal tematikáját, sötétre, vagy világosra.

Oldal közepén egy profil menü:

-Kiírja az emailünket amivel be vagyunk jelentkezve

-Név: Kiírja a nevünket

-Egyenleg: Mutatja a rendelkezésünkre álló egyenleget

-Kijelentkezés gomb: Kijelentkeztet minket, és bejelentkezés oldalra vezet

6. Összefoglalás, Értékelés és Köszönetnyilvánítás

**6.1.** Összefoglalás

A Fundelio projekt egy átfogó, modern webes szimulációs platform, amely a valós idejű adatfolyamok és a biztonságos felhasználói élmény ötvözésével kínál megoldást a pénzügyi piacok oktatására. A projekt fejlesztése során figyelembe vettük a moduláris architektúrát, a skálázhatóságot és a biztonságot, így egy olyan rendszert hoztunk létre, amely nem csak a jelenlegi felhasználói igényeket elégíti ki, de a jövőbeni bővítésekre is kiváló alapot nyújt.

6.2. Értékelés

A rendszer működését a következő szempontok alapján értékeltük:

* **Technikai Megvalósítás:**  
  Az egyedi algoritmusok, a backend és frontend integrációja, valamint a valós idejű adatfolyamok kezelése biztosítja a rendszer hatékony működését.
* **Oktatási Érték:**  
  A Fundelio nem csupán egy kereskedési szimulátor, hanem egy oktatási eszköz, amely segít a felhasználóknak megérteni a pénzügyi piacok dinamikáját, miközben kockázat nélkül gyakorolhatnak.
* **Biztonság:**  
  A modern titkosítási módszerek, a token alapú hitelesítés és az adatbázis biztonsági megoldások garantálják a felhasználói adatok védelmét.

6.3. Köszönetnyilvánítás

Köszönet illeti mindazokat, akik hozzájárultak a Fundelio projekt sikeréhez:

* **Témavezetők:** Szalainé Török Edit és Méhes József, akik szakmai iránymutatásukkal segítették a projekt kialakítását.
* **Fejlesztői csapat:** Karita Árpád Ferenc, Borbély Alex, akik a projekt technikai megvalósításában végigszükséges szakértelmet adták.
* **Támogatók és mentorok:** Mindazok, akik a fejlesztési folyamat során ötleteikkel, javaslataikkal és támogatásukkal hozzájárultak a rendszer folyamatos finomításához.

**7.** Irodalomjegyzék

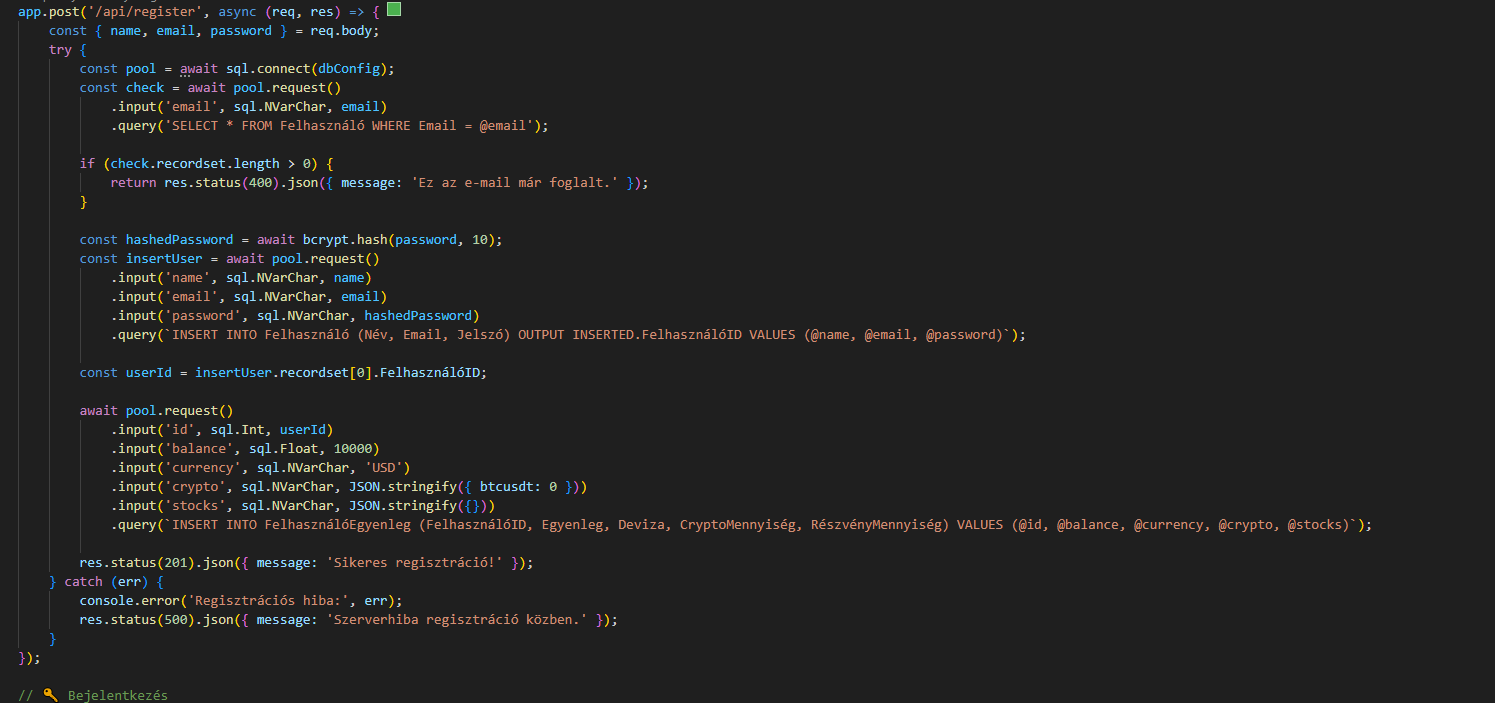
Az alábbi szakmai források és dokumentációk segítettek a Fundelio projekt megvalósításában:

* Űrlap-elemek megvalósítása HTML-ben  
  <http://infojegyzet.hu/webszerkesztes/html/urlapok/>
* PHP alapok – szerveroldali programozás  
  <http://infojegyzet.hu/webszerkesztes/php/alapok/>
* Dinamikus képgaléria  
  <http://infojegyzet.hu/webszerkesztes/php/kepgaleria/>
* Záródolgozat témák, szempontok, ötletek  
  <http://infojegyzet.hu/webszerkesztes/zarodolgozat/>
* Binance API dokumentáció  
  Az aktuális dokumentáció elérhető a Binance hivatalos oldalán.
* Twelve Data API dokumentáció  
  Az API használati útmutató részletesen ismerteti a részvényárfolyam adatok lekérését.

**8.** Függelék

A függelékben további kódrészleteket, diagramokat és konfigurációs leírásokat találhatsz, amelyek a projekt részletesebb megértését segítik elő.

**8.1. Kódrészletek és Példák**

**Példa: Regisztrációs Végpont Implementációja**

8.2. Diagramok és Adatbázis Sémák

**Adatbázis séma ábra:**  


9. Fundelio – Főoldal HTML Oldal Dokumentáció

Ez a HTML oldal a Fundelio nevű befektetési szimulátor főoldala, amely betekintést nyújt a platform működésébe, bemutatja annak célját, funkcióit, valamint interaktív elemeket kínál a felhasználók számára. Itt beszélünk az oldal felépítéséről, és technológiáiról.

9.1 Főoldal felépítése

<!DOCTYPE html>: HTML5 szabvány.  
- <html lang="hu">: A program nyelve.  
- <head>: Metaadatok, stíluslapok (Bootstrap, CSS), JavaScript könyvtárak (Chart.js, authorization.js). A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.  
**9.2 Használt Technológiák**

- Bootstrap 5.3 a reszponzív dizájnhoz.  
- Chart.js a kriptovaluta árfolyamok grafikonos megjelenítéséhez.  
- JavaScript az oldalon belüli funkciókhoz és API-kommunikációhoz.

- Téma váltó gomb is elérhető a világos/sötét mód között.  
- Külső stíluslap: -style-dark.css – ez az oldal sötét témáját biztosítja.

-style-lightcss - ez az oldal világos témáját biztosítja.

9.3.1 Navigációs Sáv (Navbar)

A navigációs sáv az oldal tetején helyezkedik el, három fő részre oszlik:  
- Bal oldal: Navigációs menüpontok  
 - Főoldal  
 - Profil  
 - Cryptók (lenyíló menü): Bitcoin, Dogecoin, Ethereum, Solana, Trumpcoin, XRP  
 - Részvények (lenyíló menü): Spy, Nvidia, ...  
- Középső rész: Oldalcím és szlogen („Üdvözöl a Fundelio!”, „Játssz, fektess be, és irányítsd a közösséget.”)  
- Jobb oldal: Téma kapcsoló gomb

- .navbar, .container-navbar: Felső navigációs sáv. Tartalmaz menüpontokat (Főoldal, Profil, Cryptók, Részvények), üdvözlő szöveget, téma váltó gombot és egy bejelentkezés gombot.

9.3.2 Oldalsáv

Az oldalsó „sidebar-news” szekció aktuális fejlesztésekről és funkcióbővítésekről ad tájékoztatást.

- .sidebar-news: Fix pozíciójú oldalsáv a hírekmegjelenítésére.

A képen szöveg, elektronika, képernyőkép, képernyő látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

9.4. Tartalmi Rész

**A. Cikk Szekció**

Láthatóak cikkek, amik friss híreket adnak a kriptovaluták/ részvények árainak változásárol, illetve a fontosabb történésekről. A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

- .szekciók: A cikkek fő tárolója, ami lehetővé teszi, hogy több cikk esetén is elrendezve legyenek.  
- .cikkek: Egyedi cikk”dobozok” képpel, címmel, bevezetővel és dátummal.  
- .img-article: cikkhez illő kép minden cikk elején.

Innen tovább tudunk lépni a cikkekre a bövebben gombra kattintva.

9.5. Dinamikus Funkciók

Az oldal több JavaScript funkciót tartalmaz, ezek célja a felhasználói élmény növelése és az adatok dinamikus betöltése. Bár ebben a részben nem térünk ki a konkrét kódokra, az alábbi fő funkciók találhatók meg:  
- Hitelesítés ellenőrzése: A felhasználó JWT tokenje alapján történik, és ennek segítségével lekérhető az aktuális egyenleg.  
- Kripto árfolyamok lekérése: A Binance API-n keresztül történik, és az árfolyamok grafikonként jelennek meg az oldalon.

9.6. Stílus és Téma Kezelés

A light-dark.js script segítségével lehetőség van a világos és sötét mód közötti váltásra. A téma váltás a felhasználói felületen egy gombbal történik, amely módosítja az oldalhoz tartozó CSS stíluslapot, a style-light.css-t, vagy a style-dark.css-t.

9.7. Felhasználói Élmény és Cél

Ez az oldal:  
- Tájékoztatja a felhasználót az oldal lényegéről.  
- Átlátható navigációval segít eljutni a különböző aloldalakra (kriptó/részvény oldalak, profil, stb.).  
- Biztosítja az interaktív funkciókat (árfolyamgrafikon, hírek, téma módosítás, animációk).  
- Az oldal célja az edukáció és szimuláció, nem pedig valódi pénzügyi tevékenység.

10. CSS Stíluslapok

**A. Sötét Téma (style-dark.css)**

- body, .bodyIndex: Sötét (#121212) háttér, fehér szöveg, Arial, Sans-serif betűtípus.  
- .navbar, .navbar a: Fehér szöveg, hover-effektus (#00c3ff), 18-as betűméret.  
- .container-navbar: sötét háttér, kék árnyékkal körbe, lekerekítéssel.  
- .sidebar-news: Fix pozíció, görgethető hírek, sötét háttér, árnyékkal körbe.  
- .szekciók, .cikkek: Cikkdobozok fehér szöveggel, hover-hatás skálázással.  
- .dropdown-menu: Legördülő menü alsó csíkkal.

**B. Világos Téma (style-light.css)**

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.- body, .bodyIndex: Világos zöldes háttér (#b2b6a1c2), sötét szöveg, Arial, Sans-serif betűtípus használata

.

A képen szöveg, Betűtípus, képernyőkép, szám látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.- .navbar, .navbar a: Fekete szöveg, hover effektus(#3e6825), 18-as betűméret

- .container-navbar: Világos háttér, barna árnyékkal körbe, lekerekítéssel.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.  
A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen. - .sidebar-news: Fix pozíció, görgethető hírek,világos háttér, árnyékkal körbe.

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen. - .szekciók, .cikkek: Cikk”dobozok” fekete szöveggel, hover-hatás skálázással.

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen. - .dropdown-menu: Legördülő menü alsó csíkkal, effekttekkel.

**Záró Megjegyzések**

Ez a dokumentáció átfogó képet ad a Fundelio projekt technikai és felhasználói aspektusairól, részletesen ismertetve a fejlesztés folyamatát, az alkalmazott technológiákat, a kód architektúráját és a tesztelési módszereket. A leírás kiterjed mind arra, amit a rendszer az indulásakor elvárt, mind pedig arra, hogyan készülhet a további bővítés és optimalizáció a jövőben.

A jövőbeni fejlesztések során további funkciók és integrációk várhatók, melyek révén a Fundelio platform még komplexebb és felhasználóbarátabb lesz, miközben a pénzügyi szimuláció értéke is növekszik. A rendszer skálázhatóságának, biztonságának és interaktivitásának további finomítása érdekében folyamatosan gyűjtjük a felhasználói visszajelzéseket, és a legjobb megoldások bevezetésével javítjuk az alkalmazás működését.

Bízunk benne, hogy ez a dokumentáció megfelelő alapot nyújt a fejlesztők, tesztelők, valamint a felhasználók számára a Fundelio platform megértéséhez és a jövőbeni fejlesztések sikeres kivitelezéséhez.