**MBank**

Fejlesztői és Bemutató Dokumentáció



Madarász Márk – 2025

1. Bevezető

Az MBank egy JavaFX alapú asztali internetbanki alkalmazás, amelynek célja egy egyszerű banki rendszer prototípusának bemutatása. A projekt lehetővé teszi a felhasználók regisztrációját, bejelentkezését, egyenlegük lekérdezését, átutalások indítását, valamint bizonyos adminisztrációs műveletek végrehajtását.

A projekt fejlesztése tanulási, fejlődési céllal történt, nagy figyelmet fordítottam a strukturált kódolásra és reszponzív felhasználói felület kialakítására, és valósághű funkciók modellezésére.

1. Felhasznált technológiák

* Java 8
* JavaFX (GUI)
* FXML (Scene Builder)
* MySQL adatbázis
* Hibernate
* SHA1 jelszó titkosítás (Apache Commons Codec)
* FontAwesome FX (ikonok)

1. Fő Funkciók
2. Felhasználói regisztráció

* Form validáció: minden mező kötelező
* Egyedi ellenőrzés e-mailre, felhasználónévre és telefonszámra
* Jelszóellenőrzés: minimum 8 karakter, 1 nagybetű, 1 szám
* Automatikus folyószámla generálás

1. Bejelentkezés

* Felhasználónév + jelszó alapú autentikáció
* Belépési napló (adatbázisba rögzíti)
* Admin és átlag felhasználói szerepkör megkülönböztetése

1. Főoldal

* Név szerinti köszöntés
* Aktuális idő + dátum, perce pontosan
* Számla állapot megjelenítés
* Havi bevétel / kiadás összegzés
* Utolsó 4 tranzakció megjelenítése
* Tranzakció indításának a lehetősége
* Harang ikon 🡪 üzenet megtekintése

1. Profil oldal

* A felhasználó személyes adatai itt jelennek meg
* Lehetőség van E-mail, illetve jelszó változtatásra

1. Számlák oldal

* A felhasználó által birtokolt számla adatai jelennek meg
* Egyenleg, számlaszám, valuta
* Havi bevétel / kiadás statisztika diagramm

1. Tranzakciók oldal

* Minden beérkezett, indított tranzakció listázása
* Tartalmazza a dátumot, címzettet, összeget
* Harang ikonra húzva az egeret, megjelenik a tranzakcióval küldött üzenet

1. Hibajelentés

* A ’’Hibajelentés’’ gombra kattintva egy új ablak nyílik meg
* Itt a felhasználó egy szövegdobozba írhatja a hibát vagy javaslatot
* Az üzenetet az Admin láthatja a felületén

1. Rendszerfelépítés

* **Modell és DTO osztályok:** Ezek azok a Java osztályok, amelyek az adatbázis egyes tábláit reprezentálják (pl. UserDto, AccountsDro stb.). A JPA annotációk segítségével az osztályok mezői megfelelő adatbázis oszlophoz vannak kötve.
* **Adatkezelő réteg:** A DAO osztályok biztosítják az adatbázis műveletekhez szükséges logikát. Itt található például a UserDao, AccountDao stb., amelyek a JPA segítségével végrehajtják a szükséges lekérdezéseket és módosításokat. A DAO-k használatával a magasabb rétegeknek (pl. a UI vezérlőknek) nem kell ismerniük az adatbázis részleteit vagy SQL utasításokat – elegendő a megfelelő metódust meghívni.
* **Vezérlők:** Minden fontosabb UI képernyőhöz tartozik egy JavaFX Controller osztály, amely az adott felület eseménykezeléséért és az adatok megjelenítéséért felel. Ilyenek például a LoginController (bejelentkező ablak logikája), a ClientMenuController (ügyfél főoldal és navigáció), AccountsController (számlaadatok oldal) és társai. A vezérlők a háttérben a DAO-kat hívják meg.
* **FXML:** Az alkalmazás felhasználói felületének elemei külön FXML fájlokba vannak szervezve. Minden fő nézet (pl. bejelentkező képernyő, főoldal, tranzakciók oldala, stb.) egy FXML leírást kapott, amely meghatározza a UI elemeket (gombok, címkék, beviteli mezők, panelek elrendezése). Az FXML fájlokban meg vannak adva a vezérlő osztályok is, így a grafikus elemek és a Java logika össze vannak kapcsolva. A tervezés Scene Builder segítségével történt, ami felgyorsította az UI kialakítását és a konzisztens elrendezést.
* **Segédosztályok és egyéb csomagok:** Az application.util csomag tartalmazza a segédfunkciókat, pl. a már említett JpaUtil (JPA inicializálás és EntityManager biztosítása) és a SceneSwitcher segédet, amely az animált jelenetváltásokat valósítja meg. A Session osztály egy egyszerű, statikus munkamenet-kezelő, amely a bejelentkezett felhasználó azonosítóját/neveit tárolja a program futása alatt. Ezen felül vannak erőforrás csomagok is: pl. az application.css mappában stíluslapok, az application.images alatt ikonok és logók (pl. alkalmazás logó), amelyeket a felület használ.

1. Admin felület

Az adminisztrátori felület funkcionalitása eltér a normál ügyfélétől, hiszen itt a cél a rendszerben lévő összes felhasználó és művelet felügyelete. Adminisztrátori fiókkal bejelentkezve az alkalmazás egy admin dashboard-ra navigál, amely kifejezetten a bank alkalmazás belső kezelőfelülete.

1. Admin főmenü

* Új ügyfél létrehozása (regisztrálása)
* Ügyfelek listája (összes ügyfél áttekintése)
* Befizetés rögzítése egy ügyfél számlájára (tranzakció nélkül)
* Beérkezett hibajelentések megtekintése
* Kijelentkezés

1. Adatbázis struktúra

Azalkalmazás egy MySQL adatbázist használ az adatok tartós tárolására. Az adatbázisban több összekapcsolt tábla található, melyek az üzleti logikában előforduló entitásokat tükrözik.

**users** – A felhasználók adatait tartalmazó tábla

* id – Egyedi azonosító
* user\_name – Felhasználónév
* first\_name – Keresztnév
* last\_name – Vezetéknév
* password – Jelszó (titkosított)
* email – E-mail cím
* phone – Telefonszám
* status – Fiók státusza
* createdAt – Regisztráció időpontja
* updated\_at – Frissítés időpontja

**accounts** – A bankszámlák adatait tároló tábla

* id – Egyedi azonosító
* user\_id – Hivatkozás a számla tulajdonosára
* account\_number – Számlaszám
* account\_type – Számlatípus
* balance – Egyenleg
* currency – Valuta
* status – Számla státusza
* created\_at – Nyitás dátuma
* card\_number – Bankkártya szám

**transactions** – A banki tranzakciók listája

* id – Egyedi azonosító
* from\_account - A forrásszámla azonosítója
* to\_account – A cél számla azonosítója
* amount – Összeg
* currency – Valuta
* timestamp – időbélyeg
* message – Közlemény / üzenet

**login\_log** – Bejelentkezési napló

* id – Egyedi azonosító
* user\_id – Felhasználó azonosító
* login\_time – Időpont
* ip\_address – IP cím, ahonnan a bejelentkezés történt
* was\_successful – Logikai érték, sikeres volt-e a bejelentkezés

**reports** – Hibajelentések / visszajelzések

* id – Egyedi azonosító
* user\_id – Felhasználó azonosító
* message – Üzenet
* submitted\_at – Beküldés időpontja

1. Adatellenőrzések és hibakezelés

Egy banki alkalmazásnál különösen fontos, hogy az adatok bevitelekor és a műveletek során megfelelő validációk legyenek, és hiba esetén a rendszer kezelje azokat elegánsan anélkül, hogy inkonzisztens állapotot hozna létre.

Felhasználói input validáció (front-end):

* Regisztrációs űrlap: A vezérlő ellenőrzi, hogy egyetlen mező se maradjon üresen, mielőtt elküldené a regisztrációs kérést. Ha valamelyik hiányzik, figyelmeztetést ad (pl. piros szöveg: "Minden mező kitöltése kötelező!"). Ezen felül azonnali ellenőrzést is végezhet: pl. amikor a felhasználó beír egy e-mail címet, a @ karakter meglétét ellenőrizheti.
* Bejelentkezés: Itt is ellenőrizzük, hogy a felhasználónév és jelszó mező nincs-e üresen, mielőtt megpróbálnánk bejelentkeztetni. Üres értékekkel felesleges az adatbázist lekérdezni.
* Átutalás indítása: A főoldali utalási űrlapnál a következő ellenőrzések vannak: a címzett mező nem lehet üres (kell számlaszám), az összeg mezőbe csak számokat engedünk (pl. tiltjuk a nem szám karakterek bevitelét vagy utólag ellenőrizzük és konvertáljuk), az összegnek pozitívnak kell lennie.

Üzleti logika (back-end):

* Regisztrációnál a UserDao.saveWithAccount előtt meghívódnak az emailExists, phoneExists, findByUserName lekérdezések. Ezek a DAO metódusok a adatbázisból nézik meg, hogy van-e ütközés.
* Bejelentkezésnél a validateUserByUserNameAndPassword belül hash-eli a megadott jelszót és egy criteria query-vel megnézi, van-e olyan user, akinél a userName és password hash egyezik. Ha nincs, null/false értéket ad vissza, amit a LoginController lekezel és "Hibás felhasználónév vagy jelszó" üzenetet jelenít meg.
* Átutalás végrehajtásnál a TransactionDao.makeTransaction előbb ellenőrzi, hogy a fromAccount egyenlege >= amount (tehát van elég pénz). Ha nincs, nem hajtja végre a tranzakciót, hanem visszaad egy hibajelzést a vezérlőnek (pl. kivételt dob vagy false-ot ad). A vezérlő ez alapján kiírja, hogy "Nincs fedezet". Ha van fedezet, akkor folytatja: lekéri a toAccount-ot. Ha a toAccount nem létezik (pl. elgépelt számlaszám), akkor is megáll és jelez (pl. "Címzett számla nem található").
* Adatbázis szintű korlátozások: a séma is tartalmaz constraint-eket, pl. unique indexeket (felhasználónév, email stb.), tehát ha valahogy a kód mégis engedne egy duplikációt, az adatbázis dob hibát (Unique constraint violation).

Hibakezelés és visszajelzés a felhasználónak:

* Hibaüzenetek a UI-n: Ahogy a fenti példák mutatják, minden fontosabb műveletnél van egy Label vagy hasonló elem, amit hiba esetén beállítunk egy üzenetre.
* Kivételkezelés a kódban: A vezérlők try-catch blokkokkal hívják a DAO műveleteket. Ha egy DAO váratlan kivételt dob (pl. adatbázis kapcsolat hiba, vagy egy nem kezelt edge case), a catch blokkban loggoljuk a hibát
* Üres állapotok: Ha pl. egy felhasználónak még nincs tranzakciója (új regisztráció), a tranzakciós listában megjelenhet egy üzenet: "Nincs megjeleníthető tranzakció".

Biztonság és hibák:

* Jelszóvédelem (hash-elés, sosem jelenik meg a jelszó sehol visszafejtve).
* Session fixáció ellen: bár egyjátékos asztali app, de ha kijelentkezik valaki, töröljük a session-t.
* Az admin funkciók nem elérhetők usernek (UI szintjén tiltott, kód szintjén külön szerepkör ellenőrzéssel vált).
* Adatbázis műveleteket csak ellenőrzött paraméterekkel hajtunk végre.

1. Fejlesztési nehézségek és tanulságok

* Kezdetben el kellett dönteni, hogyan válasszuk szét a felületet és logikát, viszont közben lett implementálva a Presenter osztály, de a projekt jelenlegi méretében nincs kihasználva rendesen.
* FXML és Scene Builder: A felületet Scene Builderrel készítettem, viszont rengeteg ötlet, már a felület elkészülte után jutott eszembe (pl. ’’kis szem ikon’’ a jelszó felfedésre), így pár dolog kimaradt, mert teljesen újra kellett volna kezdeni egyes dolgokat.
* JPA/Hibernate integráció: A Hibernate használata jelentősen megkönnyítette az adatbázis, viszont konfigurálni kellett a persistence.xml-t és a függőségeket. Eleinte volt probléma az EntityManagerFactory létrehozásával, mert hiányzott a MySQL JDBC driver betöltése.
* Session kezelés: Eredetileg elegánsabb sessiont akartam, de végül a statikus megoldás mellett döntöttem. Tisztába vagyok vele, hogy nem thread-safe, de elsőre az egyszerűség győzött.

1. Továbbfejlesztési lehetőségek:

* Regisztrációnál:
  + Dupla jelszó bekérés, elírás védelme érdekében
  + Jelszó felfedése, felhasználó kérésére
  + Jelszó erősséget jelző ’’csúszka’’
* Bejelentkezésnél
  + Szintén jelszó felfedése
  + Töltő képernyő vagy animáció
* A felületen:
  + Több fajta számla típus, pl. megtakarítás
  + Közvetlen bankkártyás vásárlás szimulálása
  + Költségkategóriák, pl. ’’élelmiszer’’ vagy ’’rezsi’’
  + Dark mode támogatás
  + Reszponzív dizájn
  + Drag & drop fájlcsatolás a hibajelentésnél
  + UI nyelvválasztás
* Biztonság és autentikáció
  + Erőseb jelszókezelés
  + Kétszintű hitelesítés
  + Session timeout
  + Bejelentkezési kísérletek korlátozása