Programozói dokumentáció

A projekt egy banki szoftvert minimális szinten imitáló konzolos alkalmazás. Funkcionalitásai közé tartozik a felhasználó beléptetése, regisztrálása, rendszer alaphelyzetbe állítása, tranzakciótörténet visszanézése, új indítása és felhasználónév/jelszó cserélése. Az alkalmazás az adatait csv fájlokban tárolja.

A program forráskódjában megtalálható pár angol nyelvű komment is.

A programot fejlesztéskor csupán macOS-en teszteltem, és az alábbi paranccsal futtattam:

gcc -o main main.c screens.c filesystem.c user.c transaction.c utils.c && ./main

A teszteléshez van egy felhasználó kezdetben, admin-admin belépési adatokkal.

Kiindulás

Az app kiindulásként (***main.c***) inicializálja az alapjállapotát a fájlokból, majd az állapot alapján indul a képernyő renderelése. Ezekről mind külön-külön lehet olvasni lentebb.

Fájlkezelés

Egy ***filesystem.h*** *nevű fájlban* a függvénydeklarációkat, egy ***filesystem.c*** nevű fájlban pedig a függvények inicializálását találni. Ezek azok a függvények, amik az alábbi fájlokkal dolgoznak:

- users.csv - a felhasználók adatai
- transactions.csv - a felhasználók tranzakcióinak adatai

Ezek a fájlok pontovesszővel (;) vannak elválasztva, így olvassa be őket a modul két metódusa, a *readUsers* és a *readTransactions* a *readLines* sorolvasó és az openFile fájlmegnyitó függvény segítségével. Ezek fixen megadott fájlnevek alapján dolgoznak, majd egyből az olvasás után létrehozzák belőlük a változókat, egyik esetben a *User*, másikban pedig a *Transaction* típusú objektumokat (ezekről lentebb lehet olvasni).

Minden egyes screen renderelésnél (ennek folyamatáról szintén lentebb lehet olvasni) lementjük az aktuális állapotát ezeknek az adatoknak, vagyis akár egy hirtelen kilépésnél is a lehető legfrissebb adatokkal lehet folytatni azt, amit abbahagytunk. Ezt a függvényt *saveAppData* néven lehet megtalálni., és egyszerű loopolással, majd fájlbaírással dolgozik.

Amennyiben a fájlok megsérülnek, hibás adat kerül bele, vagy bármi váratlan történik, a *resetAppData* függvény gondoskodik arról, hogy minden alaphelyzetbe álljon. Ez azt jelenti, hogy az összes tranzakció törlődik, a felhasználók listája pedig egy elemből fog állni, az admin-admin felhasználónév-jelszó párból álló számlából, aminek kezdőegyenlege 5000 pénz.

Típusok

A program a típusokat egy ***types.h*** nevű fájlban inicializálja, és azt használja minden más helyén az alkalmazásnak. Az alábbi típusok léteznek az alkalmazásban:

- Screen (enum) - ez egy enum típus, amiben felsorolva vannak a lehetséges képernyőállások, amik nagyban határozzák meg az applikáció működését
- AppEvent (enum) - szintén egy enum típus, az alkalmazás különböző eseményeit tárolja magában. Ebből 3 van:
  - EVENT_CONTINUE - a program fut tovább
  - EVENT_EXIT - a program kilép
  - EVENT_RESET - a program reseteli saját magát
- String - egy struct típus, ami a dinamikus szövegkezelésre van
- Transaction - egy struct típus, ami a tranzakciókkal való munkára van. Egy objektumként és láncolt listaként is működik. Tárolja:
  - idTo (int) - felhasználó végpont (hova ment a pénz)
  - idFrom (int) - felhasználó kezdőpont (honnan jött a pénz)
  - amount (int) - mennyi pénz utazott
  - date (String*) - utalás dátuma
  - usernameTo (String*) - végfelhasználó neve
  - usernameFrom (String*) - kezdőfelhasználó neve
  - next (Transaction*) - következő tranzakciója a felhasználónak
- User - szintén egy struct típus, a felhasználót modellezi, illetve ez is egyszerre működik objektumként és láncolt listaként. Tárola:
  - id (int) - felhasználó azonosítója
  - account (int) - felhasználó egyenlege
  - username (String*) - felhasználónév
  - password (String*) - jelszó
  - next (User*) - következő felhasználó a listában
  - transactions (Transaction*) - a felhasználóhoz tartozó tranzakciók
- AppState - végül szintén egy struct, ami az applikáció állapotát (state) modellezi le. Ez mindenhova továbbítva van. Tagjai:
  - loggedUser (User*) - belépett felhasználó
  - users (User*) - felhasználók láncolt listája
  - screen (Screen) - aktuális kijelzőállapot
  - appEvent (AppEvent) - aktuális alkalmazásesemény
  - errorMessage (String*) - hibaüzenet amit éppen kiírunk
  - message (String*) - üzenet amit kiír az app
  - date (String*) - aktuális dátum

Képernyőkezelés

A szoftver az állapota alapján rendereli a képernyőket egymásutánban a ***screens.c*** és ***screen.h*** fájlban lévő kóddal. A különböző képernyőkre egymásból lépnek egy fő függvény segítségével, ami a *renderScreen* névre hallgat. Minden újrarenderelésnél az előző képernyő tőrlődik, ilyenkor „tiszta lappal” indulunk.

Ilyenkor a törlés az első művelet, majd a *renderHeader* függvény kiírja az alkalmazás fejlécét, ami a neve, aktuális idő, illetve bejelentkezett felhasználó esetében a név és az egyenleg. Ezután pedig a különböző képernyő egyéb részeit íratja ki a program.

3 képernyő esetében navigálást tudunk véghez vinni, ilyenkor egyszerű renderelés, majd opcióválasztás történik egyéb komplex logika nélkül. Ezek az oldalak ilyenek:

- S_WELCOME - *renderWelcome és pickWelcomeOption* függvények. Az opciókat alkalmazás státuszok szerkesztésével választja ki
- S_HOME - *renderHome és pickHomeOption* függvények. Az opciókat alkalmazás státuszok szerkesztésével választja ki
- S_ACCOUNT_SETTINGS - *renderAccountSettings és pickAccountSettingsOption* függvények

A többi képernyő pedig:

- S_LOGIN - *renderLogin* rendereli ki, majd a *loginUser* függvénnyel kezdődik a beléptetés, amivel a felhasználótól kérünk inputot a felhasználónévhez és jelszóhoz. Ezután a függvény megnézi hogy létezik-e a felhasználó és jól adtuk-e meg az adatokat. Ha igen, bejelentkeztet, ha nem, akkor meg kiírja a hibát és visszalép.
- S_REGISTER - *renderRegister* írja ki, majd a *registerUser* függvénnyel regisztrál a felhasználó. Szintén 2 adatot kér be a program, amikkel regisztrálunk. A függvény megnézi, hogy létezik-e felhasználó a megadott névvel, és ha igen, hibával visszadob a nyitóoldalra. Ha nem létezik, létrehozza az új usert, majd szintén visszadob a nyitóoldalra, ahonnan bejelentkezhetünk
- S_ACCOUNT_HISTORY - egyszerűen kirendereljük a *renderAccountHistory* függvénnyel a számlatörténetet a láncolt lista loopolásával
- S_NEW_TRANSACTION - új tranzakciót indíthatunk a *renderNewTransaction* függvénnyel, ami renderelés után megkérdezi kinek akarunk küldeni (felsorolva a felhasználókat loopolással), majd hogy mennyit akarunk. Mindkét bemenet ellenőrizve van különböző feltételek alapján. A sikeres a bemenet, akkor létrejön az új tranzakció, mindkét félnek pedig megváltozik az egyenlege.
- S_CHANGE_USERNAME - meglehet változtatni a felhasználónevet a *renderChangeUsername* függvényből indulva. Ellenőrzése kerül, hogy létezik-e a felhasználó már
- S_CHANGE_PASSWORD - a *renderChangePassword* függvény megváltoztatja user input alapján a jelszót az aktuális felhasználóhoz.

Felhasználóműveletek

A felhasználóműveletekhez a ***users.c*** és ***users.h*** fájlokat használja a kód. Ezekben vannak olyan függvények, amik segítenek a különböző felhasználómanipulációkban.

Az *insertUser* függvény a láncolt listába helyez be a paraméterek alapján új felhasználót rekurzióval.
A *getUser* függvény megkeres egy felhasználót, majd visszaadja a pointert hozzá.
Az *isSameUser* bool-t ad vissza az alapján, hogy a két megadott felhasználó megegyezik-e.
A *freeUser* felszabadítja a paraméterben kapott felhasználó adatait a memóriából.
Végül pedig a *freeUsers* egy listát kap paraméterként, és minden felhasználót a láncolt listából felszabadít a végéről indulva.

Tranzakcióműveletek

A tranzakcióműveletek ugyanolyanok, mint a felhasználóknál, itt viszont a ***transaction.c*** és ***transaction.h*** fájlokban találjuk meg a kódjukat.

Az *insertTransaction* a láncolt listához ad új tranzakciót a paraméterek alapján.
A *freeTransaction* egy darab, paraméterként kapott tranzakciót szabadít fel a memóriából.
A *freeTransactions* pedig szintén egy Transaction* típust kap, viszont láncolt listaként kezeli, és a végéről kezdve minden tranzakciót felszabadít.

Segédfüggvények

Az alkalmazás tartalmaz segédfüggvényeket, amiket a ***utils.c*** és ***utils.h*** fájlban lehet megtalálni. Ezek javarészt szövegműveletek, de tartalmaznak mást is.

getStringFromUser: bemenetet ad vissza a felhasználótól több itteni függvényt segítségével.
stringInit: inicializál egy String típusú változót dinamikusan
stringPush: hozzárak egy String típushoz egy karaktert
stringMalloc: allokál egy String típus szövegmezőjébe a megadott mérettel.
stringCompare: összehasonlít 2 darab String-et, és igaz bool-al tér vissza, ha megegyeznek, és hamis bool-al, ha nem
stringCopy: hozzárak egy létező String objektumhoz egy megadott szövegrészt
toString: a megadott String objektumból kiveszi, és visszatérésként visszaadja az abban lévő szövegrészt
stringReplace: kicseréli egy String objektum szövegadatát egy megadottra
stringFree: felszabadít egy String objektum által használt memóriarészt
clearInputBuffer: üríti a bemeneti buffert, máskülönben tele lesz bugokkal az alkalmazás
getActualDate: egy megadott String változóba helyezi az aktuális dátumot

Egyéb megjegyzés

Észrevehető, hogy az alkalmazás készítéséhez megihletett a modern frontend keretrendszerek stílusa, például a minden elérhető alkalmazás állapot, és folyamatos újrarenderelődés. Sajnos viszont teljes másoláshoz nem volt kapacitásom, így sem view-service-model, sem pedig render state management-et nem kapott az alkalmazás.

Írta: Pór Gergő