Boola dokumentáció

Készítette: Pálvölgyi Patrik, Zakar Bálint, Ótott-Kovács Tamás

Boola dokumentáció

## Az ötlet

### A probléma

Az ötletünk az alpszik hogy az embereknek nincs egy olyan pénzügyi szoftvere ami egyhelyen feljegyzi az embereket összes költekezését.

**2. A kivitelezés nagyvonalakban**

1. *Mit csináltunk?*

A végső rendszer három részből áll: egy asztali programból, egy webes felületből és egy mobilalkalmazásból. Ezek mind lehetővé teszik költések felvételét, egyenleg feltöltését és más, egyszerűbb adminisztrációs feladatok végrehajtását. A weboldal, az asztali program, és telefonos kiadás funkcionalitásban megegyeznek.

1. *Tech Stack*

A projektben felhasznált technológiák igen sokszínűek. Igyekeztünk a legmodernebb eszközöket használni, hogy megbízható és időtálló legyen a munkánk.

Az alkalmazások webes technológiával, Vue.js keretrendszerrel készültek. Windowsra és linuxra Tauri-val, Androidra pedig Capacitor.js segítségével portoltuk át a webalkalmazást. Ez nagymértékben felgyorsította a fejlesztési folyamatot, hiszen minimális változtatással tudtunk több rendszert is támogatni. Ezen kívül használjuk még a Piniát is állapotkezelésre, a Vue Routert az oldalak közti navigációra, az Axiost a HTTP kérések egyszerűsítésére, a Bootstrapet pedig a stílusozásra.

A backendet Kotlinban, a Ktor keretrendszer használatával alkottuk meg, amit a Heroku felhőszolgáltató tárol a szerverein és tesz elérhetővé az interneten. Az adatbázis feladatokat egy PostgreSQL adatbázis látja el, ugyancsak a fenti szolgáltató által tárolva. A szervert Docker konténerbe helyezzük el, amit a Heroku futtat.

1. *Mostani funkciók*

Jelen pillanatban az alábbi műveletek lehetségesek az alkalmazásokban:

* Bejelentkezés vagy regisztráció
* Profilok készítése, megtekintése és törlése
* Költések létrehozása, megtekintése, keresése és törlése
* Egyenleg ellenőrzése és feltöltése
* Fiókadatok frissítése

1. *Mi lesz még?*

A jövőben tervezünk a jelenlegiek mellé további funkciókat is bevezetni, növelve a felhasználói élményt. Ezek közé tartozik a profilok megosztása, a fiók összekötése más pénzügyi alkalmazásokkal, hogy a költések automatikusan is bekerüljenek, címkék kiválasztása egy listából és kimutatások készítése az időszakos kiadásokról.

**3. A Backend**

1. *Tech Stack*

A projekt backend része egyáltalán nem szokványos, Kotlinban, Ktor alapon íródott. Ennek fő oka a jobb fejlesztői élmény volt, ugyanis az IntelliJ IDEA szoftver kiválóan integrálódik a keretrendszerrel és a nyelvvel is. Továbbá a megszerzett korábbi tapasztalat is a Kotlin mellett szólt. A Kotlin kódot a Gradle buildeszköz fordítja bájtkódra és készít belőle futtatható állományokat. A felhőben való futtatáshoz egyes konfigurációs adatokat környezeti változókban kellett eltárolni, de a helyi munka megkönnyítése érdekében megoldottuk a fájlból történő konfigurációt is. Ezt a dotenv-kotlin könyvtár tette könnyebbé.

A sok felhőszolgáltató közül a Heroku-ra esett a választásunk, egyszerűen azért, mert a GitHub diákoknak ad hozzá ingyen keretet, ami fel tudtunk használni a projekt elkészítéséhez. A Heroku további előnye volt, hogy támogatja a Docker konténerek futtatását, ami nagyban segítette a munkánkat.

1. *Projektstruktúra*

A fájl- és mapparend a szabványos Java struktúrát követi. A backend mappa gyökerében található a build fájl (build.gradle.kts), a helyi konfigurációs fájl (.env) és a dockerfile (Dockerfile). Maga a kód az src/main/kotlin/com/boola/ útvonalon érhető el. Itt a fájlok három mappába lettek rendezve:

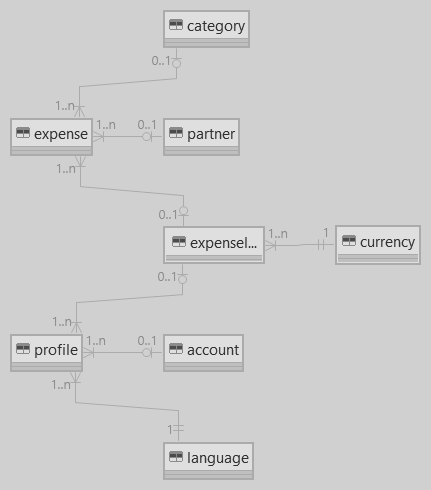
* a controllers mappában a kontroller és a kontroller factory,
* a models mappában a modellosztályok,
* a plugins mappában a Ktor beállításához szükséges fájlok vannak.

Az alkönyvtárakon kívül szerepel még egy Application.kt nevű fájl. Ez az program kiindulópontja, ez állítja be és indítja el a szervert.

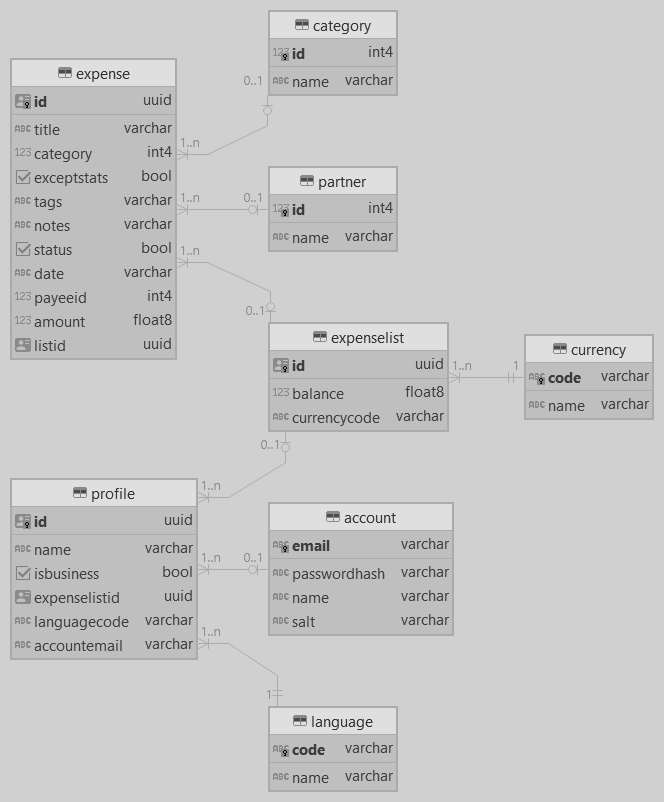
1. *Az adatbázis*

Az adatbázis tervezéskor a fő szempont az volt, hogy minden olyan adatot, amit a programnak tárolnia kell, el is tudja tárolni. Az táblák középpontjában a profil (profile) áll, a legtöbb tábla ehhez kapcsolódik. Ebben tároljuk a profil azonosítóján kívül (id) a nevét (name), a nyelvet (languageid), és hogy üzleti-e a profil (isbusiness). A fiók (account) azonosítja a profil tulajdonosát e-mail cím alapján. A számla (expenselist) tárolja a pénznemet (currencycode) és az aktuális egyenleget (balance), illetve a kiadások (expense) is erre hivatkoznak (expenselistid). A kiadás a legnagyobb táblánk, mezők és sorok tekintetében is. Minden kiadásnak van azonosítója (id), címe (title), kategóriája (category), jelzése, ha szeretnénk, hogy kihagyjuk a statisztikából (exceptstats), címkéi (tags), megjegyzései (notes), esedékességi dátuma (date), jelzése, hogy kiadás ki van fizetve vagy sem (status), kezdeményezettje (partnerid) és összege (amount).

Az adatbázis egyed-kapcsolat diagramja:



Az adatbázis bachmann-ábrája:



1. *DbConnector és DataController*

Az adatbázishoz való kapcsolódást több módon is meg lehet valósítani. Java-ban, amivel a Kotlin együttműködik, az egyik népszerű megoldás a JDBC API.

A közös kommunikációhoz először létre kell hozni egy kapcsolatot (Connection). Ezt a DbConnector osztályban tesszük meg, ami felállítja a kapcsolatot, majd átadja használatra (getConnection()), általában egy DataControllernek.

A DataController osztály a backend egyik fő eleme, ez intézi a tranzakciókat az adatbázis felé. Minden végrehajtandó művelet SQL lekérdezése tárolva van egy PreparedStatement-ként. Amikor egy olyan kérés érkezik a backendre, amihez az kell az adatbázis, a DataController beállítja a lekérdezés paramétereit és elküldi azt, a választ pedig értelmezi és visszaadja a végpontnak.

1. *A DataControllerFactory*

Ez a megoldás remekül működik, de akadtak vele problémáink. Minden alkalommal új kapcsolatot készíteni nem túl hatékony, ráadásul a távoli adatbázissal mindössze 20 kapcsolatot tarthattunk fenn egy időben. Emiatt szükség volt a kapcsolatok újrafelhasználására. Ezt a szerepet látja el a DataControllerFactory.

A DataControllerFactory a szerver indulásakor egy megadott számú DbConnector-t hoz létre és az ezekből készített DataController-eket adja ki a REST API végpontoknak (DataControllerFactory.getController()). Miután az API már nem tart igényt a DataController-re, visszaadja azt a DataControllerFactory-nak, amit később újra ki tud adni, ha szükség van rá. Így tudunk figyelni arra, hogy ne legyen túl sok DataController, miközben biztosítjuk, hogy minden kéréshez jutni fog egy DataController.

1. *Hitelesítés, JWT, biztonság*

A backend nagyon sok érzékeny adat feltöltését és elolvasását teszi lehetővé, emiatt szükség volt valamilyen hitelesítésre. Ehhez mi a JWT-t (JSON Web Token) választottuk, mert egy modern megoldás, egyszerű az implementálása és az iskolában is ajánlották.

A Ktor keretrendszer maga is támogatja a JWT alapú hitelesítést, a beállítás ehhez illően nem volt túl bonyolult. A plugins/Security.kt fájlban történik a fő konfiguráció, a kibocsátó (issuer), a célközönség (audience) és az ellenőrzés folyamatának megadása után már készen is áll a rendszer a JSON Web Tokenek kiadására.

A hitelesítést a “/login” végpontra küldött POST kéréssel kezdeményezhetjük, ami tartalmazza a belépő fiók adatait. Sikeres belépés után pedig a kliens kap egy hozzáférési (access) tokent a belépéshez és egy frissítő (refresh) token a lejárt token megújításához.

A JWT-n kívül is nagyon komolyan vettük a szerveroldali biztonságot, a Heroku alapértelmezetten HTTPS-en keresztül kommunikál, a feltöltött jelszavakat BCrypt-tel lehash-eltük és természetesen mindenki csak azokhoz az adatokhoz fér hozzá, ami az övé vagy nyilvánosan elérhető.

1. *Tesztelés*

A szerver megbízható működése érdekében készítettünk pár tesztet, amivel ellenőrizni tudjuk a munkánk helyességét.

A tesztjeinket két kategóriába soroljuk: vannak egységtesztek és API tesztek.

Az egységteszteket a test/kotlin/com/boola mappába helyeztük el. A tesztkészítéshez a Kotlin saját kotlin.test könyvtárát használtuk. Ezek a tesztek leginkább a saját osztályainkat teszik próbára, hogy mindig jól működjenek.

Az API tesztek a teljes backend funkcionalitását ellenőrzik, afféle integrációs tesztként. HTTP kéréseket küldenek megadott végpontokra és eltárolják a választ. A tesztek a test/APItest.http fájlban találhatók és az IntelliJ IDEA Ultimate fejlesztői környezetből futtathatók. Ezeket a teszteket nem tudtuk automatizálni, hanem szükségszerűen indítottuk el őket és magunk elemeztük az eredményeket.

1. *CI/CD, GitHub Actions*

Az automata teszteket szerettük volna, ha maguktól lefutnak, amikor változtatunk a programon. Erre a GitHub Actions-t találtuk a legkézenfekvőbbnek, hiszen így a tesztek minden push és pull request után lefutnak és láthattuk, hogy a változások érintik-e más összetevők működését.

A GitHub Actions definíciók a <projekt gyökere>/.github/workflows mappában találhatóak.

A backendet a Build and test backend munkafolyamat teszteli. Gradle-lel lebuildeli a projektet és teszteli a programot.

Van egy másik munkafolyamat is, ami a tárhelyre való feltöltést könnyíti meg. Ezt minden pénteken éjfélre ütemeztük.

1. *Deploy to Heroku*

A Heroku szerverünkre történő feltöltés, vagyis a deployment szinte elengedhetetlen volt számunkra, hiszen enélkül csak igen nehezen tudtunk haladni a backend leprogramozásával.

A Heroku miatt néhány fontos adatot, például az adatbázis címét és a szerver portját környezeti változókban kellett eltárolnunk. Azokra a helyzetekre, amikor nem akartunk környezeti változókat létrehozni, például az iskolában, lehetővé tettük a konfiguráció .env fájlból történő beolvasását is.

Maga a deployment kézzel is megtörténhet a Heroku CLI programjával, de mi automatizáltuk a folyamatot. Péntekenként a fentebb említett GitHub Action lebuildeli a projektet, ebből létrehoz egy Docker képet és azt feltölti a Heroku szerverére. Ott készítenek a képből egy konténert, majd azt elindítják.

1. *Nehézségek*

A backend fejlesztése közben több akadályba ütköztünk. Ezekre kihívásként tekintettünk és végül sikeresen túljutottunk rajtuk.

Az egyik legproblémásabb rész a DataControllerFactory volt, ahol a rendszer működéséből adódóan nagyon figyelni kellett egyes dolgokra. A legnagyobb az volt, hogy kézzel gondoskodtunk arról, hogy minden kérés után ténylegesen visszatérjen a DataController a DataControllerFactory-hoz, ami lehetséges hibaforrás. Sokszor volt gondunk abból, hogy elfogytak a kontrollereink, mert egy-két API-nál elmaradt a visszaadási metódus meghívása. Egy másik hasonló, bár nem olyan katasztrofális hiba abból volt, hogy néhány API végpont többféle választ is tud adni és mindegyik után visszaadta a kontrollerét. Ez azt okozta, hogy a DataControllerFactory-nak úgy tűnt, hogy néha több kontrollere volt, mint ahányat készített induláskor. Persze ez nem volt igaz, a plusz kontrollerek valójában már a DataControllerFactory-ban léteztek, csak többször is hozzá lettek adva. Szerencsére a megoldás igen egyszerű volt: a kéréseket fogadó metódusokat válaszadás és a kontroller visszajuttatása után leállítottuk, ezért csak a “beadás” kérésenként csak egyszer történt meg.

Másrészt a különleges Tech Stack-ünk különleges kihívásokkal is járt. Az iskola biztonsági beállításai nem engedték, hogy az ottani gépeken a szervert futtassuk és az adatbázishoz kapcsolódjunk, ezért volt szükség arra, hogy a backend hostolását máshol oldjuk meg.

A deployment-tel is merültek fel gondok. A Heroku kétféle feltöltést támogat: git-en keresztül vagy Docker konténerként. Először mi az első opcióval próbálkoztunk, úgy gondoltuk, hogy egyszerűbb lesz a dolgunk, ha nem használunk még egy külön technológiát. Sajnos nem lett igazunk, az elvárt projektstruktúra miatt a kiadást végző GitHub Action nem találta meg a backend projekt helyét és típusát, ezért nem bírta elindítani a szervert. Számtalan próbálkozás után végül a Docker konténerek használata jelentett egy jó megoldást számunkra.

**4. A frontend**

1. *Design tervek*

A design-nál próbáltunk arra törekedni hogy a felhasználók szemszögéből letisztult és egyszerű legyen minél felhasználóbarátabb.

1. *Tech Stack*

A frontend alapját a Vue.js JavaScript keretrendszer adja, a többi használt technológia ehhez igazodik. Jó példa erre a Vue Router, a Pinia és a vue-toastification. Formázásra a Bootstrap-et, HTTP kérésekhez pedig Axios-t használunk. Egyéb kisebb könytárak közé tartozik a cors, az uuid, illetve a pinia-plugin-persistedstate.

A projekt buildelésében segít a Vite, desktopon az Tauri, mobilon pedig a Capacitor.js.

1. *Projektstruktúra*

A gyökérkönyvtárban látható a readme fájl (README.md), a függőségeket tartalmazó fájl (package.json), a fő HTML fájl (index.html) és a Vite konfigurációs fájl (vite.config.js). A kódfájlok az src/ mappában vannak, ezen belül külön mappákba rendeztük el őket, components, views és stores néven. A router mappában csak az index.js fájl található, amiben az oldalak útvonalai vannak megadva. Az App.vue fájl a fő oldal, amibe a Vue Router elhelyezi az aloldalakat (views). A view-k komponensekből állnak.

Az oldalhoz szükséges egyéb fájlokat a public könyvtárba tettük. Ezek közé tartozik az oldal betűtípusa és az alap stílusfájl.

1. *view-k, komponensek*

Az oldal fő részei a view-k és a komponensek. A view-k tartalmazzák az oldal legfontosabb részeit például a HomePage.vue és a Login.vue. A komponeneseket arra használtuk az oldalon hogy a view-kat kiegészítsük.

1. *HomePage*

Az alkalmazásba való belépés után a főoldal köszönti felhasználót, ahol kap egy áttekintést a profilja pénzügyi helyzetéről. Láthatja az egyenlegét, a két legutolsó költését és kiválasztott profil adatait.

1. *NewExpense*

Ezen az ablakon tudjuk felvinni a különböző költéseinket.. Vannak különböző kategóriák amikkel meg tudjuk adni a költések típusát , tudjuk őket címkével ellátni és azt is megadhatjuk hogy már kifizettük-e

1. *ExpenseList (költések)*

A költések oldalon tekinthetjük meg feltöltött költségeinket. Minden kiadásnak látható a neve, hogy kifizettük-e vagy sem és az esedékesség dátuma. A költések mellett található ‘Törlés’ gombbal törölhetjük őket, rájuk kattintva pedig a megjelenő oldalsávban több információt is megtudhatunk róluk.

Ha keresünk egy bizonyos költést, segítségül hívhatjuk az oldal tetején lévő keresőmezőt, amibe a kiadás nevét begépelve megjelennek a találatok.

1. *TopUp*

Ha költségeket szeretnénk felvenni akkor szükségünk lesz egyenlegre is. ezen az oldalon feltölthetjük az egyenlegünket amiből automatikusan levonja a költéseket.

1. *Nehézségek*

Az egyik akadály, amivel szembesülnünk kellett, nem is a technológiával volt kapcsolatos, hanem magunkkal. Rá kellett jönnünk, hogy egyikünknek sincs érzéke az interfész design-hoz, vagy egyáltalán a front-end fejlesztéshez. A feladat megoldásához viszont mindenképp szükség volt egy vonzó felhasználói felületre. Egy weboldal segítségével készítettünk egy színpalettát, választottunk egy betűtípust és kiszabtunk pár irányelvet, ezek alapján készítettük el az alkalmazás oldalait.

Egy másik dolog, amin fennakadtunk, az asztali program készítése. Mivel a WPF-ben készült verzió nagyon lassan haladt és nem tudtuk, hogy készen lesz-e a határidőre, a webes felület asztalra való átvitele mellett döntöttünk. Ehhez eleinte az Electron.js könyvtárat használtuk, de mint kiderült, ez nem működik együtt a Vue Routerrel, úgyhogy váltanunk kellett. Végül a Tauri mellett tettük le a voksunkat, ami kiválóan működött.

**5. A csapatmunka**

1. *A csapat*

A projekten hárman dolgoztunk: Pálvölgyi Patrik, Zakar Bálint és Ótott-Kovács Tamás.

Ugyan mindhárman egyenlő mértékben dolgoztunk, közülünk Patrik nőtt ki, mint projektvezető. Övé volt az ötlet és ő csinálta a munka legnagyobb részét. Bálint legtöbbet a frontend designban segédkezett, Tamás pedig a backendet programozta. Természetesen mindenki belekóstolt a fejlesztés összes részébe, hogy full-stack tapasztalatunk is legyen.

1. *Munkaelosztás*

Patrik fő dolga a frontend részeinek elkészítése volt, ezen kívül hozzá fűződik a projekt tervek megírása, a backenden a currency API, az asztali és mobil platformra portolás és a csapat általános koordinációja.

Bálintnak nem volt ennyi feladata, ő inkább a dizájnért és az oldal reszponzivitásáért felelt. Emellett a category API is az ő munkáját dicséri.

Tamás fejlesztette a backendet. Az összes többi REST API, az osztályok és az adatbázis mind ő általa valósult meg. A frontenden a bejelentkezés és a regisztráció ablakot készítette el és segített egyes bonyodalmak elsimításában. Továbbá a GitHub Actions-szel automatizált tesztelés és feltöltés is az ő megvalósított ötlete volt.

1. *A GitHub jó oldala*

Közös munkánk fő színtere a GitHub volt. Itt tároltuk a kódot, osztottuk el egymás között a feladatokat, teszteltük a kódot és projektet menedzseltünk.

A GitHub-on keresztüli csapatmunkát hamar megszoktuk, az elvégzendő feladatoknak issue-kat írtunk, Projects-ben követtük állapotokat, milestone-okat állítottunk fel, hogy kövessük a projektrészek haladását. Az elvégzett munkát külön branch-ekbe töltöttük fel és pull requesteket küldtünk, hogy az egyszerre írt kódot könnyedén tudjuk egyesíteni.

1. *A GitHub rossz oldala*

Azonban a GitHubon történő munkának megvoltak a maga hátulütői.

A beküldött pull request-ek sokszor tartalmaztak összeütközéseket, amiket a régebben leváló branch-ek miatt többször is meg kellett oldanunk.

Jó néhány alkalommal az is előfordult, hogy ugyan feltöltöttük és be is olvasztottuk a változásokat a fő branch-be, mégis nyomuk veszett, vélhetően egy másik, régebbi branch automatikus beolvadása miatt. Ezeket a kódokat kézzel vissza kellett írnunk.

1. *Kommunikáció*

A csapat a kapcsolatot leginkább személyesen tartotta az iskolában, hiszen ez a kommunikáció legegyszerűbb és leghatékonyabb módja. Beszélgetés közben látjuk, hogy mit csinál a másik és mi is tudunk neki helyben segíteni.

Amikor távol voltunk egymástól, kénytelenek voltunk más módszerekhez fordulni. Ebben segített nekünk a GitHub, de még jobban a Discord. Itt tudtunk beszélgetni és akár azt is megnézni, hogy mi történik a csapattársunk számítógépén, ha segítségre volt szüksége.

**6. API dokumentáció**

1. *Backend*

A backend dokumentációért lásd a backend/index.html fájlt, ami elérhető [itt](Backend%20Dokumentáció/index.html).

1. *Frontend*

## *A kiadások kezelése*

***Komponens adatai:***

* *hostName: A háttér API végpont URL-címe.*
* *authToken: A bejelentkezett felhasználó jogosultság token-e.*
* *expenseListId: A bejelentkezett felhasználó aktuális kiadási listájának azonosítója (az expenseStore-ból származik).*
* *filtered: Szűrt kiadások listája (reaktiv referencia).*
* *beker: A keresési szűrőhöz használt bemeneti mező értéke (reaktiv referencia).*
* *expense: A szervertől lekérdezett kiadások tömbje (reaktiv referencia).*

***Függvények:***

* *filter(): A keresési szűrő alkalmazása a beker érték alapján.*
* *Delete(id): Kiadás törlése azonosító alapján.*
* *selectExpense(expense): Kijelölt kiadás beállítása az expenseStore-ban és az oldal átirányítása a megjelenítési nézetre.*

***Sablon:***

* *A sablon egy listát jelenít meg a felhasználó kiadásairól.*
* *Lehetőség van a kiadások szűrésére a keresőmező segítségével.*
* *Minden kiadás esetén megjelenik a neve, státusza (fizetendő/kifizetett), dátuma és egy törlés gomb.*
* *A kiadásra kattintva a felhasználó a kiadás megjelenítési nézetére navigál.*

***API meghívások:***

* *A komponens az axios könyvtárat használja a háttér API elérése érdekében.*
* *Lekérdezi a kiadások listáját a felhasználó aktuális kiadási listájának azonosítója alapján.*
* *Törli a kiadást azonosító alapján.*

***Eseményvezérlés:***

* *A filter() függvény a beker érték változásakor fut le, és ennek megfelelően frissíti a filtered listát.*
* *A Delete(id) függvény a törlés gombra kattintáskor fut le, megerősítést kér a felhasználótól, majd törli a kiadást az API-ból és frissíti a kiadások listáját.*
* *A kiadás elemre kattintáskor a selectExpense(expense) függvény fut le, beállítja a kijelölt kiadást az expenseStore-ban, és átirányítja az oldalt a megjelenítési nézetre.*

***Megjegyzések:***

1. *Ez a dokumentáció a rendelkezésre álló kódból készült, a pontos API endpoint-ek és válasz séma a szerver implementációjától függhet.*
2. *A expenseStore és profileStore pincéreket a kód valószínűleg nem definiálja ebben a fájlban, de valahol másutt a Vuex tárolóban.*
3. *A vue-toastification könyvtár opcionális, a hibák és sikerek megjelenítéséhez használható.*

## *Kiadások megjelenítése*

***Komponens adatai:***

* *hostName: A háttér API végpont URL-címe.*
* *authToken: A bejelentkezett felhasználó jogosultság token-e.*
* *store: expenseStore pincér (Vuex tároló)*
* *profilStore: profileStore pincér (Vuex tároló) - feltételezhetően a profil adatokat tartalmazza*
* *expense: Kijelölt kiadás adatai (reaktiv referencia).*
* *currency: A kiadás pénzneme (reaktiv referencia).*
* *categoryName: A kiadás kategória neve (reaktiv referencia).*

***Függvények:***

* *closeWindow(): Bezárja a kiadás megjelenítési ablakot a kijelölt kiadás értékét nullára állítva.*

***Sablon:***

* *A sablon a kijelölt kiadás részleteit jeleníti meg.*
* *Tartalmazza a kiadás nevét, összeget, pénznemet, kategóriát, státuszt (fizetve/nincs fizetve), dátumot, címkéket és megjegyzést.*
* *Bezárás gombbal is rendelkezik.*

***API meghívások:***

* *A komponens az axios könyvtárat használja a háttér API eléréséhez.*
* *Lekérdezi a kiadáshoz tartozó kategória nevét a kategória azonosítója alapján.*

***Eseményvezérlés:***

* *Az expenseStore pincére változásakor a komponens frissíti a kijelölt kiadás adatait és a kategória nevét az API-ból.*

***Reaktiv tulajdonságok:***

* *expense: A store.selectedExpense értékéhez kötődik, így a kijelölt kiadás változásakor frissül.*
* *currency: A kiadási lista pénznemét a profilStore.expenseListId alapján lekérdezett kiadási lista adataiból állítja be.*
* *categoryName: A kijelölt kiadás kategória azonosítója alapján az API-ból lekérdezett kategória névvel frissül.*

***Stíluskészítés:***

* *A megjelenítést a scoped stílusokkal definiálják.*
* *A megjelenítés különböző képernyőméretekhez igazodik.*

***Megjegyzések:***

* *Ez a dokumentáció a rendelkezésre álló kódból készült, a pontos API endpoint-ek és válasz séma a szerver implementációjától függhet.*
* *A kód feltételezi, hogy az expenseStore és profileStore pincérek definiálva vannak a Vuex tárolóban.*

## *Az oldalsó menü*

***Komponens állapota:***

* *A komponens nem használ reaktív állapotváltozókat.*

***Függvények:***

* *A komponens nem definiál függvényeket.*

***Sablon:***

* *A sablon a bal oldali, fixált elhelyezkedésű oldalsó menüt jeleníti meg.*
* *A menüpontok router linkeket használnak a navigációhoz az alkalmazás különböző oldalai között.*
* *A MenüStore pincére isMainMenuDisplayed tulajdonsága határozza meg, hogy a menü megjelenjen-e.*
* *A MenüStore pincére isMainMenuOpened tulajdonsága határozza meg, hogy a menü ki legyen-e nyitva.*
* *A MenüStore pincére állapotváltozásait a Vue 3 komponens beépített reaktiv tulajdonságainak segítségével olvassuk be.*

***Stíluskészítés:***

* *A megjelenítést a scoped stílusokkal definiálják.*
* *A stílusok a var(--main-background), var(--sec-background), var(--sec-text-color), var(--border-radius), és var(--menu-def-width) CSS változókat használják, amelyeket valószínűleg a globális stíluslapban definiáltak.*
* *A menü alapértelmezetten el van rejtve, és a MenuStore.isMainMenuOpened tulajdonság értéke alapján nyílik ki.*
* *A menüpontok szövege alapértelmezetten rejtett, és a MenuStore.isMainMenuOpened tulajdonság értéke alapján válik láthatóvá.*

***Használt komponensek:***

* *RouterLink (vue-router): Router link a navigációhoz.*
* *RouterView (vue-router): A router nézet megjelenítési helye.*

***Kommunikáció:***

* *A komponens a MenuStore pincére állapotváltozásaira reagál a Vue 3 reaktiv tulajdonságok segítségével.*

***Megjegyzések:***

* *A kód feltételezi, hogy a MenuStore pincért definiálták a Vuex tárolóban.*

## *A profil menü*

***Komponens állapota:***

* *A komponens nem használ reaktív állapotváltozókat.*

***Függvények:***

* *Dob(): Kijelentkezési logika.*
  + *Elrejti a menüket a MenuStore segítségével.*
  + *Megjelenít egy "Viszlát!" értesítést a vue-toastification könyvtárral (opcionális).*
  + *Törli a sessionStorage tartalmát.*
  + *A profileStore pincér profile és email tulajdonságait nullára állítja.*

***Sablon:***

* *A sablon egy legördülő profiladat-menüt jelenít meg az alkalmazás jobb felső sarkában.*
* *A menü a felhasználó profil ikonjára kattintva jelenik meg és tűnik el.*
* *A MenüStore pincére isProfileMenuDisplayed tulajdonsága határozza meg, hogy a menü ikonja megjelenjen-e.*
* *A MenüStore pincére isProfileMenuOpened tulajdonsága határozza meg, hogy a menü ki legyen-e nyitva.*
* *A menü két elemet tartalmaz:*
  + *Profilok: RouterLink a profilok oldalához.*
  + *Kijelentkezés: RouterLink a bejelentkezési oldalhoz, a Dob() függvény meghívásával a kijelentkezési logikához.*

***Stíluskészítés:***

* *A megjelenítést a scoped stílusokkal definiálják.*
* *A stílusok a var(--main-background), var(--sec-background), var(--sec-text-color), var(--border-radius) CSS változókat használják, amelyeket valószínűleg a globális stíluslapban definiáltak.*
* *A menü alapértelmezetten el van rejtve, és a MenuStore.isProfileMenuOpened tulajdonság értéke alapján nyílik ki.*
* *A menüpontok szövege alapértelmezetten rejtett, és a MenuStore.isProfileMenuOpened tulajdonság értéke alapján válik láthatóvá.*

***Használt komponensek:***

* *RouterLink (vue-router): Router link a navigációhoz.*
* *RouterView (vue-router): A router nézet megjelenítési helye.*

***Kommunikáció:***

* *A komponens a MenuStore pincére állapotváltozásaira reagál a Vue 3 reaktív tulajdonságok segítségével.*

## 

## *Az alkalmazás fejléce*

***Fájl neve:*** *top-bar.vue (feltételezve)*

***Használt könyvtárak:***

* *Vue.js*

***Komponens állapota:***

* *A komponens az isDarkTheme reaktív változóval tárolja a sötét téma beállítását.*

***Függvények:***

* *onMounted(): A komponens létrehozásakor lefutó életciklusfüggvény.*
  + *Beállítja az isDarkTheme értékét a localStorage alapján.*
  + *A toggleTheme() függvényt meghívja a téma alkalmazásához.*
* *toggleThemeValue(): A témabeállítást váltja.*
  + *Frissíti az isDarkTheme értékét.*
  + *Beállítja a localStorage értékét az új témához.*
  + *A toggleTheme() függvényt meghívja a frissített téma alkalmazásához.*
* *toggleTheme(): A téma megjelenítését frissíti a CSS változók módosításával a isDarkTheme érték alapján.*

***Sablon:***

* *A sablon az alkalmazás fejlécét jeleníti meg.*
* *Tartalmazza a következőket:*
  + *Menügomb (ha a MenuStore.isMainMenuDisplayed igaz) a főmenü megjelenítéséhez/elrejtéséhez (a MenuStore pincére vezérlésével).*
  + *Az alkalmazás neve ("Boola").*
  + *Profilmenü a témaváltó gombbal és a profilmenü megjelenítéséhez (profileMenu komponens).*
* *A fejléc háttérszíne a var(--sec-background) CSS változóval van definiálva.*

***Stíluskészítés:***

* *A megjelenítést a scoped stílusokkal definiálják.*
* *A stílusok több CSS változót használnak, amelyeket valószínűleg a globális stíluslapban definiáltak:*
  + *var(--main-background)*
  + *var(--sec-background)*
  + *var(--main-text-color)*
  + *var(--sec-text-color)*
  + *var(--menu-width) (feltételezhetően a főmenü szélességét definiálja)*
  + *var(--menu-def-width) (feltételezhetően a főmenü alapértelmezett szélességét definiálja)*
* *A sablon a profileMenu komponenst használja a profilmenü megjelenítéséhez.*
* *A fejléc megjelenése reagál a képernyőméretre:*
  + *Nagyobb képernyőkön (1200px vagy nagyobb):*
    - *A Menü gomb a bal oldalon jelenik meg.*
    - *Az alkalmazás neve középre kerül a főmenü szélessége alapján.*
  + *Közepes képernyőkön (577px és 1199px között):*
    - *Az alkalmazás neve középre kerül a var(--menu-def-width) értéke alapján.*

***Kommunikáció:***

* *A komponens a MenuStore pincére isMainMenuDisplayed tulajdonságának változásaira reagál.*

***Megjegyzések:***

* *A kód feltételezi, hogy a MenuStore pincért definiálták a Vuex tárolóban.*
* *A localStorage használatával menti el a felhasználói beállítást a téma megjelenítéséhez.*

1. *REST API*
2. /

get:

summary: Welcome to Boola!

description: Returns a greeting message for the Boola financial app.

responses:

'200':

description: OK

content:

text/plain:

schema:

type: string

example: Hello World! This is the home of Boola, the new financial app!

1. /tst

paths:

/tst:

get:

summary: Get Database Status

description: Retrieves the database status for authorized users.

security:

- boola-auth

responses:

'200':

description: OK

content:

application/json:

schema:

type: boolean

'503':

description: Service Unavailable

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message

components:

securitySchemes:

boolaAuth:

type: http

scheme: bearer

bearerFormat: JWT

1. /login

post:

summary: Login User

description: Allows users to authenticate and obtain access and refresh tokens.

requestBody:

required: true

content:

application/json:

schema:

Account

responses:

'200':

description: OK

content:

application/json:

schema:

type: object

properties:

access:

type: string

description: Access token

refresh:

type: string

description: Refresh token

'401':

description: Unauthorized

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message (e.g., The token was incorrect)

'503':

description: Service Unavailable

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message

1. /register

post:

summary: Register User

description: Allows users to create a new account.

requestBody:

required: true

content:

application/json:

schema:

Account

responses:

'201':

description: Created

'503':

description: Service Unavailable

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message

1. api/account/{email}

get :

summary: Get Account Information

description: Retrieves account information for the specified email address (requires authorization).

security:

- boolaAuth

parameters:

- name: email

in: path

required: true

description: User's email address

schema:

type: string

responses:

'200':

description: OK

content:

application/json:

schema:

'401':

description: Unauthorized

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message (e.g., The token was incorrect)

'503':

description: Service Unavailable

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message

put:

summary: Update Account Information

description: Allows users to update their account information (requires authorization).

security:

- boolaAuth: []

parameters:

- name: email

in: path

required: true

description: User's email address

schema:

type: string

requestBody:

required: true

content:

application/json:

schema:

Account

responses:

'200':

description: OK

'401':

description: Unauthorized

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message (e.g., The token was incorrect)

'503':

description: Service Unavailable

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message

delete:

summary: Delete Account

description: Deletes an account associated with the provided email address.

parameters:

- name: email

in: path

required: true

schema:

type: string

responses:

'204':

description: No Content

'400':

description: Bad Request - The request could not be understood or was invalid.

content:

application/json:

schema:

type: object

properties:

message:

type: string

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

1. refresh

post:

summary: Refresh Access Token

description: Generates a new access token using a refresh token (not implemented in this example).

security:

- boola-refresh: [] # Reference security scheme

requestBody:

required: true

content:

application/json:

schema:

type: object

properties:

email:

type: string

description: User email address

responses:

'200':

description: OK

content:

application/json:

schema:

type: object

properties:

access:

type: string

description: New access token

'401':

description: Unauthorized

content:

application/json:

schema:

type: object

properties:

message:

type: string

description: Error message (e.g., "The token was incorrect")

1. api/currency

get:

summary: Get Currency by Code

description: Retrieves details of a specific currency based on its code.

parameters:

- name: code

in: path

required: true

schema:

type: string

responses:

'200':

description: OK

content:

application/json:

schema:

Currency

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

1. api/currency/{code}

get:

summary: Get All Currencies

description: Retrieves a list of all available currencies.

responses:

'200':

description: OK

content:

application/json:

schema:

Currency

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

1. api/category/

get:

summary: Get All Categories

description: Retrieves a list of all available categories.

responses:

'200':

description: OK

content:

application/json:

schema:

type: array

items:

type: object

Currency

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

1. api/category/{id}

get:

summary: Get Category by ID

description: Retrieves details of a specific category based on its numeric ID.

parameters:

- name: id

in: path

required: true

schema:

type: integer

responses:

'200':

description: OK

content:

application/json:

schema:

Category

'400':

description: Bad Request - The request is invalid (e.g., missing ID).

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

1. api/profile

get:

summary: Get User Profiles

description: Retrieves all profiles associated with the authenticated user.

security:

- boolaAuth

responses:

'200':

description: OK

content:

application/json:

schema:

type: array

items:

type: object

Profile

'401':

description: Unauthorized - The request requires user authentication but valid credentials were not provided.

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

post:

summary: Create User Profile

description: Creates a new user profile associated with the authenticated user.

requestBody:

required: true

content:

application/json:

schema:

Profile

type: object

responses:

'201':

description: Created

headers:

Location:

description: URL of the created profile

schema:

type: string

'403':

description: Forbidden - The user is not authorized to create a profile for another user's email.

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

1. api/profile/{id}

get:

summary: Get User Profile by ID

description: Retrieves a specific user profile based on its ID.

parameters:

- name: id

in: path

required: true

schema:

type: string

format: uuid

responses:

'200':

description: OK

content:

application/json:

schema:

Profile

'401':

description: Unauthorized - The request requires user authentication but valid credentials were not provided.

'403':

description: Forbidden - The user is not authorized to access the requested profile.

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

put:

summary: Update User Profile

description: Updates an existing user profile based on its ID.

parameters:

- name: id

in: path

required: true

schema:

type: string

format: uuid

requestBody:

required: true

content:

application/json:

schema:

# Update this section based on your Profile data structure

# (e.g., add properties for profile details)

type: object

properties:

id: # Assuming ID is kept unchanged during update (optional)

type: string

format: uuid

description: Unique identifier of the profile

accountEmail: # Assuming email cannot be changed through update (optional)

type: string

description: Email address associated with the profile owner (read-only)

expenseListId:

type: string

description: Unique identifier of the expense list associated with the profile (optional)

# Add other profile properties here

responses:

'200':

description: OK - The profile was updated successfully.

'400':

description: Bad Request - The request body is invalid, missing required fields, or the ID format is incorrect.

'403':

description: Forbidden - The user is not authorized to update the requested profile.

'404':

description: Not Found - The specified profile ID was not found.

'500':

description: Internal Server Error - The server encountered an error while processing the request.

'503':

description: Service Unavailable - The server is currently unable to handle the request due to a temporary overload or maintenance of the service.

delete:

summary: Delete a user profile

description: Deletes a user profile along with its associated expense list.

tags:

- Profile

parameters:

- name: id

in: path

required: true

description: The UUID of the profile to delete

schema:

type: string

format: uuid

responses:

'204':

description: No Content - The profile was deleted successfully.

'400':

description: Bad Request - There was a problem with the request.

'403':

description: Forbidden - The user does not have permission to delete the profile.

'503':

description: Service Unavailable - The DataController is not available.

1. api/expenselist

get:

summary: Get all expense lists for the authenticated user

description: Retrieves all expense lists associated with the user's email address claimed in the JWT.

responses:

'200':

description: OK - A list of expense list objects.

content:

application/json:

schema:

type: array

items:

ExpenseList

'400':

description: Bad Request - The JWT is missing the required "email" claim or the user cannot be found.

'503':

description: Service Unavailable - The DataController is not available.

1. api/expenselist/{id}

get:

summary: Get ExpenseList by ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

200:

description: Successful retrieval of ExpenseList

content:

application/json:

schema:

type: object

400:

description: Bad Request - Missing ID parameter

403:

description: Forbidden - User is not the owner of the Expense List

503:

description: Service Unavailable - DataController not available

put:

summary: Update ExpenseList

parameters:

- name: id

in: path

required: true

schema:

type: string

requestBody:

required: true

content:

application/json:

schema:

ExpenseList

responses:

200:

description: ExpenseList updated successfully

403:

description: Forbidden - User is not the owner of the ExpenseList

404:

description: Not Found - ExpenseList with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

delete:

summary: Delete ExpenseList

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

204:

description: Expense List deleted successfully (No Content)

403:

description: Forbidden - User is not the owner of the Expense List

404:

description: Not Found - Expense List with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

1. api/partner

get:

summary: Get All Partners

responses:

200:

description: List of all partners

content:

application/json:

schema:

type: array

items:

Partner

503:

description: Service Unavailable - DataController not available

post:

summary: Create Partner

requestBody:

required: true

content:

application/json:

schema:

Partner

responses:

201:

description: Partner created successfully (Created)

400:

description: Bad Request - Invalid partner data in request body

503:

description: Service Unavailable - DataController not available

1. api/partner/{id}

get:

summary: Get Partner by ID

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

200:

description: Partner details for the provided ID

content:

application/json:

schema:

Partner

404:

description: Not Found - Partner with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

put:

summary: Update Partner

parameters:

- name: id

in: path

required: true

schema:

type: string

requestBody:

required: true

content:

application/json:

schema:

Partner

responses:

200:

description: Partner updated successfully

404:

description: Not Found - Partner with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

delete:

summary: Delete Partner

parameters:

- name: id

in: path

required: true

schema:

type: string

responses:

204:

description: Partner deleted successfully (No Content)

404:

description: Not Found - Partner with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

1. api/expense

get:

summary: Get All Expenses for an Expense List

parameters:

- name: listId

in: query

required: true

schema:

type: string

format: uuid

responses:

200:

description: List of all expenses for the specified expense list

content:

application/json:

schema:

type: array

items:

Expense

400:

description: Bad Request - Invalid listId format

403:

description: Forbidden - User is not the owner of the Expense List

503:

description: Service Unavailable - DataController not available

post:

summary: Create Expense for Expense List

requestBody:

required: true

content:

application/json:

schema:

Expense

responses:

201:

description: Expense created successfully

headers:

Location:

description: URL of the created expense

schema:

type: string

format: uri

400:

description: Bad Request - Insufficient funds in Expense List or invalid expense data

403:

description: Forbidden - User is not the owner of the Expense List

503:

description: Service Unavailable - DataController not available

1. api/expense/{id}

get:

summary: Get Expense by ID

parameters:

- name: id

in: path

required: true

schema:

type: string

format: uuid

responses:

200:

description: Expense details retrieved successfully

content:

application/json:

schema:

Expense

400:

description: Bad Request - Invalid ID format

403:

description: Forbidden - User is not the owner of the Expense List containing the Expense

404:

description: Not Found - Expense with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

put:

summary: Update Expense

parameters:

- name: id

in: path

required: true

schema:

type: string

format: uuid

requestBody:

required: true

content:

application/json:

schema:

Expense

responses:

200:

description: Expense updated successfully

400:

description: Bad Request - Invalid ID format or invalid expense data in request body

403:

description: Forbidden - User is not the owner of the Expense List containing the Expense

404:

description: Not Found - Expense with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

delete:

summary: Delete Expense

parameters:

- name: id

in: path

required: true

schema:

type: string

format: uuid

responses:

204:

description: Expense deleted successfully (No Content)

400:

description: Bad Request - Invalid ID format

403:

description: Forbidden - User is not the owner of the Expense List containing the Expense

404:

description: Not Found - Expense with the provided ID does not exist

503:

description: Service Unavailable - DataController not available

1. api/language

get:

summary: Get All Languages

responses:

200:

description: List of all supported languages

content:

application/json:

type: array

items:

type: string

503:

description: Service Unavailable - DataController not available

1. api/language/{code}

get:

summary: Get Language by Code

parameters:

- name: code

in: path

required: true

schema:

type: string

responses:

200:

description: Language details retrieved successfully

content:

application/json:

Language

404:

description: Not Found - Language with the provided code does not exist

503:

description: Service Unavailable - DataController not available