Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Elektronikai Technológia Tanszék

Borbély Katalin

GYEREKBANK WEBALKALMAZÁS FEJLESZTÉSE JAVAEE PLATFORMON ANGULARJS FRONTENDDEL

KONZULENS

Dr. Villányi Balázs

BUDAPEST, 2017

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Borbély Katalin**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2017. 12. 11.

...……

Borbély Katalin

Összefoglaló

Szakdolgozatom témája egy tanító jellegű Gyerekbank webalkalmazás megvalósítása. A dolgozat első részében röviden bemutatok létező gyerekbank megoldásokat Magyarországról és külföldről, továbbá ismertetem a magyarországi bankszámla létesítésére vonatkozó legelterjedtebb banki előírásokat és szerződési feltételeket. Ezek után rátérek saját gyerekbank alkalmazásom követelményeire és bemutatom, hogyan kapcsolódik a megoldás a bank architektúrájához.

A fejlesztéshez felhasznált eszközök és technológiák bemutatása a dolgozat 3. fejezetében található. A frontend fejlesztéséhez az Angular JS és a Bootstrap keretrendszert használtam valamint a JavaScript programozási nyelvet választottam. A backend megvalósítása JAVA EE platformon történt, az általam használt adatbáziskezelő pedig a MySQL lett. A webes alkalmazások és szolgáltatások kiszolgálásához az Apache Tomcat Java Enterprise Edition részeként választásom a TomEE webszerverre esett. A forráskód menedzselését és a fordítást a Maven keretrendszer biztosítja.

A 4. fejezet részletesen bemutatja a szülő és a gyerek felhasználók által elérhető funkciókat és a rendszer mögötti adatbázis struktúrát. Az alkalmazás által támogatott hitelfelvétel, megtakarítások, egyenleglekérdezés és valamennyi kapcsolódó tevékenység támogatott folyamatát is bemutatásra került.

A megvalósítás részleteit az implementálás fejezet tartalmazza. Ismertetem a frontend webosztályait, a REST hívásokat, az üzleti logika és az entitások osztályait valamint az adatbázis műveletek DAO osztályait és az adatátviteli objektumokat (DTO).

Munkámat teszteléssel zártam, ahol az alkalmazás működését ellenőriztem valamennyi folyamat végig vitelével. A dolgozatban a hitelfelvétel, elbírálás és betétfelbontás folyamatok tesztelését mutatom be részletesen.

Abstract

The subject of my thesis is the implementation of a banking application for children.

In the first part of my work, I briefly present existing child bank solutions from Hungary and abroad, as well as the current banking regulations and contract terms for establishing a bank account in Hungary. After that, the requirement of my own child bank application are presented. The connection points of my application into the environment of a bank are shown from the point of view of the architecture.

Presentation of the tools and technologies used for development can be found in Chapter 3 of the Thesis. For the frontend I used Angular JS and Bootstrap framework and I chose the JavaScript programming language. The backend was implemented on JAVA EE platform, and the database manager MySQL was applied. To handle web applications and services my choice fell on the TomEE webserver which is part of the Apache Tomcat Java Enterprise Edition. Source code management and translation are provided by the Maven Framework.

Chapter 4 details the functions available to parents and children and the database structure behind the system. The supported process of taking a loan, creating deposits, checking account balance and all related activities supported by the application was also presented.

Details of implementation are included in section 4.2. I describe the frontend web classes, REST calls, business logic and entity classes as well as DAO classes of database operations and data transfer objects (DTOs).

I closed my work with testing, where I checked the functionality of the application by running all the processes. The testing of the processes taking a loan, credit assessment and termination of a deposit are presented in details at the end of my Thesis.

1 Bevezetés

Mai világunkban fiatal korunktól kezdve egyre többet kerülünk kapcsolatba a bankkal és az általa nyújtott szolgáltatásokkal. Ezen szolgáltatásokat azonban legtöbbször csak éles helyzetben, igénybevételükkor ismerjük meg. Sokszor hallani afelől, hogy a mai szigorú feltételek mellett kiadott hiteleket felvevő személyek fizetésképtelenné válnak és utcára kerülnek, akár egy rosszul megválasztott hitel, akár felesleges pénzkiadások miatt.

Fontos számomra, hogy a gyerekek minél előbb megismerkedjenek a pénz és a bank fogalmával. A mostani tendenciákat nézve, ahogyan máshol a fejlett országokban úgy itthon Magyarországon is idővel egyre jobban csökken majd a készpénz forgalom, ennek következtében nem elég az, ha egy gyerek csak azt ismeri milyen, amikor van egy malacperselye otthon amiben gyűjtögethet, hanem azt is ismernie kell, hogy felnőttként miként kell majd a pénzzel bánnia elkerülve az eladósodást, fizetésképtelenséget és a rossz hiteleket.

Szakdolgozatom keretein belül szeretnék egy olyan alkalmazást létrehozni, melynek célja a legfontosabb banki szolgáltatások megismertetése a gyerekekkel. Ezek az alapvető szolgáltatások a hitelfelvétel, hiteltörlesztés, befektetés létrehozás illetve annak felbontása. Ennek segítségével a gyerekek szüleik támogatása mellett válhatnak tudatos pénzfelhasználókká már kiskoruktól kezdve.

1.1 A szakdolgozat felépítése

Napjainkban teret nyerő gyerekbanki megoldások ismertetése után bemutatom az általam fejlesztett webalkalmazást és összehasonlítom az élesben működő megoldásokkal. Majd pedig arról lesz szó, hogy hogyan integrálható be egy átlagos banki szoftverarchitektúrába. Az ezt követő fejezetben kiválasztom és bemutatom a webalkalmazásom fejlesztéséhez leginkább megfelelő technológiákat, értékelem és összehasonlítom őket. Ezután a tervezés és az implementálás legfontosabb részeinek demonstrálása következik. A dolgozatot a tesztelés eredményének és a továbbfejlesztési lehetőségeknek az ismertetése zárja.

2 Napjaink gyerekbanki megoldásai

Ma Magyarországon a bankok Általános Szerződési Feltételében (ÁSZF) leírtak szerint bankszámlát nyithat minden olyan személy, aki rendelkezik azonosításra alkalmas okmánnyal. Azonosításra alkalmas okmány lehet az útlevél is, melyet 0 éves kortól lehet igényelni. A 14. életévet be nem töltött gyerek csak törvényes képviselőjével tud számlát nyitni, számlája fölött a törvényes képviselő rendelkezik. A gyermek bankkártya igényelhetőségéig a számláján található pénzhez csak a törvényes képviselőjével juthat hozzá. A bankkártya igényléséhez szükséges minimális életkor bankonként változó, általában 6-7 év, de akár 14 is lehet, ahogyan az Erste Banknál is.

Annak ellenére, hogy már ilyen fiatalon is lehet felügyelet mellett bankszámlája egy embernek az interneten rákeresve a „gyerekbank", „banki alkalmazás gyerekeknek" és más hasonló kifejezésekre nem igen találtam olyan magyar nyelven is elérhető alkalmazást, mely a gyerekeket megtanítaná a pénzhasználatra és/vagy a banki szolgáltatásokat megmutatná neki, melyet saját maga is kipróbálhatna. A legtöbb találati eredmény a gyerekeknek szánt számlák vagy megtakarítások jelentették. Találtam egy-két olyan tanító jellegű játékot, mely a banki ügyintézéssel kapcsolatos. Ezen játékokon keresztül viszont a gyerekek nem ismerkednek meg a legfontosabb dolgokkal, amit tudniuk kell, és főként nem saját tapasztalataik által úgymond majdnem éles körülmények között.

Többféle számla is nyitható gyermekünknek, ezek közül néhány direkt megtakarítási céllal működik, melyre a szülőknek havonta utalniuk kell egy minimális összeget a gyerek számlájára és az ott kamatozik egészen 18 éves koráig. Ezen számlák szakdolgozatom célját tekintve nem relevánsak. Térjünk ki azon bankszámlákra és a hozzájuk tartozó szolgáltatásokra, melyeknél a gyerekek 18 éves koruk alatt is hozzáférhetnek valamilyen módon a pénzükhöz.

A gyerekeknek és fiataloknak szánt számlacsomagok sokkal kedvezőbb díjakért vehetőek igénybe, mint a felnőtt számlák, ezzel is csábítva arra a szülőket, hogy nyissanak gyereküknek bankszámlát és ott tárolják pénzüket. A gyerekeket takarékosságra nevelve egyes bankok matricacsomaggal jutalmazták őket, ha pénz érkezett a számlájukra vagy egy a bank által meghatározott összegű megtakarítást elértek. Ilyen volt a CIB Bank Szia Szimba Megtakarítási Számlához tartozó Szia

Szimba Program, melyet 4 és 14 év közötti egyének vehettek igénybe - 2013 január 2-tól megszűnt. Amennyiben a számlatulajdonos az előző havi egyenlegéhez képest 1000 Ft-tal növelte az egyenlegét az adott hónap végéig egy darab matricát kapott. A matricákat ajándékokra lehetett váltani – minden ajándékhoz fel volt tüntetve, hogy mennyi matricát kell összegyűjteni érte. Bankkártya csak 6 éves kor fölött volt igényelhető a bankszámlához.

Az AXA Bank Manószámlája 0 és 14 éves kor közötti gyerekek számára volt elérhető, ahol a számlának a gyerek a tulajdonosa, de a szülő rendelkezik a számlán tartott összeg felett. Ingyenes számlanyitás, utalási- és számlafenntartási költség volt rá jellemző. A számlához napi kamatozás tartozott lekötés nélkül. Mamabank által a szülő a banki kamaton felül plusz kamatot adhatott gyermekének a számlán tartott pénzére. A számlához nem lehetett igényelni bankkártyát, így a gyerekek csak szülői segítséggel férhettek hozzá a pénzükhöz. Az AXA Bank Magyarországról való 2016-os kivonulásakor ügyfeleinek OTP Bankba történő átvezetése után ez a számlatípus már nem igényelhető.

Külföldön kedvezőbb a helyzet, több olyan banki megoldást találhatunk, mely a gyerekeket érdemben segíti. Az ASB bank kiskorú ügyfelei speciális gyerek-bankszámlát nyithatnak maguknak[1]. A számlán elhelyezett pénz után napi kamatot kapnak, lekötéseket végezhetnek és nyomon követhetik egyenlegüket a netbank alkalmazás használatával. Pénz felvétele, illetve átutalási megbízások valamint bankkártyás vásárlás 13 éves kor alatt szülői engedéllyel, 13-18 éves kor között pedig akár már önállóan is lehetséges. A számlavezetés és a legtöbb szolgáltatás ingyenes vagy jóval kedvezőbb, mint egy hagyományos lakossági folyószámla esetében. Mindez a bank által rendelkezésre bocsátott célzottan gyerekeknek szóló játékos oktatóvideókkal együtt remek lehetőség, hogy gyermekünk elsajátítsa az alapvető ismereteket. A Standard bank 16 évnél fiatalabb gyermek ügyfelei is fentiekhez hasonlóan kedvező szolgáltatás csomagot kaphatnak és a különböző tevékenységekben (zsebpénz kérése a szülőtől, egyenleg lekérdezése, új játékra gyűjtés, engedély kérése a szülőtől költéshez, stb.) dedikált kisállat figurák segítenek nekik [2]. Ezen felül a szülők elérendő célokat – küldetéseket is definiálhatnak gyermekük számára, melyek teljesítéséért ajándék jár, illetve elkészített házi feladatért is járhat jutalom pénz. Az ANZ bank 12 évnél idősebb ügyfelei saját, 12 éves kor alatt pedig a szülő számlájához kötött, de a fentiekhez hasonló kedvezményes számlát nyithatnak[3]. Valamennyi

gyermekbank számla fontos közös vonása, hogy a szülők bármikor megtekinthetik és felügyelhetik a gyerekek pénzügyeit.

Oktatási célra kifejlesztett szoftverek is léteznek, melyek virtuális bankok létrehozására és működtetésére alkalmasak[4][5]. Ezen bankok ügyfelei lehetnek a gyerekek, a bankárok és pénztárosok szerepét pedig iskolai oktatásban tanárok, otthon pedig szülők tölthetik be. Mivel ezek hátterében nem állnak valós bankok és pénzmozgások ez a megközelítés teljesen más az általam tervezett megoldáshoz képest így a dolgozat keretein belül ezekkel részletesebben nem foglalkozom.

2.1 Gyerekbank alkalmazás bemutatása

A Gyerekbank egy tanító jellegű alkalmazás, mely saját való életbeli tapasztalatok megszerzését segíti elő. Használói ennek köszönhetően később tudatos pénzfelhasználók lesznek. Az alkalmazás segít a gyerekeket takarékosságra ösztönözni és nevelni. Alkalmazásom az online bank egy leegyszerűsített változata, ahol a gyerekek megtanulhatják az alapvető pénzügyi folyamatokat, mint hiteligénylés, hiteltörlesztés és befektetés. Számukra a szülő lesz a bank, tőlük tudnak hitelt igényelni, nekik törlesztenek, hozzájuk fektethetnek be.

A szülők utalhatnak a gyereknek, hitel és befektetési tételeket írhatnak ki nekik, látják költéseiket, adataikat, befektetéseiket, hiteligényléseiket – melyet nekik kell elbírálni. Teljesen egyedi hitel konstrukciókat és befektetési tételeket is adhatnak meg.

A gyerekek itt tudják nyomon követni költéseiket, a szüleik által kiírt hiteleket felvehetik, azt törleszthetik – mint egy banknál, illetve a heti törlesztő részlet automatikusan levonásra kerül, de tudnak elő- és végtörleszteni bizonyos díj ellenében. Befektethetnek szüleikhez, a befektetéseik felbonthatóak, de ezen esetben csak akkor kerül jóváírásra a kamat, ha a szülők által előírt minimális befektetetési időt elérték a befektetés felbontása előtt.

A Gyerekbank alkalmazás a bankok online rendszere mellett és nélkül is képes működni. Nem célja a gyerekek számára az internetbank kiváltása, ahogyan nem is lehet az kevés funkcióját tekintve. Viszont mindenképp szoros kapcsolatban áll a bankkal. A tranzakciókat a gyerekek és a szülők között a bank végzi, ahogyan az egyenlegüket is ő tartja nyílván. Az alkalmazás kiajánlott szolgáltatáson keresztül értesíti a bankot teendőjéről és kéri el a neki szükséges adatokat. A Gyerekbanknak szüksége van a bank

által kiajánlott szolgáltatásokra a következő műveletek elvégzéséhez: utalás megvalósítása, a szülő és a gyerek egyenlegének módosítása, a tranzakciók megjelenítése a számlatörténetekben.

A Gyerekbank használatát azon gyerekeknek ajánlom, akik már fel tudják mérni a pénz jelentőségét. Tudnak írni és olvasni, megértik a használt fogalmakat és a háttérben lezajló folyamatokat. Gondolok itt például arra, hogy mit jelent egy hiteligénylés, hogyan zajlik, pozitívan elbírált hiteligénylés után mi történik és mik a teendők. Ahhoz, hogy érdemben tudja használni és megalapozza a tudását értenie kell ezeket az alapvető pénzügyi műveleteket. Először a szülővel kell elsajátítania az elméleti hátteret, majd jöhet gyakorlás a Gyerekbank alkalmazásban.

A tranzakciók a gyerekek és a szülők között történnek meg. A bank egy összekötő szerepet játszik kettőjük között, kezeli a számlájukat, végrehajtja a tranzakciókat. A banknak a kettőjük között létrejövő befektetésekből és hiteligénylésekből nem származhat kára, mivel nem neki kerül visszafizetésre vagy nem visszafizetésre a hitel. A befektetett összegek után járó kamatot is a szülő fizeti meg gyereke számára, ami alapvető eltérés pénzintézetek elterjedt gyerekbank megoldásaihoz képest.

2.1.1 Összehasonlítás

Az általam fejlesztett alkalmazás nagyban eltér a mai Magyarországon is elérhető gyerekbanki megoldásoktól, melyek általában egyszerű bankszámlák gyerekeknek mindenféle tanító jellegű modul nélkül; játékok melyeken keresztül láthatják a banki szolgáltatásokat nagyvonalakban, de saját és legtöbb esetben még virtuális pénzzel sem gazdálkodhatnak benne.

A 2.1.1-es táblázat bemutatja, hogy miben hasonlít, és miben különbözik az általam tervezett alkalmazás a mai itthoni gyerekbanki megoldásoktól.

Az általam fejlesztett Gyerekbank alkalmazás	**Napjaink itthoni gyerekbanki megoldásai**
Szoros kapcsolatban áll a bankkal, mégis egy külön alkalmazás.	Maga a bank valósítja meg a gyerekbanki megoldásokat a gyerekeknek szánt számlacsomagjaikon keresztül.
A tranzakciók a szülő és a gyerek között	A tranzakciók a gyerek és a bank között

zajlanak, a bank csak a híd közöttük, mely összeköti őket.	zajlanak.
Lekérdezhető a gyerek számlatörténete a gyerek és a szülő oldalon is.	Bank- és korfüggő a számlatörténet lekérdezhetőségének minimum életkora.
A gyerek igényelhet hitelt a szülőtől.	A gyerek nem igényelhet hitelt 18 éves kora alatt a banktól.
Egyedi hitelkonstrukciókat írhat ki a szülő a gyereke számára.	A gyerek a bank által előre meghatározott standard hitelei közül válogathat a 18. életévének betöltése után.
A szülő egyedi befektetési tételeket adhat meg a gyereke számára.	A gyerek vagy nem jogosult lekötés létrehozására vagy nem térhet el a standard banki lekötésektől.
A szülő egyszeri és rendszeresen ismétlődő utalásokat indíthat gyereke számára.	Ez a funkció itt is teljes egészében elérhető.
A gyerek kis összegekre is (csak) felvehet hitelt.	-
A szülő elbírálhatja a gyerek által igényelt hiteleket. A folyósítás azonnal kezdeményezésre kerül.	-
A gyerek elő- és végtörlesztheti felvett hiteleit.	-
Lekérdezhetőek a gyerek által felvett hitelek.	-
A gyerek kis összegű befektetéseket is indíthat.	Amennyiben lehetőség van befektetés indítására úgy ez is elérhető funkció. Viszont kis összegű befektetések indítása a normál kamatok mellett nem éri meg.
Mind a szülő, mid a gyerek láthatja a gyerek által aktuálisan lekötött	Amennyiben lehetőség van lekötés indítására, úgy ez a funkció is azonos.

összegeket.	
Az internetbank számos funkciója nincs megvalósítva az alkalmazásomban. Az alkalmazás nem zárja ki a bank online felületének használatát.	Internetbank

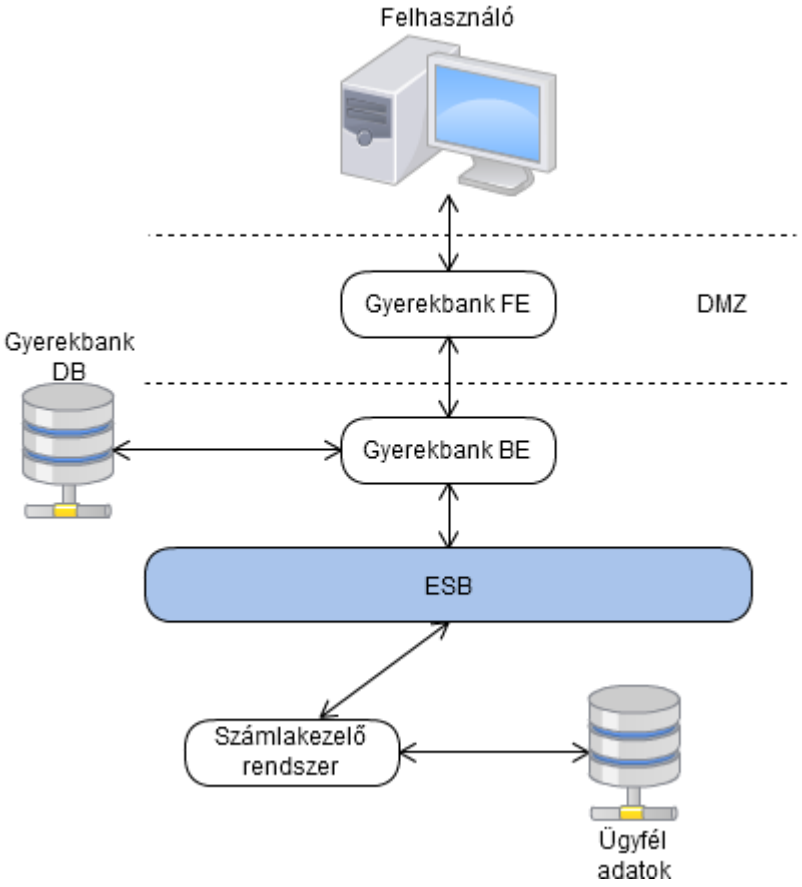
2.1.1 táblázat: Gyerekbanki megoldások összehasonlítása

A külföldi gyerekbanki megoldásoktól is eltér az általam fejlesztett alkalmazás, bár kevésbé, mint a Magyarországi gyerekbanki megoldások. A külföldi megoldások, úgy ahogyan a gyerekbank is szeretnék jobban bevonni a gyerekeket a banki világba, játszva tanítják őket, úgy, hogy közben a saját pénzükkel gazdálkodnak.

2.2 Illeszkedés egy bank szoftverarchitektúrájába

A megvalósított alkalmazás könnyen tud illeszkedni egy átlagos bank szoftverarchitektúrájába. Az alkalmazás frontend része REST hívásokon keresztül kommunikál a backenddel. A backend pedig a saját adatbázisával illetve a kiajánlott szolgáltatásokon keresztül a bank ESB (Enterprice Service Bus) interfészével. Amennyiben a banknak nincs ESB rétege, úgy közvetlenül a számlakezelő- és ügyféladat kezelő rendszerrel kerül összeköttetésbe. A gyerekbank alkalmazás frontendje a demilitarizált zónában (DMZ) helyezkedik el. A DMZ egy biztonsági réteget valósít meg megakadályozva azt, hogy a külső, nem megbízható forrásból érkező kérés eljusson a belső rendszerekhez.

A 2.2.1-es szoftver architekturális ábrán csak a Gyerekbank alkalmazás által használt elemeket tűntettem fel. A bankok szoftverarchitektúrája melybe integrálódnia kell ennél sokkal bonyolultabb, bankonként eltérő is lehet. Átfogóbb és bővebb szoftver architektúra felépítést ezért nem készítettem.



2.2.1 ábra: Szoftver architekturális felépítés

3 Felhasznált eszközök és technológiák bemutatása

Ebben a fejezetben az általam használt eszközöket és technológiákat ismertetem. Az AngularJs frontend és JavaEE backend technológia adott volt, minden mást saját magam választhattam meg. Mivel egyetemi tanulmányaim alatt korábban még nem készítettem webalkalmazást, ahogyan a fent említett technológiákkal sem dolgoztam igyekeztem olyan kiegészítő technológiákat és eszközöket választani, melyek illeszkednek hozzájuk, használatuk gyorsan megtanulható és jól alkalmazhatóak banki környezetben is.

Backend fejlesztő környezet kiválasztásakor az egyik fontos szempontom az volt, hogy olyan fejlesztőkörnyezetet válasszak, mely segíti a gyors kódírást és hibadetektálást illetve nagyobb méretű projektek fejlesztésére is alkalmas.

3.1 Java Enterprise Edition

Az eddig megjelent szerver oldali Java platformok 3 „kiadása” különböző típusú szoftverek fejlesztését teszi lehetővé:

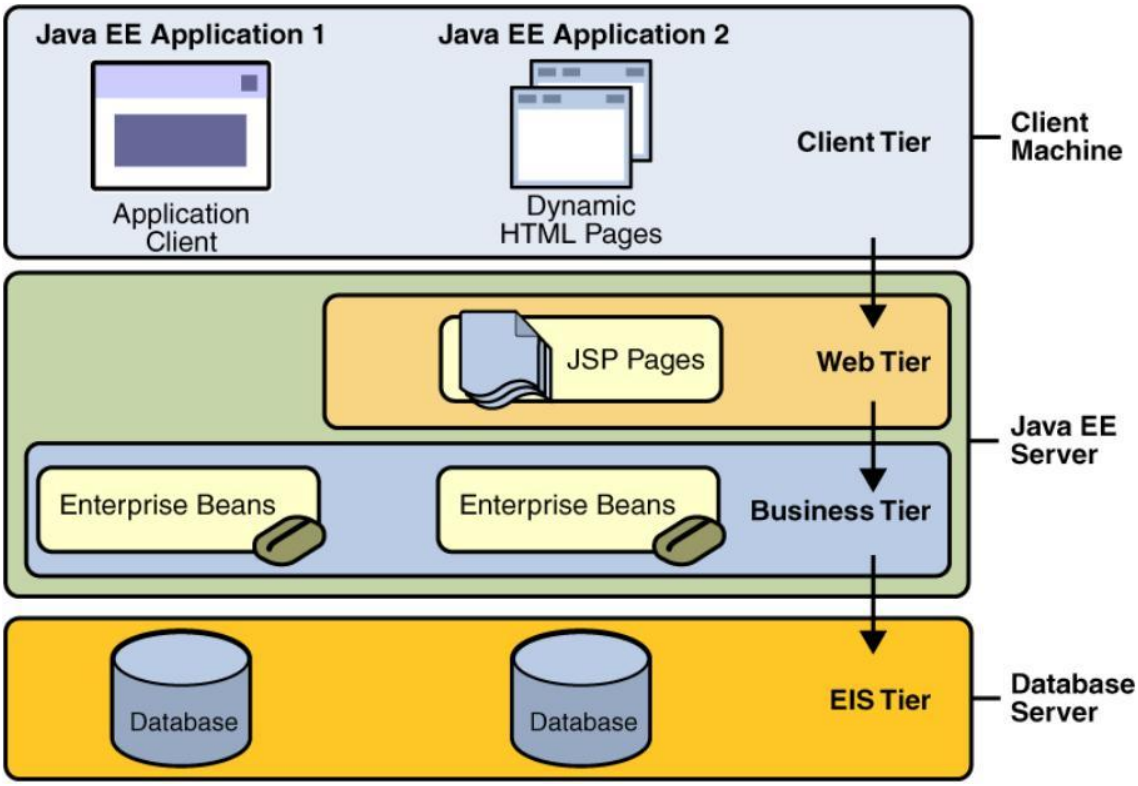
- Standard Edison (Java SE, korábban J2SE) desktop alkalmazások és appletek készítését célozza meg.
- Micro Edition (Java ME, korábban J2ME) a korlátolt erőforrásokkal jellemző mobil eszközökre való fejlesztést támogatja.
- Enterprise Edition (Java EE, korábban J2EE) hibatűrő, elosztott, sok felhasználós nagyvállalati méretű projektek fejlesztését biztosítja az alkalmazás szerverén futó többrétegű moduláris szoftverkomponensek segítségével.

A Java EE használata széles körben elterjedt, minden Java SE programkönyvtár (API) felhasználható benne az Enterprise technológiák mellett. Egy de facto szabványú specifikáció definiálja, ami tartalmazza a szoftverkomponenseket, API-kat és azok kapcsolatait [7][1].

A Java EE úgy támogatja a nagyméretű információs rendszerek fejlesztését, hogy a gyakori, együtt fellépő problémákra globális megoldást ad az ilyen rendszerek fejlesztésekor felmerülő számos hasonló követelménynek eleget téve. Ezen problémák:

perzisztencia, többszálúság, skálázhatóság, távoli elérés, biztonság, magas rendelkezésre állás, stb. problémák. A megoldás abban rejlik, hogy egy olyan keretrendszert biztosít az alkalmazásoknak, mely különböző szolgáltatásokat használ fel a problémák kiküszöbölésére, ezen szolgáltatásokat szokás middleware-szolgáltatásoknak nevezni. A Java EE több olyan technológiát is biztosít, melyek együttesen lefedik a követelményeket, mindegyikük egy-egy saját API-n keresztül érhető el [7].

A Java EE futtatókörnyezete az alkalmazásszerver mely egy réteges szoftverarchitektúrába illeszkedik, ahogyan azt a 3.1.1-es ábra mutatja. Ez általában egy háromrétegű architektúra: adatréteg, üzleti logikai réteg és kliens réteg. Webalkalmazások esetében 4. rétegként a webréteg beépül a kliens és az üzleti logikai réteg közé. [7]



3.1.1 ábra: Java EE architektúra [7]

Minden rétegnek egy saját feladata van, melyhez a közvetlenül alatta lévő réteget használja fel. Az adatréteg feladata az adatok perzisztens tárolása és az azokon való műveletvégzés támogatása. Relációs vagy perzisztens adatbázis segítségével valósítják meg ezt a réteget. Enterprice Information System (EIS) rétegnek is szokás nevezni akkor, ha hozzávesszük azon rendszereket is, melyből adatokat nyerünk ki[7].

Az adatréteg fölött van az üzleti logikai réteg, mely a megfelelő funkcionalitást biztosítja az alkalmazásnak, az adatréteget az üzleti szabályoknak eleget téve hívja meg. Az üzleti logikai réteg a webréteggel együtt az alkalmazásszerveren helyezkedik el. A webréteg a böngészőből érkező kéréseket fogadja, feldolgozza, meghívja a megfelelő üzleti logikát majd pedig választ generál. Kommunikálhat közvetlen a böngészővel, vagy közvetetten egy webszerver segítségével.

A legfelső kliens réteg a felhasználói réteget valósítja meg, biztosítva a kapcsolatot a felhasználó és az üzleti logika között. Két kliens típust különböztetünk meg: a vastag kliens - tipikusan a desktop alkalmazások illetve a vékony kliens ami a böngésző, mely webalkalmazások esetén biztosítja a letöltött oldalak megjelenítését. A gazdag webes klienseknek köszönhetően a határ a kettő között elmosódik.

3.1.1 Enterprise JavaBeans

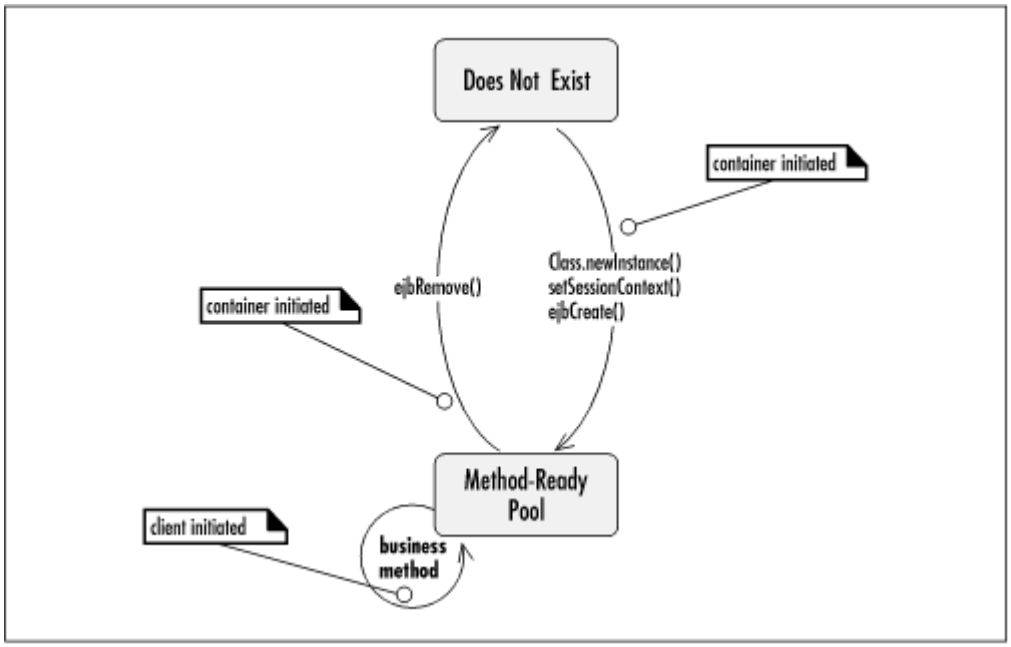
Az Enterprice JavaBeanek (EJB) a Java EE alkalmazásokban központi helyet töltenek be. Különféle szolgáltatásokat nyújtó szerver oldali komponens, mely segítségével összetett alkalmazásokat és rendszereket hozhatunk létre. Az EJB az üzleti logikát és az üzleti modelt tartalmazza. A jelenlegi EJB specifikáció 3 fajta EJB-t specifikál[7]:

1. entity bean
2. session bean
3. message-driven bean

Az entity bean az üzleti modelt, annak entitásait reprezentálja. A perzisztenciáért felelős adatbázissal tartja a kapcsolatot. Egy példánya lényegében a relációs adatbázis egy rekordjának memóriabeli cache-e. Esetemben az alkalmazás Model osztályai valósítják meg az entity beant.

A session bean-ek üzleti folyamatokat reprezentálnak. Ilyen tipikusan a banki szolgáltatások nagy része (pl. hitelkártya-kezelés, hitelügyintézés). Ezen szolgáltatásokat külön-külön metódusban valósítják meg, és az összetartozó metódusokat fogják össze egy session bean-ben. Két féle session bean létezik: állapottartó és állapotmentes. Az állapotmentes session bean azt jelenti, hogy a kliens nem számíthat arra, hogy végig ugyanazzal a beannel fog kommunikálni, így nem tarthat benne állapotot. Az állapotmentes session beanek a metódusok végrehajtásához

szükséges összes változót a metódusok paramétereiből kapják. Igen egyszerű életciklusuk a 3.1.1-es ábrán látható. Az állapotmentes session beanek inicializáló metódusának nincs bemenő paramétere. Minden bean példányt a konténer hoz létre, majd meghívásra kerülnek a serSessionContext() és ejbCreate() metódusok. Bármelyik példányhoz befuthat a kliens általi hívás. A példányokat a konténer bármikor megszüntetheti az ejbRemote() metódussal[7].



3.1.1 ábra: Állapotmentes session bean életciklusa[7]

Az állapottartó session beaneket akkor használjuk, ha szükség van arra, hogy a kliens állapotot tároljon több híváson keresztül. Nincs szükségem állapottartó session beanre, mivel az alkalmazás felhasználója megtalálható minden REST hívás fejlécében. Az adatok lekérdezéséhez kiválasztott gyereket változtathatják, ennek mindenképp paraméterben kell érkeznie. Emellett nincs olyan funkcióm, melynek szüksége lenne arra, hogy több híváson keresztül is tároljak hozzá egy állapotot.

A message-driven beanek az aszinkron üzenetváltásra szolgáltatnak kényelmes megoldást. Esetemben nincs szükség aszinkron üzenetváltásra, így nem használom ezt a fajta EJB-t.

3.1.2 Java Servlet

A szervletek olyan Java nyelven írt osztályok, melyekkel egyszerűen és hatékonyan fejleszthetünk dinamikus tartalmat generáló szerveroldali megoldásokat. Tehát a szervlet valójában egy speciális Java program, amely a szerveroldalon lehetővé

teszi HTML oldalak dinamikus létrehozását és paraméterezését különböző átviteli protokollon keresztül. A szervletek életciklusa három fázisra bontható[7]:

- inicializáció (init metódus)
- kiszolgálás (sevice metódus)
- eltávolítás (destroy metódus)

Kérés érkezésekor, ha még nincs példányosítva a servlet a webkonténer betölti a szervlet osztályt és létrehoz belőle egy példányt. A példány *init* metódusának segítségével megtörténik az inicializáció és a szükséges memória lefoglalása. Minden kérés esetén a szervlet *service* metódusa kerül meghívásra, mely két objektumot kap paraméterül: egy ServletRequest és ServletResposet. A végrehajtandó kérést a ServletRequest paraméterből kapjuk meg, míg a választ a ServletResponse paraméterbe kell átadnunk. A *service* metódusban a HTTP kérés típusától függően a kérés- és válaszobjektumok a *doGet(), doPost()* stb. metódusoknak továbbítódnak. Amennyiben egy meghatározott ideig nem érkezik kérés a szervlet felé, a konténer dönthet úgy, hogy eltávolítja a szervletet a memóriából. Előtte meghívásra kerül a *destroy* metódus, mely felszabadítja a szervlet számára allokált memóriát[7].

3.1.3 Java Server Pages

A Java Server Pages (JSP) a szervleteket kiegészítő szerver oldali technológia, mely segítségével egyszerűen lehet dinamikus tartalmakat generálni. Azért választottam a JSP-t, mert amíg a dinamikus tartalmat generáló szervletek a kimenetet Java kódba ágyazott HTML-jelölőelemekkel állítják össze, addig a JSP-k egyszerű szövegfájlként képesek dinamikus tartalmat előállítani. Kérés érkezésekor a konténer megvizsgálja, hogy a JSP vagy a hozzá tartozó szervlet e a régebbi. Amennyiben a JSP az újabb, úgy először lefordítja a hozzá tartozó szervletet, majd továbbítja felé a kérést. Ellenben ha a JSP a régebbi, akkor a kérést egyenesen a szervletnek továbbítja. Ezáltal a JSP oldalak módosításakor a hozzá tartozó szervlet is újrafordul. A JSP működésének ez egy egyik nagy előnye, viszont a módosított JSP-hez érkező első kérés válaszideje hosszabb lesz, mint az átlag, mivel előtte újra kell fordítani a szervletet is. A válaszidő tartás érdekében ezt az esetet le kell kezelni[7].

3.2 AngularJS

Az AngularJS a Google által kifejlesztett népszerű nyílt forráskódú keretrendszer. A nyelv az általános JavaScript technológiáját terjeszti ki. segítségével a Model-View-Controller architektúrának kliens oldali szintjét tudja megvalósítani kibővített funkcionalitással[9].

A JavaScript hagyományosan egy egyszerű programozási szkript nyelvnek készült, amely használatánál nem kell az alkalmazás strukturálásáról külön gondoskodnunk. A weboldalak mögött futó nyelvben gyorsan létrehozhatóak a szükséges kliens oldali programok. A böngészőkben futtatott programok bonyolultságának növekedésével és az egyre komplexebb kódok megjelenésével azonban új megközelítésre van szükség, amit az AngularJS tölthet be. Ez olyan aspektusból közelíti meg a webes programozást, amely lehetőséget teremt átlátható és jó megjelenésű kódot készítésére a Model – View – Controller (MVC) architektúra mintáját követve.

Az AngularJS gyakorlatilag egy JavaScript könyvtár, amelyet a használathoz be kell importálnunk a statikus webes oldalainkba, ugyanakkor keretrendszerként is tekinthetünk rá, hiszen sokkal több támogatást nyújt, mint pusztán a plusz funkcionalitás.

Az MVC architektúrát követve a Modell az alkalmazásunk üzleti logikáját és az adatait tartalmazza, a Nézet az a komponens, amit a felhasználóink látnak, a Kontroller pedig az előző kettő közötti interakciót valósítja meg.

Az AngularJS széles körű támogatást nyújt a programozó számára, például biztosítja az Ajax (Asynchronous JavaScript and XML) alapú kommunikációt a szerver oldallal. A nézetek megjelenítésének menedzselését különböző template-ek használatával segíti, és az egyes HTML DOM (Document Object Model) elemeknek is könnyen kezelhetővé válnak.

Az egyik legnagyobb előnye az AngularJS keretrendszerének, az a HTML szintaktika kiterjesztése saját elemekkel, az úgynevezett direktívákkal. A direktívák tartják fent a kapcsolatot a HTML DOM-mal, és közben olyan tulajdonságokkal látják el azt, hogy a statikus oldal könnyen tudja kezelni az alkalmazáshoz tartozó logikát.

3.3 Apache Maven

Az Apache Maven egy build menedzsment eszköz. Kulcsfontosságú szerepet játszik a projektkészítési feladatokban, úgy mint a csomagolás, artifact menedzsment és összeállítás. A Maven szabálykészletének megfogalmazásához az Extenseble Markup Language (XML)-t használja, mely elősegíti a következetességet a rugalmasság fenntartása mellett[8].

A Maven elsődleges céljai a következők:

- olyan projektmodell létrehozása, mely újrafelhasználható és könnyen karbantartható.
- plug-inek vagy eszközök, melyek kölcsönhatásban állnak a deklaratív modellel.

A Maven alapvető egysége a Project Object Model (POM). A pom.xml fájlban a projekt struktúrája és tartalma van leírva.

3.4 Apache Tomcat

Az Apache Tomcat az Apache Software Foundation által fejlesztett nyílt forráskódú tisztán Java nyelven készült webszerver. Lehetővé teszi a Java Servletek és JSP webalkalmazások kiszolgálását, emellett számos további funkciót tartalmaz, amelyek hasznos platformot jelentenek a webes alkalmazások és webes szolgáltatások fejlesztéséhez és telepítéséhez [10].

Az Apache Tomcat Java Enterprise Edition része az Apache TomEE (Tomcat + Java EE = TomEE), mely több Java vállalati projektet is magába foglal: Apache OpenEJB, OpenWebBeans, MyFaces, stb.

A Websphere webszerver nagyobb támogatást és szélesebb használati kört biztosít az alkalmazás fejlesztése, telepítése és futtatása során. Megosztott környezetben sokkal jobb a TomEE webszervertől, de a saját számítógépemen történő fejlesztéshez és teszteléshez pontosan elég volt számomra a TomEE is. Fejlesztési környezetemben mindkét webszerver tudta szolgáltatni azt, amire szükségem volt én mégis egyszerű kezelhetősége miatt a TomEE-t választottam a bonyolultabb, de szélesebb körben alkalmazható Websphere-rel szemben.

3.5 Bootstrap

A Bootstrap egy frontend keretrendszer. Webalkalmazások és webes felületek tervezéséhez szolgáltat különböző HTML és CSS sablonokat, egyedi gombokat, komponenseket, beviteli mezőket. Én az AngularJS keretrendszer kiegészítéseként alkalmaztam rácsrendszerű kialakítása miatt. Bootstrapban a weboldalak sorokból épülnek fel. A bootstrapban minden egyes sor 12 cellára osztható fel. Minden cella további 12 részre osztható fel, egészen 1170 pixel szélességben.

3.6 Cacoo

A Cacoo egy online diagram és folyamatábra készítő szoftver, de képernyőtervek készítésére is tökéletesen alkalmas. Egyszerű elemekből lehet építkezni. Egy már meglévő ábrából másolással át hozható az ott kialakított struktúra vagy megtervezett elem az újra, emellett meglévő ábrákat, ikonokat is adhatunk hozzá számítógépünkről és az internetről is. Jelkészlete több kategóriára bomlik, az egyszerű alakzatoktól kezdve a webes világban használt ikonokig minden megtalálható és testreszabható benne.

A kialakított képernyőtervek összefoghatóak egy mappába, melyet projekthez adva megosztható másokkal a zárt felhasználói csoportból. A megtervezett képernyők egyesével kép, a mappa teljes tartalma pedig Portable Document Format (PDF) vagy Power Point) formátumban exportálható ki a Cacooból.

3.7 IntelliJ IDEA

Az IntelliJ IDEA a JetBrains által fejlesztett platform független szoftverfejlesztéshez használt Java integrált fejlesztési környezet (Java IDE). Közösségi és kereskedelmi kiadásban is megtalálható – JavaEE-t csak a kereskedelmi verzió támogatja. Első verzióját 2001-ben adták ki. Legfrissebb verziója, melyet én is használok a szeptemberben kiadott 2017.2.5 – ös verzió.

Közvetlen hozzáférést biztosít adatbázis- és alkalmazás szerverekhez, verziókezelő rendszerekhez. Gyors kódírást biztosít kódelemző, kiegészítő és refaktor funkcióival. Felismeri, jól kezeli az egymásba ágyazott különböző nyelveket. Egyaránt alkalmas kisméretű és nagyvállalati szintű projektek fejlesztéséhez.

Választásom azért esett az IntelliJ IDEA fejlesztőkörnyezetre, mert használata egyszerű, nagyban segíti a gyors és hibamentes kódírást. Többnyelvűsége miatt nem kell külön fejlesztőkörnyezetet használnom a backend és a frontend fejlesztéséhez. Rendelkezik local historyval, melynek köszönhetőn a fejlesztés első lépésétől nyomon követhető a program alakulása. Ennek köszönhetően nem csak a közvetlen előzmény vonható vissza, hanem bármely módosítás, mely a programban történt.

3.8 JavaScript

A JavaScript weboldalak fejlesztéséhez használt magas szintű, dinamikus, gyengén tipizált, prototípus alapú programozási nyelv. Egyike a weboldalkészítés 3 fő technológiájának. Segítségével interaktív weboldalakat hozhatunk létre. A modern webböngészők beépített JavaScript-motorral rendelkeznek a JavaScript kódok futtatásához, melyek .html fájlban és külön .js fájlban is elhelyezhetőek. Előnye a szerver-oldali programozási nyelvekhez képest, hogy az oldal újratöltése nélkül láthatjuk az eredményt.

3.9 MySQL

Az Oracle Corporation tulajdonában lévő MySQL egy strukturált lekérdező nyelv (Structured Query Language, SQL) alapú nyílt forráskódú, platformfüggetlen, több felhasználós relációs adatbázis-kezelő rendszer. A MySQL az egyik legelterjedtebb adatbázis-kezelő rendszer, melynek egyik oka, hogy a nyílt forráskódú LAMP (Linux-Apache-MySQL-PHP) összetevőjeként könnyen konfigurálható és költséghatékony alkalmazások fejlesztésére alkalmas[10][12].

Az SQL nyelvi elemeinek köszönhetően lehetőség van adatbázis elemek létrehozására, módosítására és törlésére az adatdefiníciós nyelv (Data Definition Language, DDL) segítségével. Az adatkezelési nyelvvel (Data Manipulation Language, DML) lehet a táblák tartalmát módosítani. A lekérdező nyelv (Query Language, QL) az adatokhoz való gyors hozzáférést biztosítja, míg az adatvezérlővel (Data Control Language, DCL) az adatokhoz való hozzáférési szabályok adhatóak meg a felhasználókhoz. Az adatokhoz való hozzáférés szabályozás által egyszerre többen is el tudják érni ugyanazt az adatot.

Adatbázis létrehozására, és karbantartására lehetőség van parancssoron keresztül vagy a MySQL oldaláról elérhető grafikus felülettel rendelkező adminisztrációs

eszközökkel. Többek között az Apache webszerverhez kapcsolódó PHP nyelven írt phpMyAdmin szolgáltat grafikus felhasználói felületet, mely a http://localhost/phpmyadmin címen érhető el. Munkám során én is a phpMyAdmin-t használtam az adatbázis létrehozásához, rekordok hozzáadásához, módosításához, teszteléshez és ellenőrzéshez. Az adatbázishoz tartozó táblák és azok kapcsolatai automatikusan a fejlesztett programom által kerültek létrehozásra.

3.10 SoapUI

A SoapUI egy nyílt forráskódú webszolgáltatás tesztelő alkalmazás szolgáltatásorientált architektúrákhoz (SOA) és representational state transfer-ekhez (REST). Java platformra épül, user interfésze Swing-et használ. Többek között webszolgáltatások felügyeletére, fejlesztésére, szimulálására és mockolására, funkcionális-, biztonsági- és megfelelőségi tesztelésére alkalmas. SoapUI-al tesztelhetőek a Simple Object Access Protocol (SOAP) és REST webszolgáltatások, Java Message Service (JMS), Hiper Text Trasfer Protocol(Secure) (HTTP(S)) és Java Database Connectivity (JDBC) hívások[13].

3.11 XAMPP

XAMPP egy nyílt forráskódú platformfüggetlen webszerver-szoftvercsomag. A szoftvercsomag egy integrált rendszert alkot webalkalmazások készítéséhez, teszteléséhez és futtatásához. Legfontosabb tulajdonsága és egyben legnagyobb előnye is, hogy a különböző elemek egy tömörített állományba vannak csomagolva, telepítésükhöz ezt az egy fájlt kell futtatni. Több példányban is telepíthető egy számítógépre, melyek függetlenül, a többi telepített példány megzavarása nélkül tudnak futni. A szerverek egyszerűen elindíthatóak és leállíthatóak manuálisan az XAMPP kontroll paneljében és automatikusan a programból is [14].

Az XAMPP kifejezés kifejtése:

- X – platformfüggetlenséget jelenti
- A – Apache webszerver
- M – MariaDB adatbázis kezelő (korábban MySQL)
- P – PHP szerveroldali szkriptnyelv
- P – Perl általános célú szkriptnyelv.

3.12 Technológiák értékelése és összehasonlítása

A frontend technológiák esetén is többféle megközelítés létezhet. A Vaadin keretrendszer például vastagkliensként egyesíti a frontend és backend részeket.[15] Legfőbb előnyei hogy könnyen és gyorsan megvalósítható segítségével a frontend és backend közti összetett kommunikáció, nincs szükség JavaScript file-okra és remek terméktámogatottságnak örvend, mint nyílt forráskódú dobozos termék. Hátrányos lehet viszont a sok komponens közötti nehéz eligazodás, egyes standard funkciók bővítésének nehézsége, továbbá hogy a keretrendszer nem állapotmentes, ami jelentősen rontja a skálázhatóságot. Ezzel szemben egy másik megközelítés lehet a JSP és HTML frontend és egy MVC-s pl. Spring MVC-s backend ami főleg elterjedt, nyílt szabványokra épít, magas szinten skálázható megoldást ad, támogatja a RESTful megközelítést és elősegíti a HTML/CSS/JS elemek teljes kontrollálhatóságát[16]. Hátrányként főleg a HTLM/CSS/JS ismeretek szükségessége róható fel.

Node.js[17] és Java EE összehasonlításában elmondhatjuk, hogy mindkét megközelítés jó IDE támogatottságnak örvend, elterjedt keretrendszerekkel használható és széles körben megtalálható már vállalati alkalmazásokban. Teljesítmény tekintetében a Node.js tud némiképp gyorsabb lenni, ugyanakkor a JAVA a maga 20 év múltjával lényegesen robosztusabbnak hat. Az adatbázis eléréséhez ugyanakkor a JSON használata nagy előnyt jelenthet és ez mindkét környezetben bevethető lehet.

Az AngularJS legfőbb előnyei közé sorolható, hogy gyors az újabb javított verziók kiadása, kliens orientált megközelítést támogat, fejlesztéshez és installáláshoz csak sima webszerverre van szükségünk – nincs szükség pl. külön virtuális futtató környezetre – és rendelkezik plug-in-ekkel a legelterjedtebb környezetekhez, mint Eclipse, IntelliJ és Netbeans.

Webszolgáltatások esetén SOAP és REST szolgáltatások megvalósítása közül választhatunk[18]. Míg a REST lehetőséget ad hibakezelésre beépített mechanizmussal addig a SOAP-ban ezt sokkal bonyolultabban kell megoldanunk. További előny lehet, hogy míg a SOAP jelentős middleware eszköztámogatást igényel, addig a REST csak minimális, illetve http támogatására épít. A SOAP felé billenti ugyanakkor a mérleget, hogy közvetlenül támogat szabványokat, melyek elősegítik a biztonság, megbízható üzenetküldés és komplex tranzakcionalitás megvalósítását.

Ez itt leírtak alapján a frontend fejlesztéséhez az AngularJS -tés a Bootstrap keretrendszert választottam a JavaScript programozási nyelv használata mellett. A backend megvalósítása a Java EE platformon történt. Az adatok kezeléséhez használok JSON-t, a kommunikációt pedig REST szolgáltatásokkal oldottam meg.

4 A feladat megvalósítása

Ebben a fejezetben kerül bemutatásra a feladat megvalósításának két fontos része a tervezés és az implementálás.

4.1 Tervezés

Első lépésként a felhasználókat és a számukra elérhető funkciókat határoztam meg, melyet a 4.1.1-es use-case diagram mutat be.

4.1.1 ábra: Use-case diagram

Minden a szülő és a gyerek számára elérhető funkció többségében egy-egy saját képernyőn helyezkedik el az alkalmazásban. Attól függően, hogy milyen típusú felhasználó (szülő vagy gyerek) szeretne használni egy-egy közös részt a funkciógombok elérhetősége változhat. A bank funkciói egy külön admin felületen kapnak helyet, melynek létrehozása nem tartozik szakdolgozatom feladatai közé.

A use-case ábrán láthatjuk, hogy a Gyerekbank alkalmazás igazi felhasználói a gyerekek és a szülők. A bank csak az adminisztrációt illetve a számlavezetési kötelezettségeinek eleget téve vesz részt az alkalmazás világában. Sem közvetve, sem közvetlenül nem befolyásolja egyetlen funkció végrehajtását sem.

A szülő és a gyerek felhasználó számára elérhető funkciók 3 nagy csoportba oszthatóak:

1. közös funkciók,
2. csak a gyerek számára elérhető funkciók,
3. csak a szülő számára elérhető funkciók.

Közös funkciók a be- és kijelentkezés, személyes adatok-, számlatörténet-, igényelt hitelek- és lekötések megtekintése és ezen funkciók adattartalma teljes mértékben megegyezik a két felhasználó típusnál. A csak a gyerek számára elérhető funkciók a csak a szülőnek elérhető funkciókból adódnak, például a szülő tud egyedi hitelkonstrukciót kiírni a gyerek számára ennek megfelelően a gyerek felveheti ezeket. A szülő elbírálja a hitelfelvételi kérelmeket, az elfogadott hiteleket a gyerek elő- és végtörlesztheti – a heti törlesztés automatikusan történik.

Ennek megfelelően bejelentkezés után mást-mást lát a szülő és a gyerek. A megjelenő oldalak mindkét esetben két nagy részre bonthatóak: keretrendszer és a megjelenített funkció. A keretrendszer tovább bontható a képernyő bal oldalán elhelyezkedő menüsávra és a képernyő tetejét elfoglaló összefoglaló részre. A menüsávban az adott felhasználó típusnak elérhető funkciók jelennek meg egy-egy menüként. Az összefoglaló részben középen a gyerek neve és egyenlege látható, a jobb sarokban pedig a felhasználó neve és a kijelentkezés gomb. A képernyő többi részét a megjelenített funkció foglalja el. Bejelentkezés után mindkét felhasználó típusnak a személyes adatai töltik ki ezt a részt.

Amíg a gyerekek esetében az összefoglaló részben található középső rész teljes egészében csak olvashatóan jelenik meg, addig a szülők egy legördülő listából ki tudnak választani egy gyereket, rögtön alatta a kiválasztott gyerekhez kapcsolódó egyenleg jelenik majd meg. A menüben azon funkciók, melyek a gyerekhez kapcsolódnak – a személyes adatainak megtekintése kivételével mind – a kiválasztott gyerekhez tartozó adatokat fogják megjeleníteni.

4.1.1 Adatbázis séma

Az alkalmazás megvalósításához tárolni kell a felhasználókat, a szülők által kiírt befektetési tételeket, hitel konstrukciókat, a felvett hiteleket és a befektetett összegeket. Hiteltörlesztés, utalás, befektetés esetén szükséges az egyenleg és a számlatörténet módosítása mindkét fél esetén, emellett rendszeres utalásnál jegyezni kell az utalás tényét és érvényességének idejét a bankban.

Mivel az alkalmazás jelenleg nem kapcsolódik a bankhoz és nem kommunikál vele, így a banktól lekérdezendő adatoknak is létrehoztam egy-egy adatbázistáblát a könnyebb kezelhetőség érdekében. A 4.1.2-es ábrán látható a jelenlegi adatbázis séma, melyből az *AccountHistory* és a *Transaction* tábla, illetve a *Person* táblában lévő *balance* mező a banki kommunikációt helyettesíti, ezen elemek az éles adatbázisban nem találhatóak majd meg.

Az adatbázis redundanciát tartalmaz az *Investment* és *Loan* táblájában. Kapcsolótáblákat a feloldásukhoz a könnyebb és gyorsabb kereshetőség érdekében nem hoztam létre.

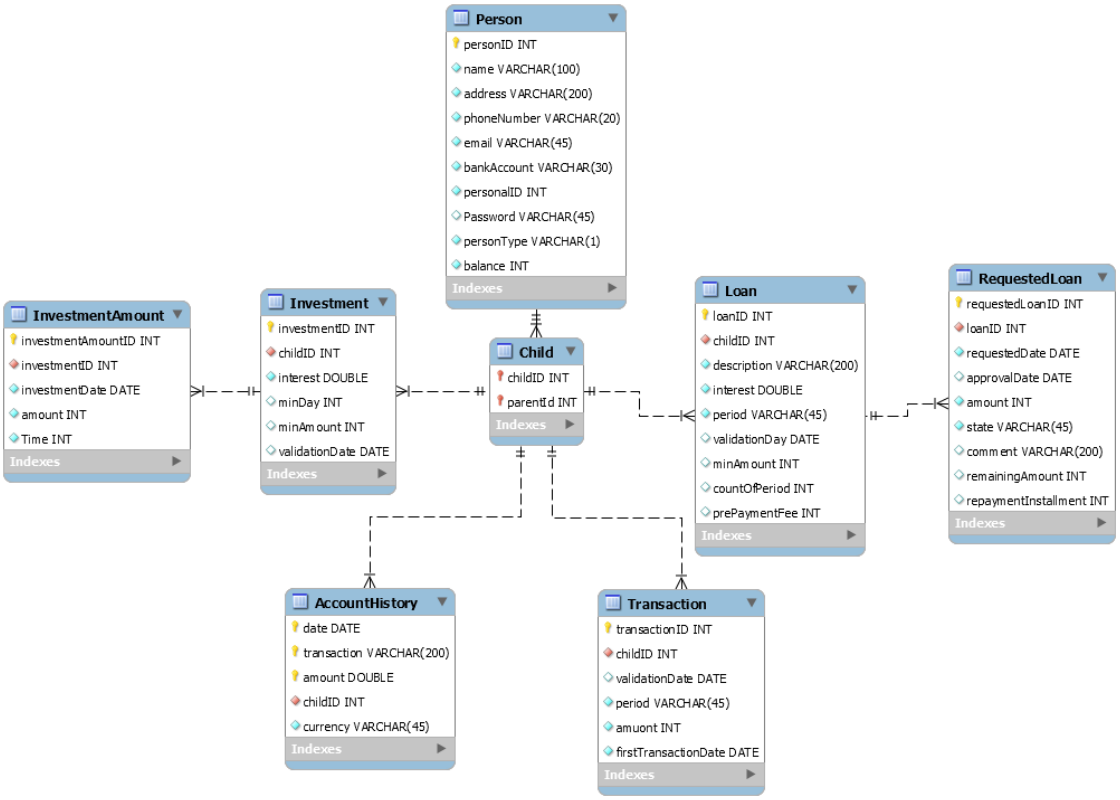
A *Person* tábla tartalmazza a felhasználók személyes adatait. Mivel mind szülő, mind gyerek felhasználó esetén ugyanazokat az adatokat szeretném megjeleníteni az alkalmazásban, így nem hoztam létre külön táblát a két felhasználó típusnak, csupán egy új *type* mezőt vettem fel típusának azonosítására. A szülőket és a gyerekeket a *Child* táblában rendelem össze. A korábbi tervezői döntésnek megfelelően a gyerekek azonosítója egyedi, ezzel szemben a szülőké nem annak érdekében, hogy egy szülőhöz több gyereket is hozzá tudjunk rendelni, míg egy gyerek csak egy szülőhöz tartozhat. A *Person* tábla az *ID* mezője külső kulcs a *ChildParent* tábla *childID* és *parentID* mezőjénél is.

Az *Investment* tábla tartalmazza a szülők által kiírt befektetési tételeket. Az *Investment* és a *ChildParent* tábla a *childID* mezőn keresztül kapcsolódnak egymáshoz,

mivel minden befektetési tétel egy gyerekhez van rendelve. A befektetési tételek azért vannak gyerekekhez kötve és nem szülőkhöz, hogy a szülők gyerekeiknek akár eltérő befektetési tételeket is kiírhassanak.

Egy befektetési tétel ugyanazon gyerek által többször is felhasználható, ezért célszerűnek láttam egy külön táblában - *InvestmentAmount* - tárolni a befektetett összegeket hivatkozva a befektetési tétel azonosítójára. A gyerek azonosítója megtudható a befektetési tételből ezért nincs szükség külön hivatkozni rá ebben a táblában is.

A szülő által kiírt hitel konstrukciók a *Loan* táblában vannak tárolva. A befektetési tételekhez hasonlóan itt is minden hitel egy gyerekhez van hozzákapcsolva. A felvett hitelek a *loanID*-n keresztül kapcsolódva a *Loan* táblához a *RequestedLoan* táblában találhatóak.



4.1.2 ábra: Adatbázis séma

4.1.2 Be- és kijelentkezés

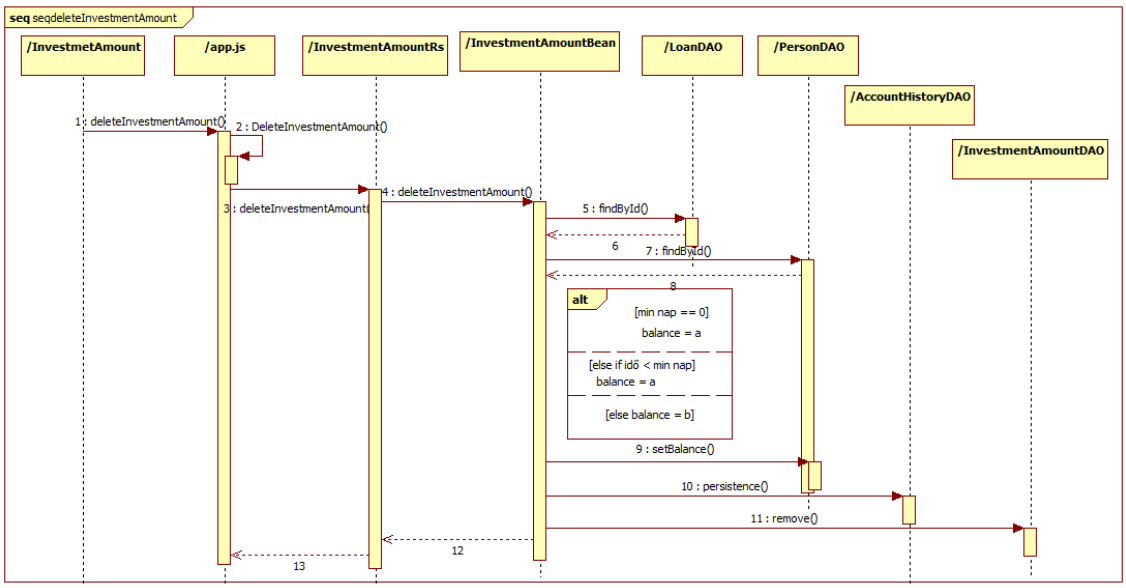
A bejelentkezés FORM-ot használó autentikációval történik meg, ennek köszönhetően létre tudtam hozni egy a gyerekbank weboldal stílusának megfelelő bejelentkezési képernyőt. Bejelentkezéshez szükség van a felhasználó email címére és

jelszavára. Amennyiben az autentikálás sikeres volt a felhasználó típusának megfelelő felület jelenik meg.

A biztonság érdekében a rest hívásokra is rákerült egy authentikálás. Ebből a felhasználó semmit nem vesz észre, bejelentkezés után a rest hívások fejlécében közlekednek a bejeletnkezett felhasználó azonosításához használt adatok. Ennek köszönhetően nem szükséges jelszó és felhasználónév megadása a rest hívásokkor. Kijelentkezni minden felhasználó esetén a jobb felső sarokban található Kijelentkezés gombra kattintva lehet. Ekkor visszanavigál minket a program a bejelentkező felületre.

4.1.3 Befektetett összegek

A gyerek folyamban lévő befektetéseit a *Befektetett összeg* menüpont alatt mind a szülő, mind pedig a gyerek nyomon követheti. A befektetett összegek a befektetés időtartalma alatt nem jelennek meg az aktuális egyenlegben. Befektetés felbontására ugyanezen a képernyőn van lehetőség, melyre csak a gyereknek van joga. Ebben az esetben a kamat csak akkor íródik jóvá, ha a szülő által kiírt befektetéshez megadott minimum befektetési idő után bontotta fel a gyerek a befektetést. Amennyiben a szülő nem adott meg minimum befektetési időt, úgy a kamat minden esetben jóváíródik.



4.1.3. ábra: Befektetés felbontás szekvencia diagram

Ahogyan a 4.1.3-as szekvencia diagram is mutatja, befektetés felbontásakor először leellenőrzésre kerül, hogy a befektetési tételhez tartozik e minimum befektetési idő. Amennyiben igen, úgy meghatározásra kerül, hogy mennyi ideig volt befektetve az

összeg. Ha több, mint a minimálisan előírt idő, akkor a kamat jóváírásra kerül. Ezután pozitívan módosul a gyerek egyenlege a befektetett összeggel és az esetleges kamattal, megjelenik a számlatörténetben az átvezetés majd törlésre kerül az adatbázistáblából a gyerek befektetéséhez tartozó rekord.

Abban az esetben, ha a befektetés időtartalmának lejártáig nem kerül felbontásra a befektetés automatikusan megtörténik annak lezárása. A befektetett összeg visszakerül a felhalmozódott kamattal együtt a gyerek egyenlegébe és megjelenik számlatörténetében is ez a tranzakció. A számlatörténetben nem kerül megkülönböztetésre, hogy a befektetés felbontották vagy pedig lejárt.

4.1.4 Felhasználói adatok

Gyerek számára a saját adatai az *Adatok*, míg a szülő számára a *Saját adatok* menüpont alatt érhetőek el. Ezen adatok a bank nyilvántartásából származnak. Lehetőség van a telefonszám, email cím és lakhely megváltoztatására. Ezen adatok csak a Gyerekbank adatbázisában kerülnek módosításra, a bank felé nem továbbítja őket az alkalmazás. Email cím megadása kötelező, mivel ezzel tud a felhasználó belépni. A belépéshez használt jelszó megváltoztatására is ezen a képernyőn van lehetőség.

A szülő a saját felületén az Adatok menüpont alatt éri el a fent kiválasztott gyerekének adatait. Neki nincs lehetősége a gyerek adatainak megváltoztatására, ahogyan jelszavát sem láthatja.

4.1.5 Felvett hitelek

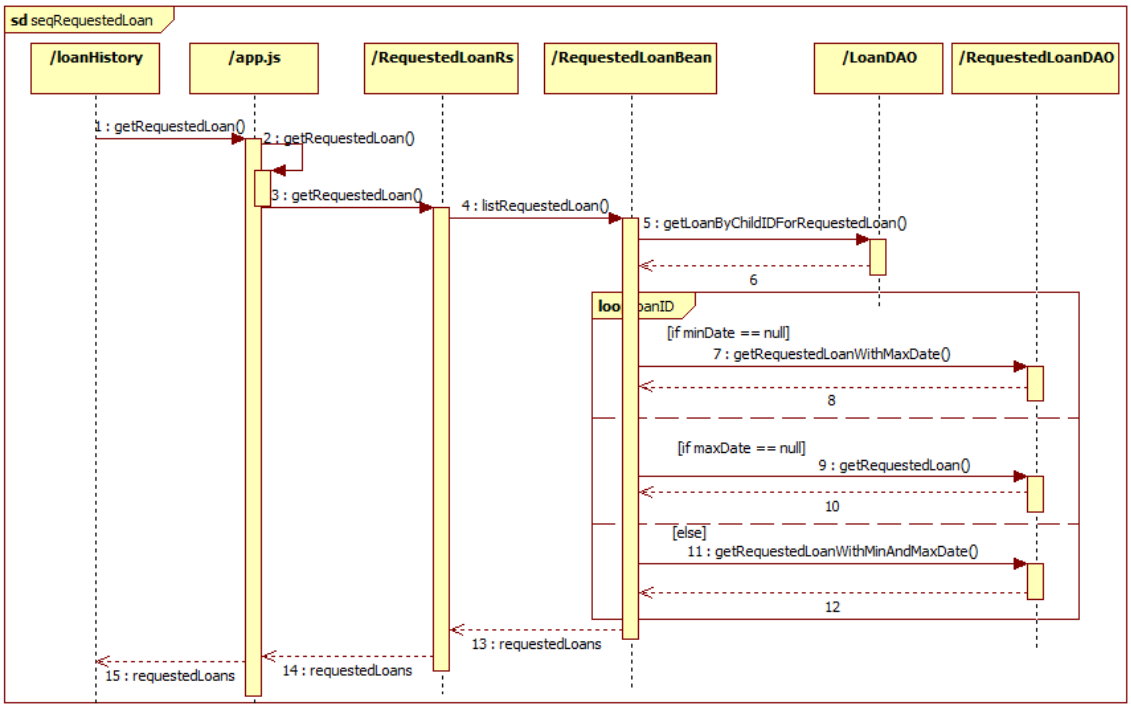
A gyerek által felvett hiteleket a *Igényelt hitelek* menüpont alatt lehet megtekinteni igénylésük dátuma szerinti csökkenő sorrendben. Ebben a táblázatban minden státuszú felvett hitel megtalálható. Alapértelmezetten az elmúlt 30 napban igényelt hitelek jelennek itt meg, ez az intervallum bárki számára módosítható. Lehetőség van egy intervallumban igényelt hiteleket megjeleníteni, ha a –tól, -ig dátumok mindegyikét kitöltjük. Csak a –tól dátum kitöltése esetén a megadott dátumtól a mai napig igényelt, míg csak a –ig dátum kitöltésével az összes igényelt hitel láthatóvá válik.

Az igényelt hitelek státuszai:

- Elbírálás alatt – a gyerek által igényelt elbírálatlan hitel.

- Elfogadott – a gyerek által igényelt jóváhagyott hitel.
- Törlesztés alatt – a gyerek által igényelt jóváhagyott hitel, melyben megtörtént az első törlesztés, legyen az akár az automatikusan levont törlesztőrészlet vagy előtörlesztés.
- Törlesztve – a gyerek által igényelt elfogadott és maradéktalanul visszafizetett, lezárt hitel.
- Elutasítva – a gyerek által igényelt elutasított hitel.

A státuszok állítására a felhasználónak manuálisan nincs lehetősége, azt a program automatikusan végzi a felhasználó műveletei alapján. A gyerek által megigényelt hitel rögtön Elbírálás alatt státuszba kerül. Annak megfelelően, hogy az adott hiteligénylést a szülő jóváhagyja vagy elutasítja Elfogadott vagy Elutasított státuszba áll át. Elutasított státuszból a hitel nem lép át más státuszba. Elfogadott státuszból a hitel az első törlesztés alkalmával Törlesztés alatti státuszt vesz fel. Amennyiben tőketartozása 0 lesz Törlesztve státuszba áll át, melyből továbblépni úgyszintén nem lehet.



4.1.4 ábra: Felvett hitel lekérdezés szekvencia diagram

4.1.6 Számlatörténet

A *Számlatörténet* minden esetben a gyerek számlatörténetét mutatja. Az igényelt hitelek menüponthoz hasonlóan itt is lehetőség van az egész számlatörténet és egy időszakban történt tranzakciók megjelenítésére is. A 4.1.5-ös ábra mutatja egy időszakos tranzakció megjelenítését.

Az alkalmazásom azért nem jeleníti meg a szülők tranzakcióit, mert azok nem szükségesek ahhoz, hogy a szülő a Gyerekbankban a gyerekét segítse. Ennek ellenére az alkalmazás a Gyerekbankban történt egyenlegváltozásához kapcsolatos tranzakcióit neki is kezeli. Jelen esetben lementi az adatbázisba, éles környezetben való alkalmazás esetén pedig továbbítja a bank felé.

4.1.5 ábra: Számlatörténet – egy időszak tranzakcióinak lekérdezése

4.1.7 Befektetés

Befektetni a szülőhöz a *Befektetés* menüpont alatt lehet. Itt láthatóak a szülők által kiírt még érvényes befektetési tételek. Minden befektetési tételhez tartozik egy érvényességi dátum mező. Amennyiben ez a szülő által meg lett adva, úgy a gyerek csak addig tud befektetést kezdeményezni az adott tételre. Ha nincs megadva érvényességi dátum, akkor a tétel törléséig a gyereknek megjelenik az a befektetési tétel is.

Minden befektetési tételhez tartozik a szülő által megadott minimális befektetési összeg és nap, mely alatt a gyerek nem fektethet be az adott tételbe. Ezen feltétel vizsgálva van a felületen. Addig letiltásra kerül a befektetés gomb az adott tételhez, míg a megfelelő értékekkel kitöltésre nem kerül a befektetés időtartama és összege mező. A befektetés gombra kattintva elindul a befektetési folyamat. Először egy új befektetett összeg sor kerül mentésre az adatbázisba a befektetési tétel azonosítójára hivatkozva. Ezután a gyerek számlájáról átvezetésre kerül az összeg a befektetés ideje alatt egy

magtakarítási számlára, mely tranzakció megjelenik az egyenlegében és a számlatörténetében is.

Befektetés felvételekor a jóváírható kamat mértékét oktatási célból nem jelenítem meg. Egy egyszerű képlet segítségével könnyen kiszámolható akár számológéppel is. Egyszerűségéből adódóan hamar megjegyezhető a képlet, melyre a későbbiekben többször szüksége lehet, így jó ha minél hamarabb megismerkedik vele a gyerek.

4.1.8 Hiteligénylés

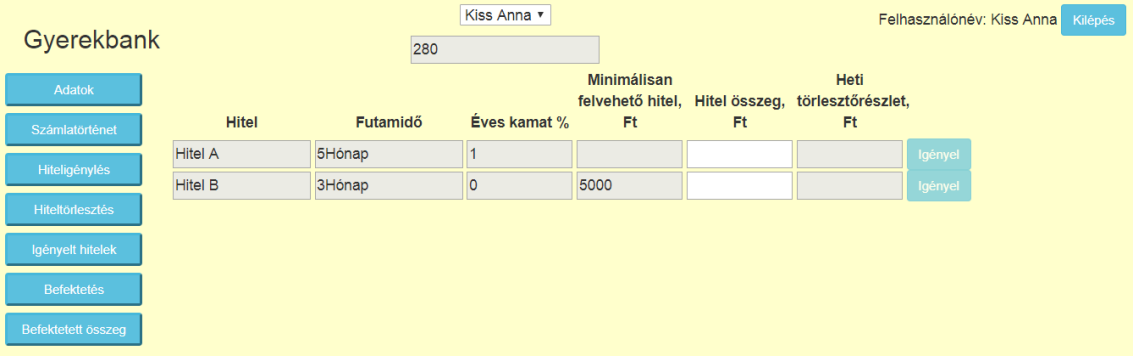
A gyerek a *Hiteligénylés* menüpontban tud a szülőtől hitelt felvenni. Soronként jelennek meg a gyerek számára a szülő által kiírt felvehető hitelek. A hitel kamatát és futamidejét a szülő határozta meg, ennek változtatására a gyereknek nincs lehetősége. Amennyiben a szülő meghatározott egy minimális összeget, mely alatt a hitel nem igényelhető meg, akkor az adott hitelhez tartozó hiteligénylés gomb mindaddig inaktív marad, míg a gyerek kisebb összeget szeretne felvenni, mint ami meg van határozva.

A hitel törlesztése heti rendszerességgel automatikusan történik a futamidő alatt azonos nagyságú törlesztőrészletek levonásával. Ennek megfelelően, amikor a gyerek begépeli a felvenni kívánt hitel összegét a törlesztőrészlet oszlopban az adott hitelhez megjelenik a heti annuitásos törlesztőrészlet. Az annuitásos törlesztőrészletet a következő képlet segítségével számolom ki [19]:

$$Törlesztőrészlet = \frac{hitelösszeg * kamat}{1 - \frac{1}{(1 + kamat)^{periódus}}},$$

ahol a periódus a futamidő hetekben mérve. Éves futamidőnél évenként 52 hét a periódus, havi futamidőnél pedig havonta 4 hét. A kamat pedig a hitel futamidejére vonatkozó kamatot jelenti.

Az igényelt hitel beküldés után a szülőhöz kerül elbírálásra. A gyerek megtekintheti az *Igényelt hitelek* menüpont alatt új elbírálás alatti hitelét. A szülői döntésről az alkalmazás ugyanezen képernyőjén értesülhet.



4.1.6 ábra: Hiteligénylés képernyő

4.1.9 Hiteltörlesztés

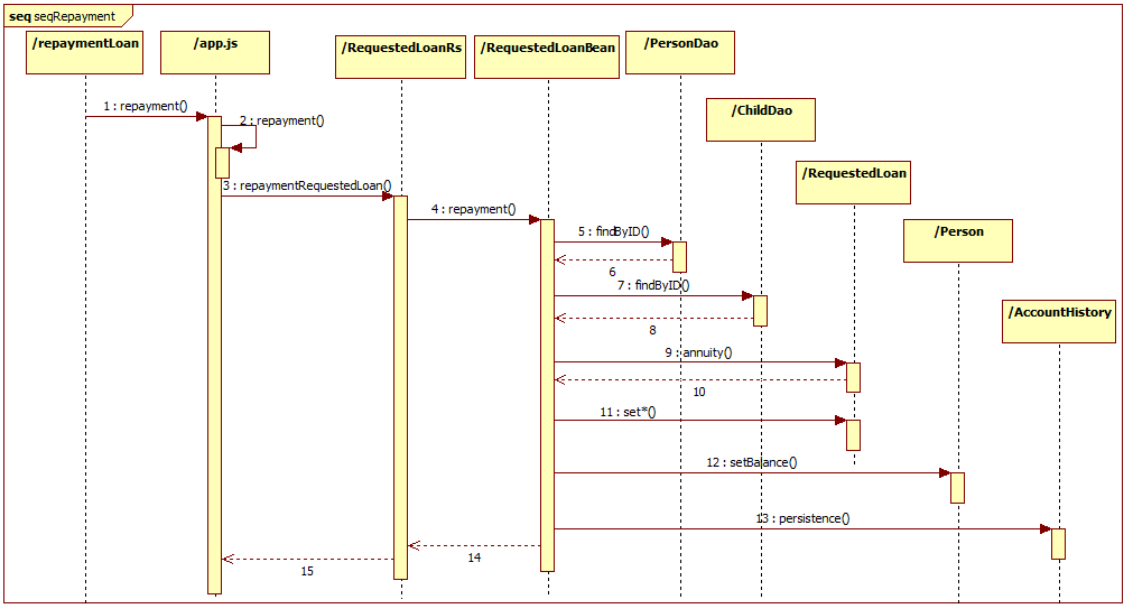
Itt láthatóak a gyerek *Elfogadott* és *Törlesztés alatt* státuszban lévő hitelei. Magát a rendszeres hiteltörlesztést nem itt kell végrehajtani, az automatikusan megtörténik a rendszer által, külön képernyő nem tartozik hozzá. Emellett lehetősége van a gyereknek eseti elő- és végtörlesztésre.

Minden hitelhez megtalálható a tőketartozás mértéke. Rendszeres törlesztés alkalmával levonásra került a tőketartozásból a felvett hitel egy heti törlesztett tőketartozása, mely a felvett hitel és a hiteltörlesztés futamideje alatti hetek számának a hányadosa.

Előtörlesztéshez meg kell adni egy összeget, mely nem lehet nagyobb a gyerek egyenlegénél és a tőketartozás mértékénél. Amennyiben ez a feltétel nem teljesül, úgy az Előtörlesztés gomb inaktívvá válik. Végtörlesztés esetén nincs szükség összeg megadására, az alkalmazás a tőketartozással végtörleszt.

Az előtörlesztés fordulónapokon valósul meg, ettől eltekintve az alkalmazás más napokon is enged előtörleszteni. Ennek alkalmával az előtörlesztett összegből levonásra kerül a következő esedékes hiteltörlesztés összege, a maradék összeggel pedig csökken tőketartozás. Ezután újraszámolásra kerül az új tőketartozással a heti törlesztőrészlet a maradék időre. Végtörlesztés esetén a tőketartozás lenullázódik a hitel pedig Törlesztve státuszba kerül.

Az elő- és végtörlesztés díja, melyet a szülő a hitelhez meghatározott a törlesztendő összegen felül kerül levonásra a gyerek egyenlegéből. Ez az összeg, ami a törlesztendő összeg megadott százaléka, a szülő számára kerül átutalásra a törlesztett összeggel együtt egy tranzakció keretében.



4.1.7 Hitel elő- és végtörlesztés szekvenciadiagram

4.1.10 Befektetési tétel kiírás és módosítás

Befektetési tételek kiírására csak a szülőnek van lehetősége. Egyazon képernyőn láthatja és módosíthatja gyerekének kiírt befektetési tételeit és hozhat létre újat. Fontos, hogy többgyerekes szülők esetén mindig csak az aktuálisan kiválasztott gyerek tételei jelennek meg a képernyőn, egyedül számára tud új befektetési tételt kiírni.

A meglévő befektetési tételeknek kizárólag az érvényességi ideje módosítható, törölhető. Lehetőség van a kiírt tételek törlésére is, ekkor azonban az adatbázisban nem kerül törlésre a befektetési tétel, csupán az érvényességi dátuma módosul a tegnapi napra. Ennek azért van jelentősége, mert a befektetett összegek a befektetési tételre hivatkoznak. Megjelenítésükkor mind a befektetett összeg, mind a befektetési tétel mezőiből kerülnek ki értékek. Ha törlésre kerülne a befektetési tétel, akkor elveszne a kapcsolat és a befektetett összeghez tartozó információk egy része is.

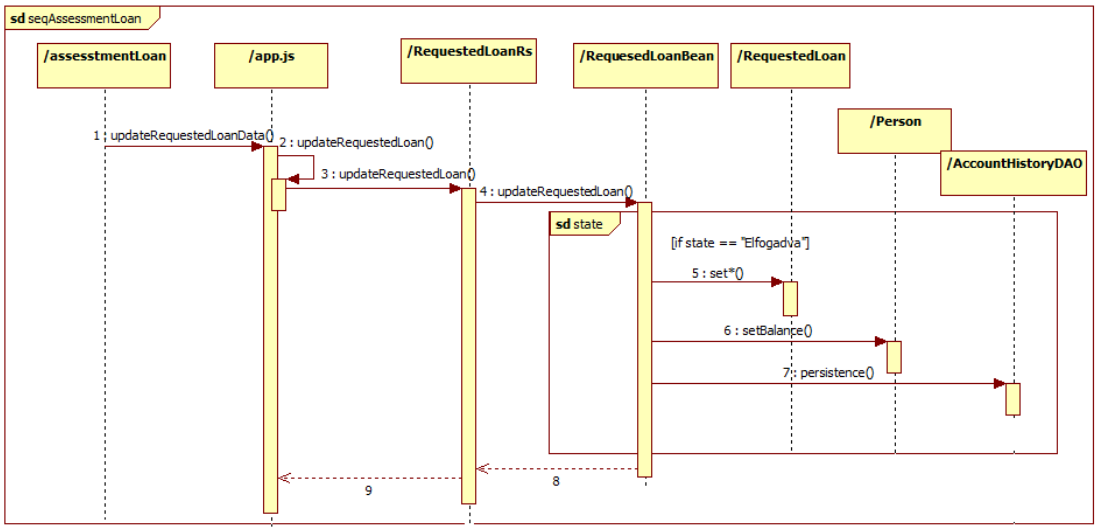
Új befektetési tétel kiírásakor egyedül az éves kamat megadása kötelező. A többi mező kitöltése a befektetési tétel felvételének szigorítása érvényességi idő, minimálisan befektethető összeg és nap tekintetében.

4.1.11 Hitel elbírálás

Hitel elbírálást a szülő egyszerre csak egy gyerekének tudja elvégezni. Ebben a menüben csak azok a hiteligénylések jelennek meg, melyek státusza az *Elbírálás alatt*. A szülőnek megjelenítésre kerülnek a hitel fontosabb adatai, az igényelt hitel összege és

a heti törlesztőrészlet. Az alkalmazás nem biztosít számára semmilyen visszajelzést arról, hogy képes e a gyerek törleszteni az általa felvenni kívánt hitelt – többek között ez függ a jövőbeni rendszeres és egyszeri utalásoktól, a gyerek egyenlegétől, illetve hogy mennyit szeretne költeni a felvenni kívánt hitelből. Hitel elbírálását a gyerekkel előzetesen egyeztetve a részletekről saját maga tudja elvégezni. Az elbírálandó hitelhez megjegyzést fűzhet, mely elutasítás esetén kötelező.

Hitel igénylés elfogadása esetén az igényelt összeg azonnal átutalásra kerül a gyerek számára. Módosul az egyenlegük és megjelenik a tranzakció a számlatörténetükben. A hitel első törlesztőrészlete az elbírálástól számított 7. napon kerül levonásra a gyerektől.



4.1.8 ábra: Hitel elbírálás szekvencia diagram

A 4.1.8-as ábrán látható a hitel elbírálás folyamatának szekvencia diagramja. A set*() esetében a state, comment, remainingAmount, approvalDate értékek kerülnek beállításra.

4.1.12 Hitel konstrukció kiírás

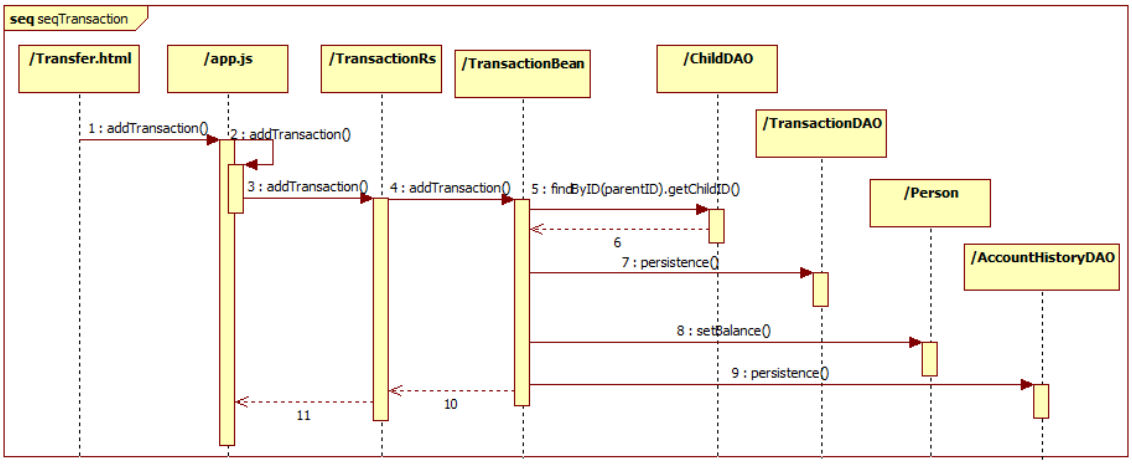
A szülő gyereke számára egyedi hitelkonstrukciókat írhat ki, az alkalmazás nem korlátozza összeállításában. A kiírt hitelkonstrukció a Hitel menüpontba való következő belépéskor megjelenik a gyerek számára is.

A befektetési tétel kiírásához hasonlóan itt is csak az érvényességi dátum szerkeszthető egy meglévő hitelkonstrukció esetén. Törlésekor pedig az érvényességi dátuma módosul a tegnapi napra.

4.1.12.1 Utalás

A szülő a Gyerekbankból is indíthat gyereke számára utalást. Lehetősége van egyszeri és rendszeres utalás indítására is. Egyszeri utaláshoz csupán az összeg megadására van szükség, rendszeres utaláshoz az összeg mellett meg kell adni az utalás periódusát, mely lehet napi, heti vagy havi. Amennyiben érvényességi időt is megad a szülő, úgy az utalások a megadott érvényességi dátumig fognak teljesülni.

A 4.1.9 ábrán az utalás szekvencia diagramja található. Először megvizsgálásra kerül, hogy a periódusban jött e érték, ha igen, akkor létrehoz egy új rendszeres utalást a gyerek számára. Ebben megadja a firstTransactionDate dátumának a mai napot. Erre azért van szükség, hogy az alkalmazás ki tudja számolni, hogy mikor kell végrehajtani az utalást. Ezután következik az egyenleg módosítása a szülő és a gyerek esetében is, majd létrehozásra kerül egy új rekord a számlatörténetben.



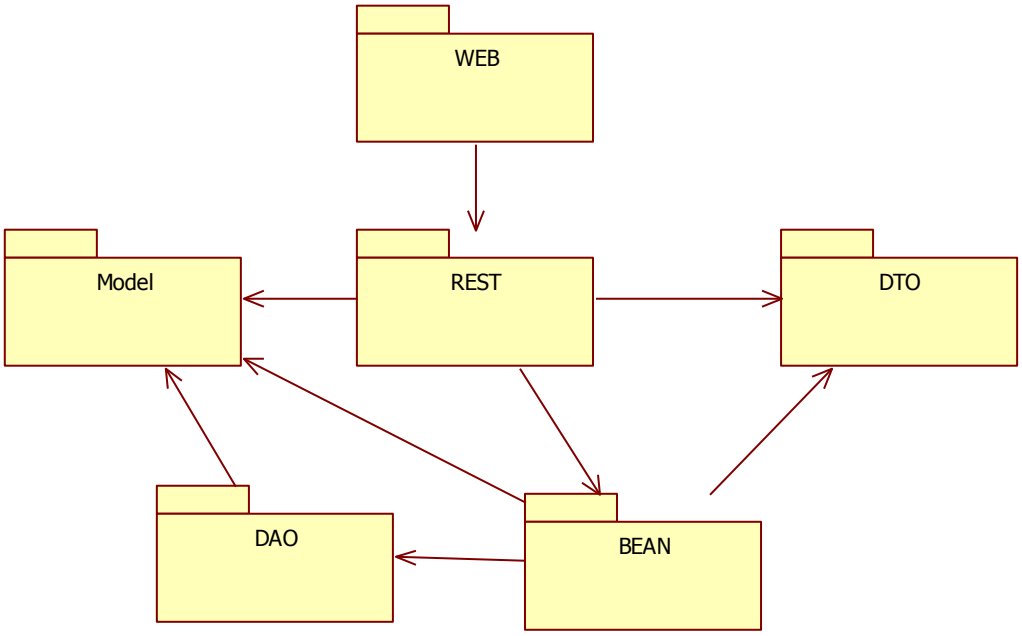
4.1.9 ábra: Utalás szekvencia diagram

A szülő számára a felületen az új utalás létrehozása mellett csak a rendszeres utalások láthatóak. A már létrehozott utalásnak egyedül az érvényességi dátuma szerkeszthető, amennyiben a periódusát vagy az összegét szeretnénk megváltoztatni, úgy új rendszeres utalást kell létrehozni. A rendszeres utalás törlésével az utalások megszűnnek és az adatbázisból is törlésre kerülnek.

4.2 Implementálás

Alkalmazásom felépítését egy céges Java EE webalkalmazást bemutató projekt mintájára készítettem el, ahol minden réteg minden eleme külön package-ben található. Az adatbázisban létrehozott tábláknak minden backand oldali package-ben megtalálható a megfelelő osztálya mely az adott szinten végrehajtandó műveleteit valósítja meg. A

4.2.1-es ábra a package-k kapcsolatát mutatja be. A *Web* packege a *Rest* packege-vel van összeköttetésben, csak vele kommunikál. A *Rest* JSON formában a *Model*, és *DTO* elemeket küldözget a *Web*-nek, emellett a funkciók megvalósításához a kapott objektumokat továbbadja az üzleti logikát megvalósító *Bean* függvényeinek. A Bean *DTO* és *Model* osztálybeli elemeket ad vissza a *REST*-nek. Az adatbázisban való lekérdezéshez a *DAO* osztályok elemeit használja. A *DAO* az entitásokat megvalósító osztályok elemein végez műveletet.



4.2.1 ábra: Package-k kapcsolata

4.2.1 Business

A business package osztályaiban valósulnak meg az alkalmazás működéséhez szükséges logikák. A use-case-ben létrehozott minden funkciónak egy-egy osztály egy-egy függvénye felel meg. Emellett itt található az automatikusan végrehajtandó műveltekhez szükséges funkciók és az időzítő is. Az entitásokhoz külön Bean osztályt hoztam létre az Entitásnév+Bean névkonvencióval ellátva.

A Bean osztályoknak nincs ősosztálya, mivel nincsenek olyan közös műveleteik, melyeket érdemes lenne egy közös ősosztályba kiszervezni. Minden Bean osztály @Stateless annotációval van ellátva. A funkciók a végrehajtáshoz csak a metódus paraméteriből kapott értékeket használják, így nem szükséges állapottárolás.

A Bean osztályban megvalósított funkciók valamilyen azonosító alapján dolgoznak, lekérdezésekkor ez az azonosító általában a gyerek azonosítója. Azonosításra a Bean osztályok függvényeibe mindig két érték kerül átadásra, az egyik a REST hívás fejlécéből kapott bejelentkezett felhasználó azonosítója, a másik a hívás paraméteréből kapott azonosító. Az utóbbi csak szülő felhasználó esetén érkezik, amikor a gyerekéhez tartozó adatokat – számlatörténet, befektetés, stb – szeretné lekérdeni. Mivel a paraméterből kapott azonosító illetéktelenek által könnyen módosítható, ezért a művelet végrehajtása előtt ellenőrzésre kerül, hogy a paraméterként kapott azonosító hozzá van e rendelve a bejelentkezett felhasználóval a *Child* táblában. Amennyiben nem található egyezés, úgy a műveletek nem kerülnek végrehajtásra, az adatlopások ezzel biztosítva vannak. Az alábbi kódrészlet bemutatja a fent említett ellenőrzést a személyes adatokhoz való hozzáférés előtt.

```
public Person listPerson(Integer id, String email) {
   Person person = new Person();

   if (id != null) {
      ArrayList<Child> child = new ArrayList<>();
      child.addAll(childDAO.getChild(personDAO.findPersonByEmail(email)
         .getId()));
      Boolean relationship = false;
      for (int i = 0; i < child.size(); i++) {
          if (child.get(i).getChildID() == id) {
             relationship = true;
          }
      }

      if (relationship) {
          person = personDAO.findById(id);
      }
      else {
         throw new IllegalArgumentException("Hibás személy azonosító”);
      }
   }
   else if (email != null) {
      person = personDAO.findPersonByEmail(email);
   }
   return person;
}
```

4.2.1 kód: Személyes adatok lekérdezése ellenőrzéssel

A heti törlesztőrészlet levonásához, rendszeres utalásokhoz és a befektetések lejáratukkor való átvezetéséhez gyerek számlájára a nekik megfelelő Bean osztályban létre van hozva egy függvény, mely ellenőrzi, hogy az adott napon van e olyan tétel, melyhez tranzakciót kell végrehajtani. Amennyiben igen, úgy módosításra kerülnek az érintett személyek egyenlegei és létrejön 1-1 új számlatörténet rekord számukra. A

TimerBean osztály run() függvényében az időzítő lejártakor meghívódnak ezek a függvények. Az időzítő 24 órás intervallumra van beálltva.

4.2.2 DAO

Az üzleti logikában a Bean osztályok függvényeihez szükséges adatbázis műveleteket a DAO osztályok biztosítják. Ők felelnek az adatbázis elérésért, a rekordokkal történő műveletek végrehajtásáért. Egyfajta közvetítő szerepet töltenek be az adatbázis és az alkalmazás között. Legnagyobb előnye a DAO használatának, hogy az alkalmazás objektumai függetlenek maradnak az adatbázis rekordoktól. Amennyiben a DAO interfész változatlan marad, úgy anélkül lehet módosításokat végrehajtani az adatbázisban, hogy azok az üzleti logikát érintenék, ugyanez fordítva is igaz.

A DAO osztályok ősosztálya a GyerekBankBaseDAO, mely az alapvető adatbázis műveleteket tartalmazza: létrehozás, törlés, keresés ID alapján, keresés feltételek nélkül. Minden DAO osztály használja ezen műveletek illetve tartalmaz egy EntityManager példányt ezért gondoltam célszerűnek ezeket kiszervezni egy külön ősosztályba. Alkalmazás fejlesztésekor az entitások az EntityManager interfészen keresztül érhetőek el. Az EntityManager példányok entitások olyan halmazával dolgoznak, melyben minden elsődleges kulccsal rendelkező perzisztens példányhoz egyetlen egyedi memóriabeli példány tartozik. Az ilyen entitáshalmazokat perzisztenciakontextusnak hívják. Igaz, hogy az EntityManager osztályok nem szálbiztosak, de EJB környezetben nincs többszálúság így ez szerencsére nem okoz gondot az alkalmazás fejlesztése során [7].

Minden entitáshoz létrehozásra került egy DAO osztály mely az Entitásnév+DAO névkonvenciót használja. Ezen DAO osztályokban a speciálisabb lekérdezéseket valósítottam meg, például hitelek lekérdezése a gyerek ID-ja alapján, jóváhagyásra váró hiteligénylések listázása, számlatörténet lekérdezése adott intervallumon. Az alábbi példa az előtörleszthető hiteleket listázza ki a hitel azonosítója alapján.

```
public List<RequestedLoan> getUnderRepaymentLoan(Integer loanID){
    CriteriaBuilder cb = getEm().getCriteriaBuilder();
    CriteriaQuery<RequestedLoan> cq = cb.createQuery(RequestedLoan.class);
    Root<RequestedLoan> rootEntry = cq.from(RequestedLoan.class);
    cq.where(cb.equal(rootEntry.get("loanID"), loanID),
        cb.or(cb.equal(rootEntry.get(("state"), "Elfogadva"),
            cb.equal(rootEntry.get(("state"), "Törlesztés alatt")));
    CriteriaQuery<RequestedLoan> all = cq.select(rootEntry);
```

```
        TypedQuery<RequestedLoan> allQuery = getEm().createQuery(all);
        return allQuery.getResultList();
    }
```

4.2.2 kód: Elő- és végtörleszthető hitelek adatbázisból való lekérdezése

4.2.3 DTO

Az adatátviteli objektumok (Data Transfer Object, DTO) a folyamatok között közvetítenek adatokat. Esetemben a felületről érkező adatok 1-1 DTO elemként érkeznek meg a backend rest hívásának paraméterébe. Minden adatbázistáblához készítettem egy DTO osztályt is, melyek @XmlRootElement annotációval vannak ellátva. A @XmlRootElement lehetővé teszi az objektumok XML formátumú szöveggé alakítását. A konverzió oda-vissza automatikus.

A DTO osztályok létrehozására azért kézenfekvő, mert szükségem volt a Model osztály általi válasz kibővítésére, anélkül, hogy az adott elemeket az adatbázisban is létrehoznám. Például igényelt hitel esetén a hitel nevét, futamidejét, kamatát, elő- és végtörlesztési díját a Loan osztály tartalmazza, míg az igénylés dátumát, elbírálás dátumát, – ha van – igényelt összeget, státuszt és megjegyzést a RequestedLoan osztály tartalmazza. Ez esetben egyszerűbb és költséghatákonyabb létrehozni egy osztály az adatok közlekedtetésére, mint két kérést elindítani a backend felé. Ugyanez a helyzet a számlatörténet lekérdezése esetén, ahol a frontendtől meg kell kapnunk az intervallum kezdő és vég dátumára is a gyerek Id-ja mellett a pontos lekérdezéshez. Ennek ellenére a lekérdezni kívánt időszak kezdő- és vég dátuma máshol nem kerül felhasználásra, adatbázisba történő mentése felesleges.

4.2.4 Model

Az adatokat MySQL adatbázisban tárolom. A táblák és a kapcsolatok a Model osztályaiban kerülnek definiálásra. Az adatbázis kapcsolat felépítése és a séma frissítése a persistence.xml fájlban van megvalósítva. Az entitások osztályaiból legenerálhatóak az adatbázis sémái, melyek minden SessionFactory létrehozásakor megtörténnek. A *hibernate.hbm2ddl.auto* property értéke határozza meg, hogy mi történjen az adatbázis séma létrehozásakor. Lehetséges értékei:

- validate – validálja a sémát, nem okoz adatbázisbeli változtatást
- update – a meglévő séma frissítésre kerül

- create – a megléső séma törlésre kerül, ezután létrehozza az új sémát
- create-drop – létrehozza az új sémát, majd a kapcsolat lezárásakor eldobja azt. Teszteléshez alkalmazzák leginkább.

```
<property name="hibernate.dialect"
    value="org.hibernate.dialect.MySQLDialect" />
<property name="hibernate.show_sql" value="true" />
<property name="hibernate.hbm2ddl.auto" value="update"/>
<property name="hibernate.default_schema" value="gyerekbank"/>
<property name="hibernate.transaction.manager_lookup_class"
    value="org.apache.openejb.hibernate.TransactionManagerLookup" />
```

4.2.3 kód: Adatbázis séma létrehozása

Ahogyan azt a mellékelt kódrészlet is mutatja a meglévő séma frissítését választottam, így anélkül tudom változtatni az entitás osztályokat, hogy az adatbázisban lévő rekordok elvesznének. A persistence.xml a @Entity-vel ellátott osztályokból képezi le az adatbázissémát. Minden @Entity osztályban kötelező lennie @Id annotációval ellátott mezőnek. Lehetőség van Id automatikus generálására a @GeneratedValue(strategy = GenerationType.AUTO) használatával. @Table annotációval a táblával kapcsolatos egyediségek adhatóak meg, például a @Table(name = „accounthistory”) - val megadható, hogy a tábla neve „accounthistory” legyen. Ha nem használunk @Table annotációt a tábla nevének megadására, úgy a Persistence Provider feltételezi, hogy a tábla neve megegyezik az osztály nevével. Ugyanez igaz az attribútumokra is. Attribútumok esetén @Column annotációval adható meg az alapbeállítástól való eltérés, például @Column (nullable = false, unique = true) annotációval ellátott attribútumnak egyedinek kell lennie emellett nem lehet null az értéke. A táblák kapcsolatainak kialakításához a @ManyToOne, @OneToMany és @OneToOne annotációk használhatóak.

4.2.5 Rest

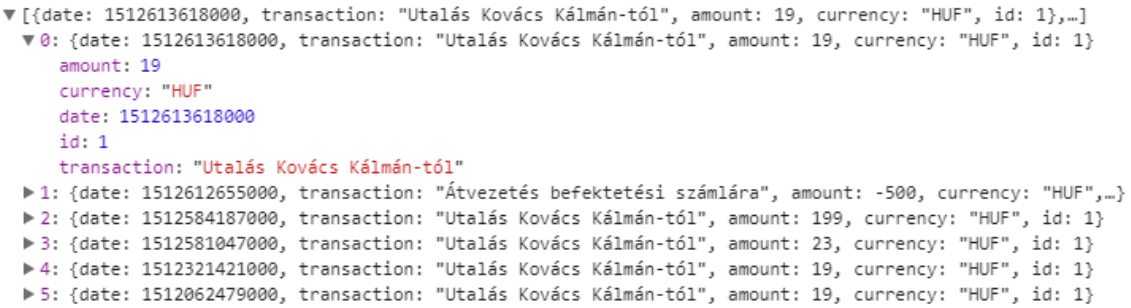
A Gyerekbank alkalmazás backendje és frontendje REST hívásokon keresztül kommunikál egymással. Ebben a package-ben a kommunikációhoz szükséges REST hívások vannak megvalósítva. Minden elemhez külön Rest osztály került létrehozásra Entitásnév+Rs névkonvencióval ellátva.

```
@POST
@Path("/list")
@Produces(MediaType.APPLICATION_JSON)
```

```
public Response getAccountHistory(AccountHistoryDTO dto) {
   List<AccountHistory> accountHistory =
      accountHistoryBean.listAccountHistory(dto.getID(),
          dto.getDate(), dto.getMaxDate());
    return
      Response.status(Response.Status.OK).entity(accountHistory).build();
}
```

4.2.4 kód: Számlatörténet lekérdezéséhez használt rest hívás

A fenti kódrészlet egy általam használt tipikus rest hívás backend oldali részét mutatja be. Először definiálásra kerül a hívás típusa a @POST, @GET, @PUT, @DELETE kulcsszavakkal, ez megadja, hogy a böngésző milyen HTTP típusú kéréssel tudja elérni az erőforrást. Az osztálynak és a restnek is a @Path annotációval adhatjuk meg az eléréshez szükséges URL darabot. A rest hívás mindig az osztály elérési útvonalán keresztül érhető el hozzátéve a konkrét hívás URL-jét is. A válasz Multipurpuse Internet Mail Extension (MIME) típusát a @Produces annotációban kell megadni. Az általam használt rest hívások válaszai mindig JavaScript Object Notation (JSON) formátumban kerülnek átadásra a frontend felé. A JSON formátumban küldött adatok az ember által is olvasható formátumban jelennek meg. Ez nem jelent biztonsági kockázatot az alkalmazásomban, mivel a REST hívásokhoz is jelszóval lehet hozzáférni, így kívülről nem elérhetőek. A beérkező kéréseknél mindig történik ellenőrzés, hogy ahhoz a felhasználóhoz vagy gyerekéhez tartozik e az azonosító, mint aki be van jelentkezve. Amennyiben nem, úgy a kérés nem kerül végrehajtásra. Ezáltal védve van az alkalmazásom az adatlopástól és illetéktelen adatmódosítástól. A 4.2.1-es ábrán látható a *getAccountHistory(...)* válasza JSON formátumban.



4.2.1 ábra: Számlatörténet JSON formátumban

A fent említett kérés az osztálynak megfelelő DTO-ban, jelen esetben accountHistoryDTO-ban kapja meg a frontendtől a paramétereket. Visszatérési értéke egy accountHistory lista, amit az accountHistoryBean *listAccountHistory(...)* függvényéből kapott vissza.

4.2.6 WEB

A web package-ben van megvalósítva az alkalmazás fontendje. Minden funkcióhoz tartozik egy képernyő, melyek külön .html fájlban találhatóak. Ezen kívül mind a szülő, mind a gyerek felhasználónak van egy keretrendszere megvalósítva, ahová a meghívott funkciónak megfelelő képernyő kerül includolásra. Keretrendszer létrehozására azért volt szükség, hogy az állandó elemeket ne kelljen duplikálni minden funkcióhoz, és a közös funkciókat sem a gyerek és a szülő felhasználóhoz. A képernyők működéséhez tartozó logikák az app.js fájlban találhatóak. Minden képernyőnek saját kontrollere van ahol a működéshez szükséges függvények kerültek megvalósításra. Egy-egy rest osztály hívásai a .factory-ban vannak definiálva, melyet fel tudnak használni a megfelelő controllerek. Listák fogadásához isArray: true-val kell megadnunk a factory-ban, hogy a hívás egy listával tér vissza.

Bejelentkezés után megvizsgálásra kerül a felhasználó típusa és az annak megfelelő keretrendszer lesz includolva az index.html fájlba. A keretrendszer betöltésekor mindkét felhasználó típus esetén automatikusan betöltésre kerül az adatok megtekintése funkcióhoz tartozó képernyő. A menügombokra kattintva a képernyő középső részébe betöltődik a menühöz tartozó képernyő elem. A HTML fájt betöltésekor egyből meghívódik a kontrollere, mely a rest hívás válaszából visszaadja számára a megjelenítendő adatokat.

A szám formátumú mezőknél korlátoztam a mező értékkészletét számokra annak érdekében, hogy a felhasználó más karaktert ne tudjon oda beírni, ezáltal csak valid adat kerül át a backendhez. Az éves kamat % és elő- és végtörlesztési díj mezőkbe nem csak egész számokat lehet beírni, itt a *$watch*(…) és a *parseFloat(...)* függvényeket használtam. Az alábbi kódrészlet a használatát mutatja be. A newLoan.interest attribútumot új hitel felvételekor használom a kamat megadására. Amennyiben az értéke nem felel meg egy float típusú számnak, úgy kicserélésre kerül a float típusra konvertált értékével.

```
$scope.$watch ('newLoan.interest', function (newValue) {
    if(newValue != parseFloat(newValue))
       $scope.newLoan.interest = parseFloat(newValue)
});
```

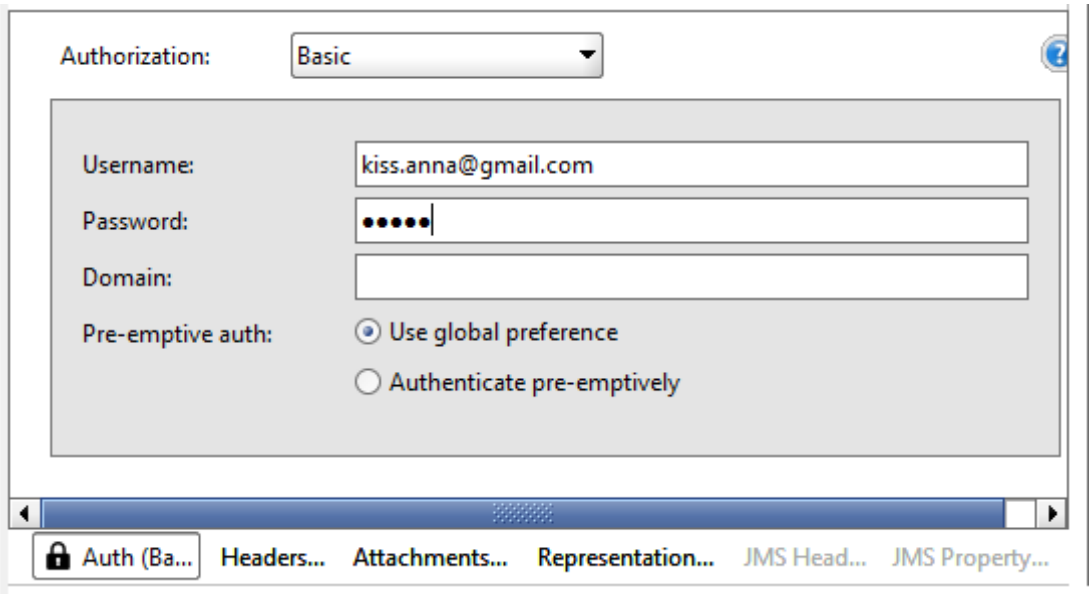
4.2.5 kód: Csak float típusú számok beírásának engedélyezése

Az alkalmazás felületének létrehozásakor felhasználtam a Bootstrap által kihasználható cellás elrendezést. Ezáltal nem csak a három fő elrendezést: balra-, jobb- és középre igazítva, illetve az egymás mellett elhelyezkedő elemekre tudtam építeni, hanem egyedi elhelyezéseket is létrehozhattam.

5 Tesztelés

A tesztelés egy program fejlesztése során ugyanolyan fontos, mint maga a tervezés vagy a fejlesztés. Minél később derül ki egy hiba, annál költségesebb a javítása. Ennek érdekében a programom fejlesztésekor folyamatosan teszteltem az addig létrehozott funkciókat.

Az elkészült weblalkalmazásom teszteléséhez a SoapUI-t választottam, mely segítségével egyszerűen tesztelhetőek a webszolgáltatások. A webalkalmazásom frontendje Rest hívásokon keresztül kommunikál a backenddel, ennek megfelelően teszteléséhez először létrehoztam egy új Rest projeck-et, amelynek meg kellett adnom a tesztelni kívánt hívás elérési útját. Rest hívásaim Basic típusú authentikációval vannak védve, ennek megfelelően be kell állítanom egy ilyen típusú authentikációt a SoapUI-ban is.

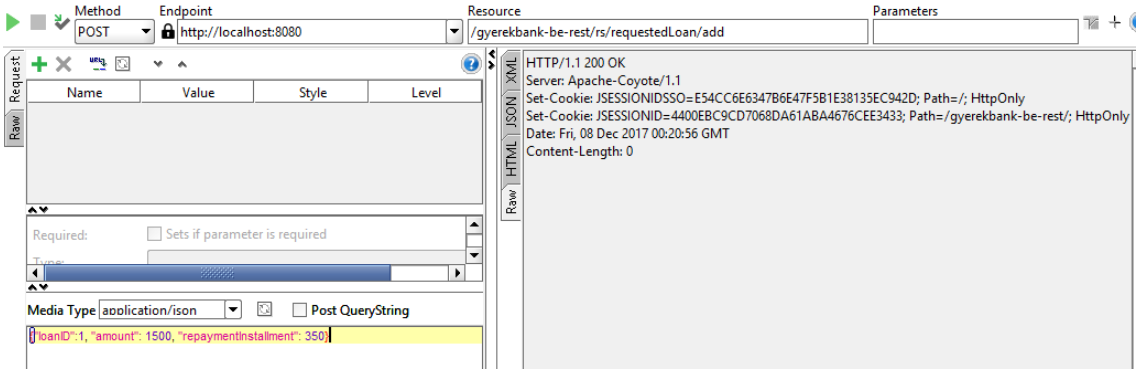


5.1.1 ábra: Basic authentikáció SoapUI-ban

Bejelentkezni csak az adatbázis Person táblájában található felhasználókkal lehet. Egy hiteligénylési folyamatot szeretnék tesztelni, ehhez Kiss Anna, gyerek adatait adom meg az azonosítási folyamathoz. Hitel igényléséhez egy új RequestedLoan rekordot kell létrehozni az adatbázisban, ehhez egy POST hívás szolgál. A hívás paraméterében megadok egy Kiss Annához hozzárendelt hitel azonosítóját, 1500 Ft felvenni kívánt hitelt, melynek a törlesztőrészletét 350 Ft-ra állítottam be. A hívás

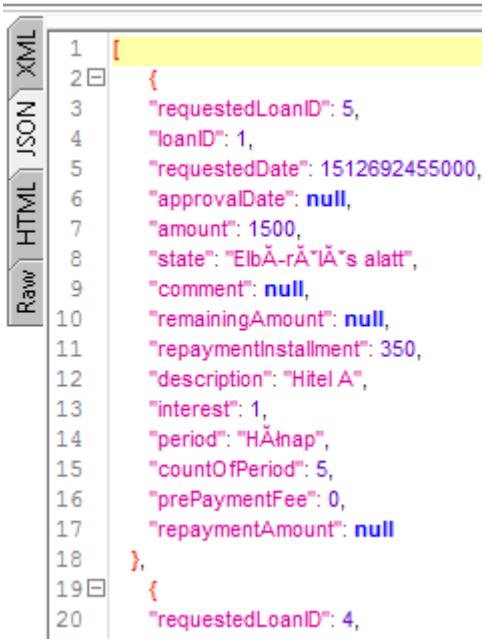
indítása után az elvárt kimenet egy OK státuszú válaszüzenet lenne, mely jelzi, hogy sikeresen megigénylésre került Kiss Anna számára a hitelt.

Az 5.1.2-es ábrán látható, hogy a hitel felvétel sikeres volt, 200-as OK üzenettel tért vissza a REST.



5.1.2 ábra: Hiteligénylés SoapUI-ban

Ellenőrzésképpen lekérdeztem a Kiss Annához tartozó igényelt hiteleket, ahonnan első helyről kaptam vissza az Elbírálás alatti státuszban lévő felvett hiteligénylét.

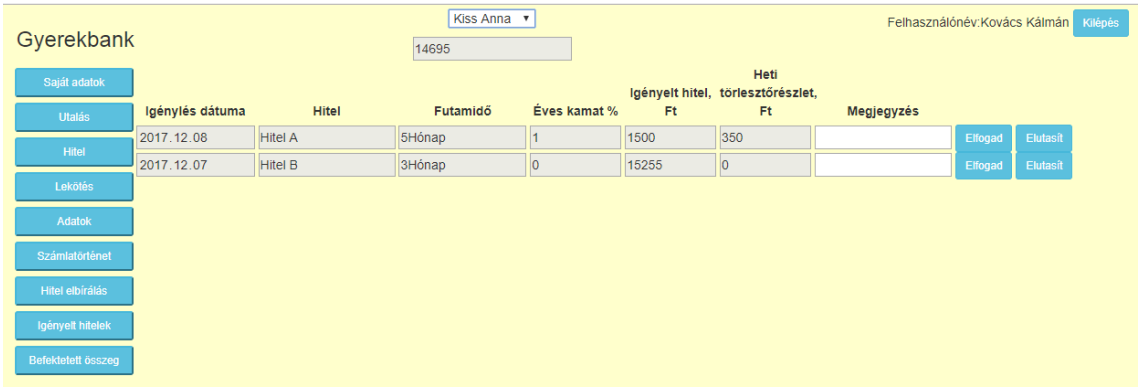


5.1.3 ábra: Kiss Annához tartozó igényelt hitel

SoapUI-al való tesztelésem során megállapítottam, hogy a Rest hívások az előírtnak megfelelően működnek. Ezáltal elmondható, hogy a hívásokhoz tartozó üzleti logikák jól vannak megírva, ahogyan az adatbázis lekérdező és módosító funkciók is.

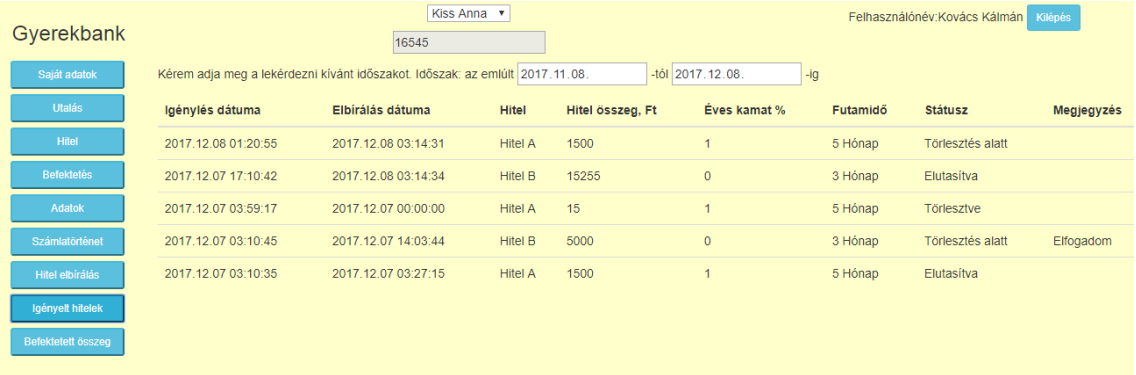
Következő lépésben a hiteligénylés folytatásaként a megjelenítést tesztelem a szülő általi hitel elbírálás folyamatával.

Kiss Annához hozzárendelt felhasználónál meg kell jelennie egy új hitel elbírálás sornak a most felvett hiteligényléssel a Hitel elbírálás menüpontban, amennyiben Kiss Anna a kiválasztott gyerek.



5.1.4 ábra Hitel elbírálás

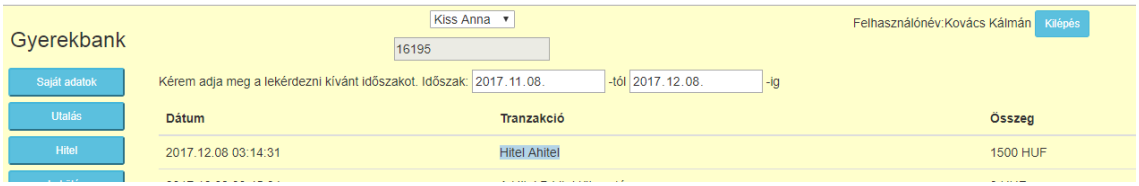
A vártnak megfelelően látható az elbírálásra váró hitelek között az új hiteligénylés is. Elutasítás esetén az igényelt hitel státusza elutasított lesz. Ebből a státuszból nem lehet tovább lépni. Elfogadás esetén az igényelt hitel státusza Elfogadott lesz, az összeg átutalásának megkezdésére azonnal sor kerül. Ebben az esetben mindkét szülő egyenlegének módosulnia kellene a hitel összegével, a szülőnek negatív, a gyereknek pedig pozitív irányba. Amellett a számlatörténetben meg kell, hogy jelenjen a tranzakció is. Mivel úgy is két hitelem van, így az elsőt elfogadom, míg a második elutasításra kerül.



5.1.5 ábra: Igényelt hitelek listája

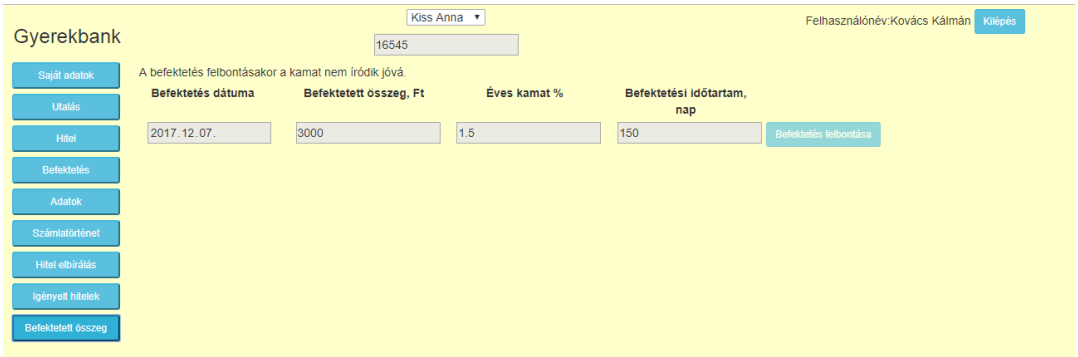
Ahogyan az 5.1.5-ös ábrán is látható a gyerek egyenlege 1500 Ft-al nőtt az elfogadott hitel összegének megfelelően. Az első, december 8-án igényelt hitel státusza

Elfogadva, míg a december 7-én igényelt B hitelének státusza *Elutasítva*. Számlatörténetét megtekintve látható, hogy megjelent a tranzakció is a listába.



5.1.6 ábra: Számlatörténet részlet

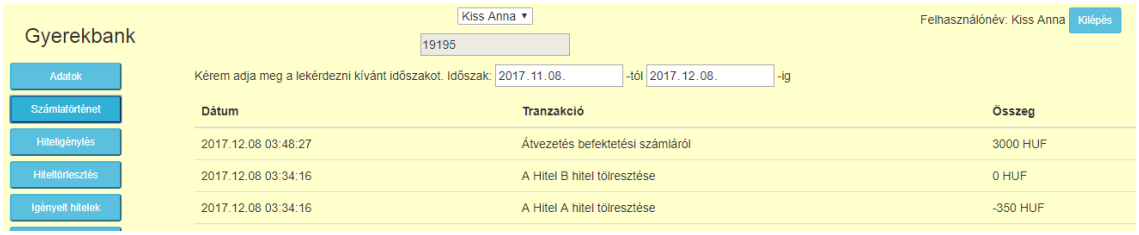
A következő dolog, amit érdemes lehet letesztelni, az a befektetés felbontása. Mind a szülő, mind a gyerek láthatja a Befektetett összegek menüpontban a gyerek befektetéseit, de felbontani azt csak a gyereknek van joga. A szülőnél a befektetés felbontása gombnak inaktívnak kell lennie.



5.1.7 ábra: Inaktív befektetés felbontása a szülőnél

Gyerek felhasználó esetén befektetés felbontása esetén vissza kell, hogy kerüljön a befektetett összeg a gyerek számlájára a kamattal együtt amennyiben van. Kamat csak akkor íródik jóvá, ha elérte a gyerek a minimum befektetési napot a befektetés során.

Mivel december 7-én lett befektetve 3000 Ft 150 napra, de minimum befektetési idő nincs megadva a befektetési tételhez, így 8-án felbontva egy napi kamatot jó kell, hogy írjanak, ami 0,12 F, kerekítve 0 Ft, azaz kamat felbontásakor csak 3000 Ft-ot kell, hogy jóváírjanak a számláján.



5.1.8 ábra: Átvezetés a befektetési számláról

Gyerekbank alkalmazás bank - szülő - gyerek viszonylatban: A bank biztosítja a szülőnek és a gyereknek az alkalmazás használatát. Az alkalmazás használatához minden esetben rendelkeznie kell a felhasználónak egy bankszámlával. Kiskorú esetén a számlanyitáshoz egy törvényes gyámra van szükség, aki rendelkezik a számla fölött. Ő tudja intézni a számlával kapcsolatos ügyeket, pénz tud felvenni róla, stb. A számlával kapcsolatos minden tranzakcióért jogilag ő a felelős, melyet aláírásával elfogadott a számla megnyitásakor. A Gyerekbank alkalmazásban a gyerek csak a szülővel áll kapcsolatban, senki mással nem. Minden műveletnél ellenőrzésre kerül, hogy elvégezheti e azt a felhasználó, azaz kapcsolódik e valamilyen módon ahhoz az ID-hoz mellyel dolgozni szeretne – legyen az akár egy hitel felvétel. Ennek köszönhetően mind a szülők, mind a gyerekek védve vannak a nemkívánatos műveletektől az alkalmazásban.

6 Összefoglalás

Szakdolgozatom témája egy tanító jellegű Gyerekbank webalkalmazás megvalósítása. Röviden bemutattam létező gyerekbank megoldásokat Magyarországról és külföldről. Ezek után ismertettem a saját gyerekbank alkalmazásom követelményeit, bemutattam, hogyan kapcsolódhat egy banki szoftverarchitektúrába.

A frontend fejlesztéséhez az Angular JS és a Bootstrap keretrendszert használtam valamint a JavaScript programozási nyelvet választottam. A backend megvalósítása JAVA EE platformon történt, adatbáziskezelőnek pedig a MySQL adatbázist használtam. A webes alkalmazások és szolgáltatások kiszolgálásához az Apache Tomcat Java Enterprise Edition részeként választásom a TomEE webszerverre esett. A forráskód menedzselését és a fordítást a Maven keretrendszer biztosítja az alkalmazásomban.

Ezek után bemutatom a szülő és a gyerek felhasználók által elérhető funkciókat és a rendszer mögötti adatbázis struktúrát. Az alkalmazás által támogatott hitelfelvétel, megtakarítások, egyenleglekérdezés és valamennyi kapcsolódó tevékenység támogatott folyamata is ismertetésre került.

Ismertetem a megvalósításhoz szükséges frontend webosztályokat, a REST hívásokat, az üzleti logika és az entitások osztályait valamint az adatbázis műveletek DAO osztályait és az adatátviteli objektumokat (DTO).

Munkámat teszteléssel zártam, ahol az alkalmazás működését ellenőriztem valamennyi folyamat végig vitelével. A dolgozatban a hitelfelvétel, elbírálás és betétfelbontás folyamatok tesztelését mutattam be részletesen.

7 Továbbfejlesztési lehetőségek

A webalkalmazásnak számos továbbfejlesztési lehetősége van, többek között:

- szülő felhasználónál a tétel kiírás csoportba tartozó menüpontokhoz külön képernyő létrehozása többgyerekes szülők számára annak érdekében, hogy egyszerre tudjanak gyerekeiknek tételeket kiírni. Többgyerekes szülőknél egy lenyíló listából lehetne választani, hogy melyik gyerekének szeretne tételt kiírni, legalább egy gyermek kiválasztása kötelező. Így ugyanazt a tételt a szülőknek nem kell többször megadniuk.
- értesítő email küldése szülő számára hitel igényléskor, gyerek számára pedig a hitel elbírálásakor.
- a funkciók tanító jellegű bővítése: a kifejezésekhez magyarázatok, példák hozzáadása. Például mit jelent a kamat, a THM; hogyan kell jól hitelt választani; mikor érdemes lekötni pénzünket.
- többnyelvűsíteni is lehetne az alkalmazást. A gyerekek így megismerhetnék a banki kifejezéseket más nyelveken is, emellett más anyanyelvű személyek is tudnák használni.
- reszponzív mobil applikáció létrehozása elérhetővé tenné az alkalmazás kényelmes felhasználását mobil vagy tablet felületről.

Irodalomjegyzék

[1] ASB Bank Gyerekbank számla információk, https://www.asb.co.nz/bank-accounts/headstart.html (megtekintés 2017. december 7.)

[2] Standard Bank Gyerekbankszámla információk, https://www.standardbank.co.za/standardbank/Personal/Banking/Current-accounts/(sum)1 (megtekintés 2017. december 7.)

[3] ANZ bank gyerekbank számla információk, https://www.anz.com.au/personal/bank-accounts/kids-banking/ (megtekintés 2017. december 7.)

[4] MyKidsBank oktató banki alkalmazás - http://mykidsbank.org/about.htm (megtekintés 2017 december 7.)

[5] Bankaroo pénzügyi oktató szoftver - https://www.bankaroo.com/#mobile-apps (megtekintés: 2017. december 7.)

[6] Wikipédia: *Java Platform, Enterprise Edition*, https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition (megtekintés: 2017. november 22.)

[7] Balogh P., Berény Zs., Dévai I., Imre G., Soós I., Tóthfalussy B.: *Szoftverfejlesztés Java EE platformon*, ISBN 987-963-9131-97-2, Szak Kiadó, 2007

[8] Oracle: *Building and Running a Java EE Application by Using Maven*, http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Maven_EE/MavenEE.html, (megtekintés: 2017.december 6.)

[9] AngularJS, https://angularjs.org/ megtekintés (2017. december 7)

[10] Apache Software Foundation: *Apache Tomcat 8*, http://tomcat.apache.org/tomcat-8.0-doc/index.html, megtekintés(2017. december 6.)

[11] Wikipédia: *MySQL*, https://en.wikipedia.org/wiki/MySQL (megtekintés: 2017. november 29.)

[12] Welling L., Thomson L.: *PHP and MySQL Web Development*, 4th edition, ISBN 987-963-9929-13-5, Addison-Wesley, 2009

[13] Wikipédia: *SoapUI*, https://en.wikipedia.org/wiki/SoapUI (megtekintés: 2017. december 3.)

[14] Wikipédia: *XAMPP*, https://hu.wikipedia.org/wiki/XAMPP (megtekintés: 2017. november 22.)

[15] Vaadin: https://vaadin.com/? megtekintés(2017. december 7.)

[16] Spring https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc, megtekintés(2017. december 7)

[17] Node.js https://nodejs.org/en/docs/ megtekintés (2017. december 7)

[18] SoapUI: *Soapvs rest 101: Understand the deffirences*, https://www.soapui.org/testing-dojo/world-of-api-testing/soap-vs-rest-challenges.html

[19] Wikipédia: *Annuitás (pénzügy),* https://hu.wikipedia.org/wiki/Annuit%C3%A1s_(p%C3%A9nz%C3%BCgy) (megtekintés: 2017. december 4)

Függelék

Standard Bank – kisállat segítők[2]

A név alapján kiválasztott gyerek ID-ját adja vissza a szülő felületen

```
$scope.addChildID = function (childName) {
    for (const index in $scope.child) {
        if ($scope.child[index].name === childName) {
            actualChildID = $scope.child[index].id;
            $rootScope.$broadcast("actualChildIDChanged",
actualChildID)
        }
    }
}
```

actualChildID változásakor újratöltődik a számlatörténet oldal

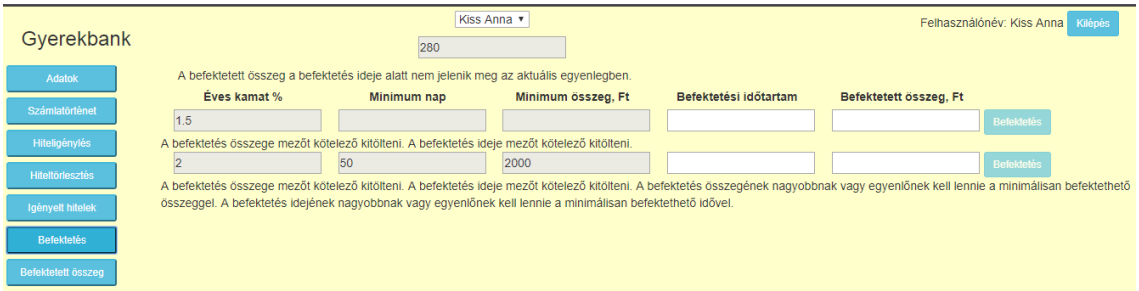
```
$scope.$on('actualChildIDChanged', function () {
    $scope.getHistory($scope.minDay, $scope.maxDay)
});
```

Igényelt hitelek

Hiteltörlesztés

Befektetett összeg

Befektetés



Számlatörténet lekérdezés frontend oldalról

```
$scope.getHistory = function(minDay, maxDay){

        if (minDay != null){
            minDay.setHours(0,0,0);
        }

        if (maxDay != null){
            maxDay.setHours(23,59,59);
        }
        accountHistoryData.getAccountHistory(
            {
                id: actualChildID,
                date: minDay,
                maxDate: maxDay
            },
            function (resp) {
                $scope.accountHistorys = resp;
            })
```

```
    }

    $scope.getHistory($scope.minDay, $scope.maxDay);

}])
```

Kijelentkezés

```
$scope.logout = function () {

    personData.logOut(
        {},
        function () {
            $window.location.href = '/gyerekbank-fe-web/index.html';
        }
    )
};
```

.factory

```
.factory("childData", ['$resource', function ($resource) {
    var url = "http://localhost:8080/gyerekbank-be-rest/rs/child/";
    return $resource(url,{},
        {
            "getChild": {url: "http://localhost:8080/gyerekbank-be-
rest/rs/child/listChild",
                method: 'POST', isArray: true}
        });
}]);
```

Igényelt hitelek view megvalósítás

```
<div ng-controller="requestedLoanController">
    <div class="container normalmargin">
        <p align="left">
            Kérem adja meg a lekérdezni kívánt időszakot. Időszak: az
emlúlt
            <input type="date" ng-model="minDate" ng-
keypress="getRequestedLoan(minDate, maxDate)"> -tól
            <input type="date" ng-model="maxDate" ng-
keypress="getRequestedLoan(minDate, maxDate)"> -ig
        </p>

        <table class="table" disabled>
            <thead>
            <tr>
                <th>Igénylés dátuma</th>
                <th>Elbírálás dátuma</th>
                <th>Hitel</th>
                <th>Hitel összeg, Ft</th>
                <th>Éves kamat %</th>
                <th>Futamidő</th>
                <th>Státusz</th>
                <th>Megjegyzés</th>
            </tr>
            </thead>
            <tbody>
            <tr ng-repeat="requestedLoan in requestedLoans">
                <td>{{requestedLoan.requestedDate | date: "yyyy.MM.dd
HH:mm:ss"}}</td>
                <td>{{requestedLoan.approvalDate | date: "yyyy.MM.dd
HH:mm:ss"}}</td>
```

```
                <td>{{requestedLoan.description}}</td>
                <td>{{requestedLoan.amount}}</td>
                <td>{{requestedLoan.interest}}</td>
                <td>{{requestedLoan.countOfPeriod}}
{{requestedLoan.period}}</td>
                <td>{{requestedLoan.state}}</td>
                <td>{{requestedLoan.comment}}</td>
            </tr>
            </tbody>
        </table>
    </div>
</div>
```

Igényelt hitelek megjelenítése

```
public List<RequestedLoan> getRequestedLoanWithMinAndMaxDate(Integer
loanID, Date minDate, Date maxDate) {
    CriteriaBuilder cb = getEm().getCriteriaBuilder();
    CriteriaQuery<RequestedLoan> cq =
cb.createQuery(RequestedLoan.class);
    Root<RequestedLoan> rootEntry = cq.from(RequestedLoan.class);
    cq.where(cb.equal(rootEntry.get("loanID"), loanID),

cb.or(cb.greaterThanOrEqualTo(rootEntry.<Date>get("requestedDate"),
minDate)),
            cb.lessThanOrEqualTo(rootEntry.<Date>get("requestedDate"),
maxDate));
    cq.orderBy(cb.desc(rootEntry.get("requestedDate")));
    CriteriaQuery<RequestedLoan> all = cq.select(rootEntry);
    TypedQuery<RequestedLoan> allQuery = getEm().createQuery(all);
    return allQuery.getResultList();
}
```

Timer

```
@Timeout
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
public void run() {
    requestedLoanBean.requestedLoan();
    transactionBean.transaction();
    investmentAmountBean.newInvestmentAmount();
}

@PostConstruct
public void init() {

    TimerConfig config = new TimerConfig();
    config.setPersistent(false);
    timer = timerService.createIntervalTimer(01, 24 * 36000001,
config); //szám?
}
```

Befektetés felbontásakor a kamat számítása

```
Date date = new Date();
long howMuchDay = (date.getTime() -
investmentAmount.getInvestmentDate().getTime())/(24*60*60*1000);
Integer balance = 0;

if
(investmentDAO.findById(investmentAmount.getInvestmentID()).getMinDay(
) != null){
```

```
    if (howMuchDay <
investmentDAO.findById(investmentAmount.getInvestmentID()).getMinDay()
) {
        balance = 0;
    }
    else {
        balance = (int) round(investmentAmount.getAmount() *
howMuchDay *

((investmentDAO.findById(investmentAmount.getInvestmentID()).getIntere
st() / 365)/100));
    }
}
else if
(investmentDAO.findById(investmentAmount.getInvestmentID()).getMinDay(
) == null) {
    balance = (int) round(investmentAmount.getAmount() * howMuchDay *

((investmentDAO.findById(investmentAmount.getInvestmentID()).getIntere
st() / 365) / 100));
}
```