Szókártya alkalmazás

Szoftverarchitektúrák házi feladat

2015 ősz

Dokumentáció

Készítették:

Gyuris Bence Zsolt (MFSX9D)

Olasz-Szabó Bence (Q95A6S)

Konzulens:

Nagy Ákos

Tartalomjegyzék

1. Feladatleírás

A szókártyák az idegen nyelvi szókincs bővítésének hatékony és játékos módszerei. A feladat egy olyan alkalmazás megvalósítása, ami ilyen módon segíti a nyelvtanulást.
Megvalósítandó feladatok:

- Felhasználókezelés
- Szókártyák készítése
- Szókártyák tematikus rendezése
- Megosztás közösségi oldalakon
- Teszt
- Egyéb, megbeszélés szerinti feladatok

Felhasználandó technológiák:

- A szerveroldali komponensek megvalósítása a vonatkozó Microsoft technológiákkal (.NET, Entity Framework, WCF/Web Api stb.)
- A kliensoldal megvalósítása Windows 8/8.1, Windows Phone 8/8.1 vagy Android alkalmazásként

2. Követelményspecifikáció

2.1. Funkciók

Az alkalmazás a következő funkciókat fogja megvalósítani.

2.1.1. Felhasználókezelés

Az alkalmazásnak lehetővé kell tennie, hogy a felhasználók fiókokat hozhassanak létre. Ez a következő alfunkciókat foglalja magában: regisztráció, bejelentkezés, kijelentkezés.

Regisztráció: Első használatkor új fiók létrehozása, e-mail, jelszó és felhasználói név megadásával.

Bejelentkezés: E-mail és jelszó megadásával történő bejelentkezés.

Kijelentkezés: Fiókból való kilépés, egyéb felhasználói fiókba való bejelentkezést teszi lehetővé.

Az alkalmazás minden egyéb funkciójához szükséges, hogy a felhasználó bejelentkezve legyen a fiókjába. Ehhez aktív internet kapcsolat kell, így az alkalmazás offline működést nem tud megvalósítani.

2.1.2. Szókártyák készítése

Minden felhasználó létrehozhatja a saját szókártyáit. Egy szókártyának két oldala van, létrehozáskor a felhasználó ezt a két oldalt adja meg a megfelelő létrehozó oldalon. A kártya eleje az amely alapján ki kell találni a hátulját. A kártya hátulja egy idegen nyelvi szó (szöveg), az eleje lehet magyar nyelvű szöveg vagy kép. A kártyák hátulján lévő szó nyelvét meg kell adni (angol, francia, német stb.). A kártyák törölhetőek és módosíthatóak is.

2.1.3. Szókártyák tematikus rendezése

A felhasználók a saját kártyáikat kategóriákba (=csoportokba) rendezhetik. Létrehozhatnak névvel ellátott kategóriákat a megfelelő menüpont kiválasztásával. Minden kártyának megadható kategória létrehozáskor, később ez módosítható. A kategóriának megadható

nyelv, ez a nyelv lesz az alapértelmezett a kategória kártyáin. A kategóriák törölhetőek és módosíthatóak is.

2.1.4. Szókártyák megosztása

A felhasználók elérhetik egymás szókártya kategóriáit (csoportjait). A kategóriák lehetnek publikusak és privátak is. A kategóriákat meg lehet osztani más felhasználókkal közösségi oldalon keresztül (Facebook). A felhasználó ki tud tenni egy hivatkozást a közösségi oldalra, amely egy kategóriára mutat. Aki ezt a hivatkozást megnyitja, az később elérheti a kategória szókártyáit (a kártyák meg lesznek osztva vele). A hivatkozás létrehozásának és használatának technikai részleteit később definiáljuk.

Tehát, egy felhasználó elérheti azokat a szókártyákat/kategóriákat, amelyek:

- publikusak vagy
- megosztva vannak vele (pl. hivatkozásra kattintáson keresztül) vagy
- ő hozta létre őket (sajátjaik).

2.1.5. Teszt

Az alkalmazás lényege a nyelvtanulás (szókincs bővítés), ezért a legfontosabb funkció a nyelvi tesztelés. Bejelentkezés után elindítható többféle teszt, amelyben kártyák sorozatát kapjuk, ezek előlapjából kell kitalálni a kártyák hátlapját (kapunk egy képet/szót, ez alapján kell megadni a megfelelő idegen nyelvű szót).

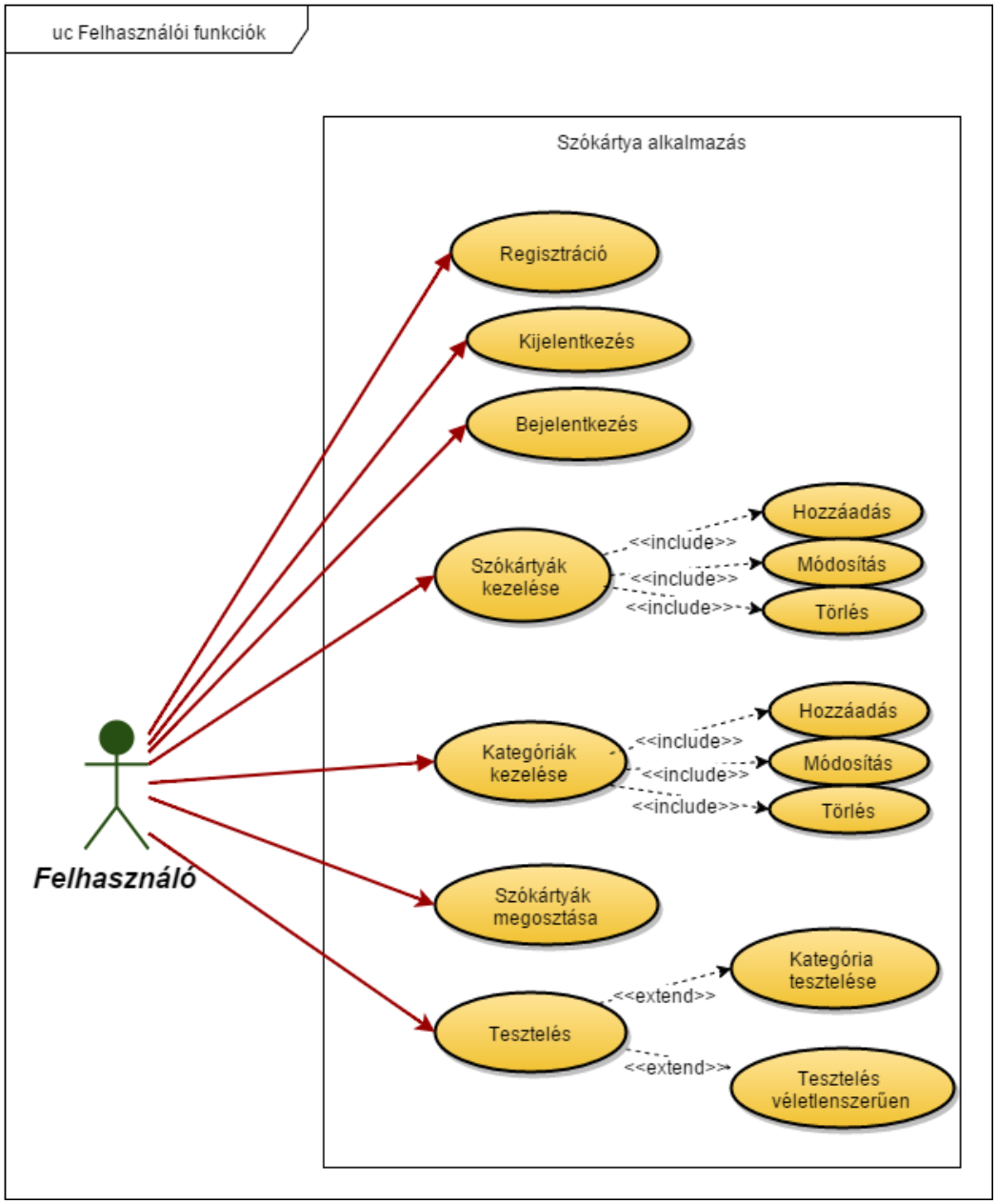
A kártyasorozat attól függ, hogy milyen típusú tesztet indítottunk el. A következők közül választhatunk:

- **Véletlenszerű teszt nyelv alapján**: Bármilyen kártyát kaphatunk a sorozatban (egyet maximum egyszer), amely a megadott nyelvhez tartozik. A teszt addig tart amíg a felhasználó le nem állítja, vagy el nem fogynak a kártyák a rendszerből.
- **Teszt egy kategórián belül**: A kártyák egy kiválasztott kategóriából fognak érkezni, addig tart a teszt, amíg minden kártyán végig nem érünk a kategóriából vagy a felhasználó le nem állítja. A sorrend véletlenszerű.

A befejeződik a teszt (vége a sorozatnak vagy a felhasználó megszakítja) az alkalmazás kiírja a felhasználó eredményét (hány szót talált el).

2.1.6. Használati esetek

A felhasználói funkciókat a következő Use-case diagram foglalja össze:



2.2. Nem funkcionális követelmények

Az alkalmazásnak egyéb, nem funkcionális követelményeket is meg kell valósítania.

2.2.1. Naplózás

Az alkalmazásnak naplóznia kell a fontosabb eseményeket: hibák, leállások, bizonyos műveletek (pl. regisztráció, bejelentkezés, kártya feltöltés, törlés, stb.), rendszeresemények. A naplózást meg kell valósítani mind szerver, mind kliens oldalon. Ennek a célja, hogy a későbbi hibák oka kideríthető legyen, és a javítás könnyebb (lehetséges) legyen.

2.2.2. Biztonság

A felhasználó kliens programa és a szerver közötti kommunikáció megfelelő szintű titkosítással kell hogy legyen ellátva (HTTPS), más ne legyen képes elolvasni az üzeneteket.
Legyen megfelelő a szerver oldali jogosultságkezelés: a felhasználó csak a saját szókártyáit és csoportjait tudja módosítani, csak a saját nevében tudjon létrehozni ilyeneket, csak a sajátjait tudja megosztani, más felhasználók adataihoz ne férjen hozzá.
Legyen megfelelő hitelesítés (authentication): A felhasználónak egyszer kell bejelentkeznie (első használatkor), utána mindig az ő nevében fog történni a kommunikáció (hitelesítő token szükséges).

2.2.3. Teljesítmény

Az alkalmazás felé nincsenek külön teljesítmény igények, a kliens és szerver programok best effort szolgáltatást nyújtanak a minél jobb felhasználói élmény érdekében.

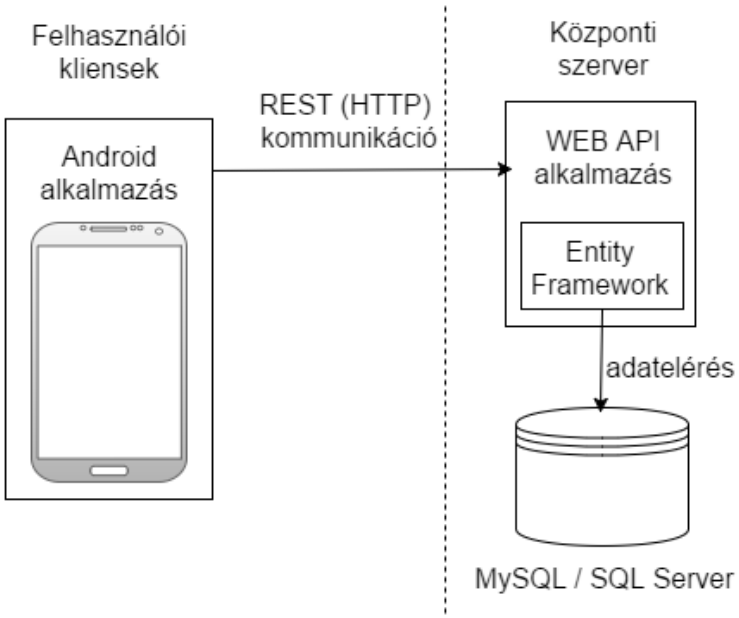
3. Rendszerterv

Az alkalmazás egy kliens alkalmazásból egy szerveralkalmazásból és az adatok tárolásáért felelős adatbázisból fog állni.

3.1. Architektúra

3.1.1. Rendszer ábra

A megvalósítandó szoftverrendszer komponenseinek áttekintése a következő ábrán látható:



3.1.2. Kliens oldal

A kliens oldalt egy Android alapú mobilalkalmazás valósítja meg. Ez felhasználói input hatására az egyes funkciókhoz tartozó REST kéréseket elküldi HTTPS csatornán keresztül, majd a kapott válaszok (Json/XML) alapján megjeleníti a felhasználónak a szükséges tartalmat.

3.1.3. Szerver oldal

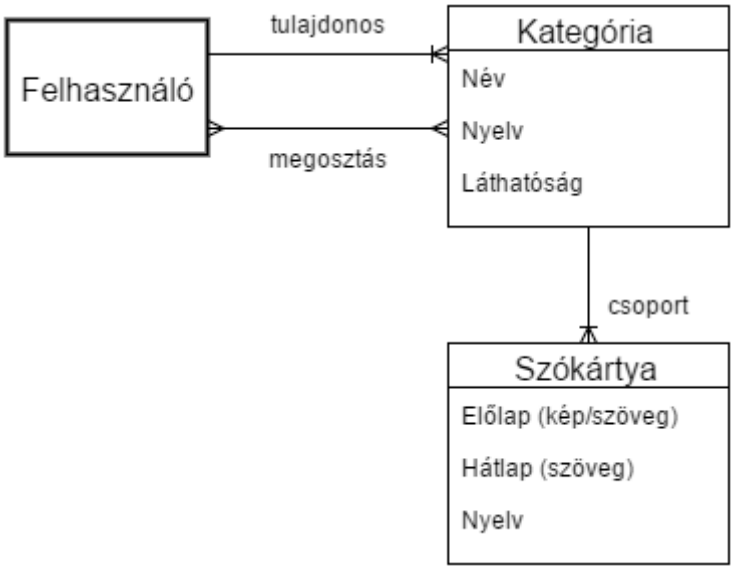
A szerver oldali logikát egy IIS szerveren futó WEB API alkalmazás fogja megvalósítani (C# nyelven kódolva). A perzisztens adatok tárolásáért egy adatbázis szerver lesz felelős. Ezt meg lehet valósítani SQL Server-rel (Microsoft) és MySQL adatbázis szerverrel is (MySQL előny: ingyen létrehozható Azure-ban hallgatói licence-szel, SQL Server előny: egyszerűbb integráció). Az akalmazáslogika és az adatbázis közötti kapcsolatot az Entity Framework keretrendszer fogja megvalósítani. A szerver feladata, hogy az egyes funkciókhoz tartozó műveleteket a kliens kérésére végrehajtsa és a válaszukat visszaadja.
Felhasználókezelésre a beépített Individual User Accounts modult fogjuk használni.

3.1.4. Architektúra modell

Az alkalmazás ebből következően a háromrétegű architektúra modelljét fogja követni.
Megjelenítés: Ennek a rétegnek a megvalósítása a kliens alkalmazás feladata.
Üzleti logika: Ezt a réteget a szerveroldali vezérlőkben valósítjuk meg.
Adatelérés: Ezt a réteget az Entity Framework által automatikusan generált leképzés fogja megvalósítani.

3.1.5. Adatbázis séma

A következő ábra áttekinti azt a sémát, amellyel lehetséges lesz a szókártyák és kategóriák eltárolása. A felhasználókezelés itt nem szerepel, mert annak a sémáját a keretrendszer tartalmazza.



3.2. Kommunikáció

Az architektúrában fontos szerepet játszik a szerver és kliens közötti kommunikáció. A kliens felhasználói input hatására az egyes funkciókhoz tartozó REST kéréseket elküldi HTTPS (tehát titkos) csatornán keresztül, majd a kapott válaszok (Json/XML) alapján megjeleníti a felhasználónak a szükséges tartalmat. Ezek alapján definiáltuk az alkalmazás funkcióinak megvalósításához szükséges REST kérések formátumát, valamint a válaszok és paraméterek adatstruktúráit.

3.2.1. Adatstruktúrák

Az adatstruktúrák definiálásánál az üzleti objektumok struktúráit adtuk meg, mivel mind a szerver, mind a kliens oldalon használt technológiák képesek az üzleti objektumok automatikus sorosítására JSON és XML formátumra.

```
struct FlashCard {
        int ID;
        string Front;
        string Back;
        int CategoryID;
        string Language;
}

struct Category {
        int ID;
        string Name;
        string Language;
        boolean IsPublic;
        string UserID;
}

struct User {
        string ID
        string DisplayName
}

struct FlashCards {
        int[] CardIDs
}
```

```
struct RegisterParams {
        string Email
        string DisplayName
        string Password
        string ConfirmPassword
}
struct LoginResponse {
        string access_token
        string token_type
        int expires_in
        string userName
        string .issued
        string .expires
}
```

Megjegyzés: A Language mezők 2 betűs kódokat tartalmaznak az ISO 639-1 codes szabvány szerint.

3.2.2. Webszolgáltatás interfész

A REST API funkciók elérési módját az alábbi táblázat tartalmazza:

HTTP	relatív url	paraméterek	visszatérés	funkció
GET	cards?category={id}	Az {id} a lekérdezett kategória azonosítója.	FlashCards	kategória kártyáinak lekérdezése
GET	cards?rand={num}	A {num} a véletlenszerűen lekért kártyák (maximális) darabszáma	FlashCards	kártyák véletlenszerű lekérdezése kategóriától függetlenül
GET	cards/{id}	Az {id} a kártya azonosítója.	FlashCard	kártya lekérdezése
POST	cards	Body-ban egy FlashCard struktúra.		kártya létrehozása
PUT	cards/{id}	Az {id} a kártya azonosítója. Body-ban egy FlashCard struktúra.		káryta módosítása
DELETE	cards/{id}	Az {id} a kártya azonosítója.		kártya törlése
GET	categories		Category[]	összes látható kategória lekérdezése

HTTP	relatív url	paraméterek	visszatérés	funkció
GET	categories?own=true		Category[]	saját kategóriák lekérdezése
GET	categories/{id}		Category	kat. lekérdezése
POST	categories	Body-ban egy Category struktúra.		kategória létrehozása
PUT	categories/{id}	Az {id} a kategória azonosítója. Body-ban egy Category struktúra.		kategória módosítása
DELETE	categories/{id}	Az {id} a kategória azonosítója.		kategória törlése
GET	categories/{id}?share	Az {id} a kategória azonosítója.	string	Kategória megosztásához kulcs kérése.
PUT	categories/{id}?unlock	Az {id} a kategória azonosítója. Body-ban egy jelszó string-et vár.		Kategória feloldása kulcs segítségével.
GET	users/{id}	Az {id} a felhasználó azonosítója. 0=saját	User	felhasználó lekérdezése
POST	account/register	Body-ban egy RegisterParams struktúra.		regisztráció
POST	token	Body-ban egy bejelentkező formot vár.	Login-Response	bejelentkezés, leírás lejjebb

3.2.3. Autentikáció

Az autentikáció az OAuth2 szabvány használatával zajlik, amely beépíthető az ASP.NET WEB API projektek létrehozásakor. A webszolgáltatás utolsó két művelete a keretrendszer által van megvalósítva. A bejelentkezéshez egy POST üzenetet kell küldeni a /token címre, amely body-jában egy speciális urlencoded form szerepel (grant_type, username, password mezőkkel). Ennek a kérésnek a válaszában egy token található (access_token mező), amely bizonyos ideig használható azonosításra. A kliens ezt a tokent küldi el később a HTTP kérések fejlécében az Authorization mezőben, így igazolja magát. A kéréseket a szerver ennek hiányában nem teljesíti.

4. Szerver

A következő fejezet a szerver alkalmazás és az adatbázis fejlesztői dokumentációját tartalmazza. Az elkészült szerveralkalmazás megtalálható a Microsoft felhőszolgáltatójának, az Azurenak, a rendszerében a testweb-gyurisb.azurewebsites.net címen.

4.1. Adatbázis

Az alkalmazás két féle adatbáziskezelőt támogat a perzisztens adatok tárolására. Az adatokat tárolhatjuk MySQL adatbázisban, ezt a módot választottam az Azure felhőben. Az adatokat SQL Server-en is tárolhatjuk, a lokális fejlesztéshez ezt használtam. Az adateléréshez először generálni kell a megfelelő adatbázis sémát. Az ezt generáló kódot az ADO.NET Entity Data Model Designer segítségével hoztam létre a rendszertev alapján, így írtam egy szkritet az SQL Server sémájának generálására és a MySQL Server sémájának generálására. Ezek a fájlok megtalálhatóak a projekt gyökerében *DBSchema_mysql.sql* és *DBSchema_sqlserver.sql* nével.
Az alkalmazás **robosztusságát** növelik a különböző kényszerek az adatbázisban (pl. külső kulcsok: nem lehet törölni kártyát tartalmazó kategóriát).

4.2. Adatelérés

Miután generáltam az adatbázist, konfiguráltam a rendszer legalsó rétegét, az adatelérést. Ehhez Entity Framework-öt használtam. Létrehoztam egy ADO.NET Entity Data Modelt database first módszerrel, ez automatikusan generálta az entitások elérését segítő osztályokat és a táblák elérését lehetővé tevő *FlashCardContext* osztályt.
Ahhoz, hogy az adatelérés csatlakozhasson az adatbázishoz be kell állítani a megfelelő connector-okat a *Web.config*-ban. Ehhez a <connectionStrings> elemeit és az <entityFramework> szekciót kell beállítani. Természetesen más beállítások szükségesek a lokális adatbázishoz és más az Azure-ban. Az Azure-ban található konfigurációt a *Web_mysql.config* fájlba helyeztem el.

4.3. WEB API vezérlők

Az üzleti logikai réteget a keretrendszer vezérlőiben, ApiController osztályokban valósítottam meg. Három saját vezérlőt hoztam létre: AccountController, CardsController, CategoriesController. Ezek tagfüggvényeiben valósítottam meg a rendszerterv REST interfészében specifikált műveleteket. Minden művelethet egy külön függvény tartozik.
A vezérlők először **engedélyezik** (authorization) a kérést, ellenőrzik, hogy az azonosított (authentication) felhasználónak van e joga az adott művelethez, illetve létezik e az adott erőforrás. Azonosítatlan felhasználóknak *401-es* (unauthorized), jogosulatlan művelet esetén *403-as* (forbidden), nem létező erőforrásra hivatkozáskor *404-es* (not found), értelmetlen műveletekre *400-as* (bad request) hibaüzenet érkezik. A jogosultság ellenőrzés során módosításkor azt ellenőrzik hogy az adott erőforrás a felhasználó sajátja-e, olvasáskor hogy hozzáférhet e (a kategória vagy a kártya kategóriája publikus/saját/megosztott-e).
Engedélyezés után a vezérlők **végrehajtják** a kérést. Ezt az adatelérés (entity framework) különböző függvényeivel teszik meg (keresés, beszúrás, módosítás vagy összetettebb lekérdezések). Ezután 200/201-es üzenettel és a megfelelő tartalommal, esetleg erőforrás-azonosítóval visszatérnek.

4.4. Kategória megosztási kulcs

A specifikázióban kimondtuk, hogy a privát kategóriákat meg kell tudni osztani másokkal hivatkozáson keresztül, viszont illetéktelenek nem szabad hogy hozzáférhessenek. Ehhez az szükséges, hogy a szerver adjon egy kategóriához tartozó jelszót a tulajdonosának, majd ennek segítségével más felhasználók elérhesség a megosztott kategóriát. A jelszó képzéshez RSA algoritmust használtam. A jelszó egy szerveren tárolt privát kulcs (serverkey.config) segítségével a kategória azonosítójából áll elő enkriptálással. Ez a jelszó bármikor előállítható az azonosítóból, így a szerver megosztáskor és feloldáskor is ellenőrizni tudja a helyességét, de csak a privát kulcs segítségével, így illetéktelen személyek számára nem hozzáférhető. Csak azok tudják elérni a megosztott kategóriát, akik megkapták a kódot a kliens által generált hivatkozás formájában.

4.5. Naplózás

Szerver oldali naplózáshoz az NLog keretrendszert használtam. Ez egy könnyen kiterjeszthető és konfigurálható keretrendszer, különböző célokba tud naplóbejegyzéseket menteni (pl. fájlba, adatbázisba, hálózatra, email üzenetbe, stb.). Konfigurálható, hogy milyen adatokat mentsen és ezeket hova, valamint milyen formátumban. A naplózás célja a monitorozás és hibakeresés.

Három féle naplófájl keletkezik futás közben, minden naplóbejegyzésben mentem a bejegyzés időpontját valamint fontossági szintjét. Ezeken felül a következőket:

request-log: A *FlashCardServer.Handlers.LoggingHandler* osztályban valósítottam meg. A *WebApiConfig* osztály *Register* metódusában adom hozzá a MessageHandler-ekhez az osztály példányát. Ekkor minden HTTP kérés kezelésébe bele tud szólni. A *SendAsync* függvényben naplózom minden kérés URL címét, a küldő IP címét, a felhasználói azonosítóját valamint a válasz kódját.

application-log: A *LoggingHandler* osztály létrejövését és törlését naplózom, de ide lehetne naplózni egyéb alkalmazásszintű eseményeket speciális üzenettel.

error-log: Ha bármilyen kezeletlen kivétel keletkezik az alkalmazásban azt ide mentem. A kivétel típusát, üzenetét és StackTrace-ét mentem. *FlashCardServer.Handlers. ErrorLoggingFilter* osztályban és a Global.asax-ban naplózok ide.

4.6. Biztonság

A rendszert úgy terveztük meg, hogy teljesen biztonságos legyen. Ez a következő összetevőkből áll:

- A kommunikációt nem lehet lehalgatni, tartalmát csak a kliens és a szerver végpontok ismerhetik. **Megoldás:** HTTPS kommunikációs csatornát használunk minden kérésre. Ez automatikusan elvégzi a titkosítást.
- A szervert nem lehet meghamisítani és így jelszavakat vagy egyéb adatokat ellopni a kliensektől. **Megoldás:** Az azurewebsites.com domain rendelkezik tanusítvánnyal, tehát a webszolgáltatás identitását ők tudják igazolni.
- Senki nem tud más nevében (azaz azonosító token vagy felhasználó/jelszó nélkül) műveleteket végrehajtani. **Megoldás:** Az autentikációt az OAuth2 keretrendszer biztosítja, a titkosításnak köszönhetően tokent nem lehet lopni.

- Senki nem tud számára nem engedélyezett műveletet végrehajtani. **Megoldás:** A vezérlő műveleteit csak azonosított felhasználók hívhatják meg, ekkor szerver oldalon hajtjuk végre a jogosultságuk ellenőrzését (authorization).
- Az adatbázis nem támadható csak szabályos műveletek hajthatóak végre. **Megoldás:** Az entity framework biztosítja a paraméterkezelést és az SQL Injection elleni védelmet.
- Privát adatok nem kerülhetnek ki (konfigurációs fájlok, naplók, adatbázisok, kulcsok). **Megoldás:** Az ASP.NET keretrendszer nem teszi lehetővé akármilyen kiterjesztésű fájlok (.config), valamint az App_Data tartalmának (napló, adatbázis) elérését kívülről.

5. Kliens

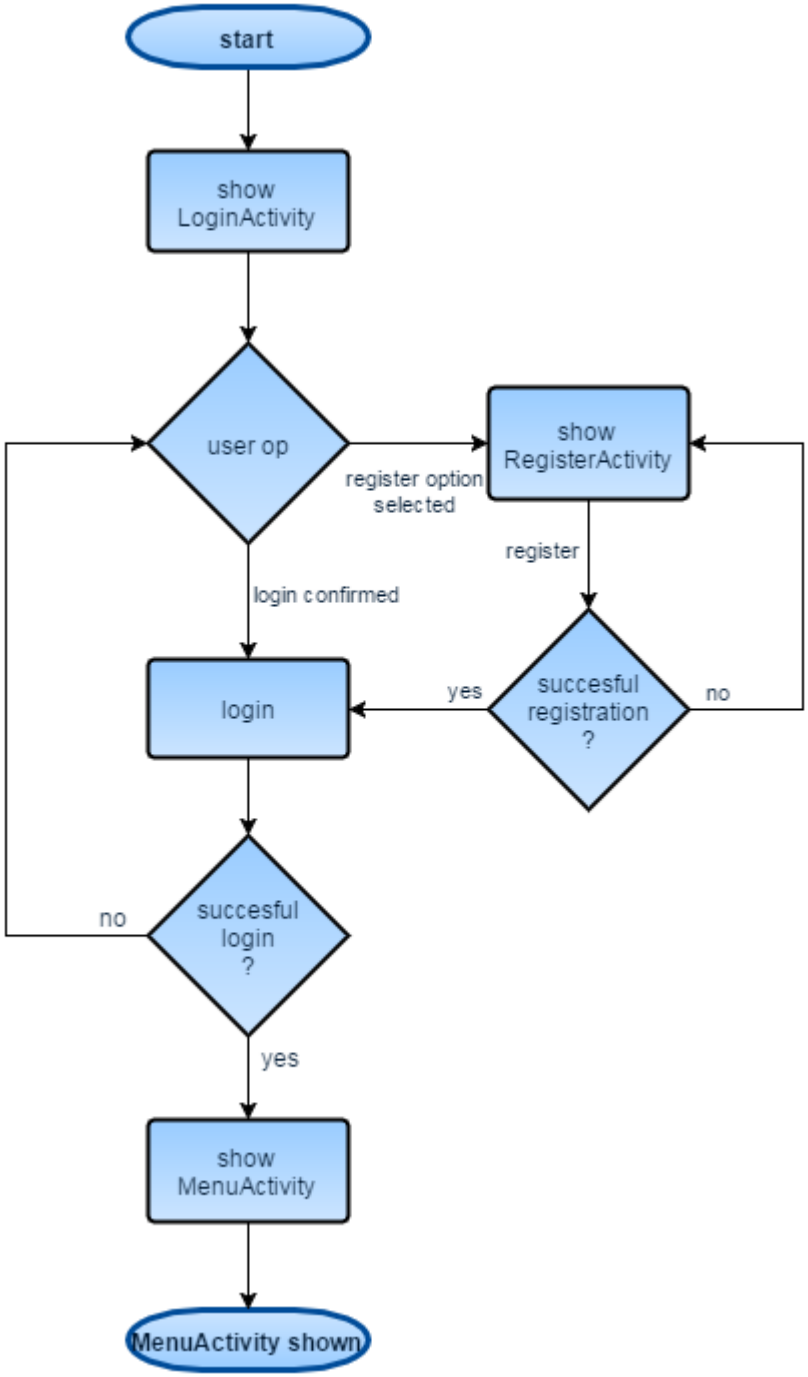
A kliens alkalmazás feladata, hogy a szerver által REST API-n elérhető funkciókat a felhasználó számára kezelhetővé tegye. Ehhez a felhasználónak az alkalmazás funkcióit grafikus formában, az alkalmazás logikához illeszkedve kell elérhetővé tenni.

5.1. Alkalmazás logika

Az alkalmazás funkcióit a 2.1.6. fejezetben látható use-case-ek alapján 4 fő csoportra osztottam:

- Felhasználókezelés
- Kategóriák szerkesztése
- Kártyák szerkesztése
- Játék (tesztelés)

Az alkalmazás indulásakor a bejelentkezést végző képernyő kerül előtérbe, mivel a követelményspecifikáció 2.1.1. pontja alapján az alkalmazás minden egyéb funkciójához szükséges, hogy a felhasználó bejelentkezve legyen a fiókjába. Innen lehetőség van a regisztrációs képernyőre navigálni. Azonban az alkalmazás főmenüje csak sikeres bejelentkezést követően jelenik meg. Az alkalmazás ekkor megjegyzi a bejelentkező tokent így később nem lesz szükséges ismételt bejelentkezésre. Az alkalmazás indulásának terveit az alábbi folyamatábra szemlélteti:



Sikeres bejelentkezést követően a felhasználó az alkalmazás főmenüjében 3 lehetőség közül választhat:

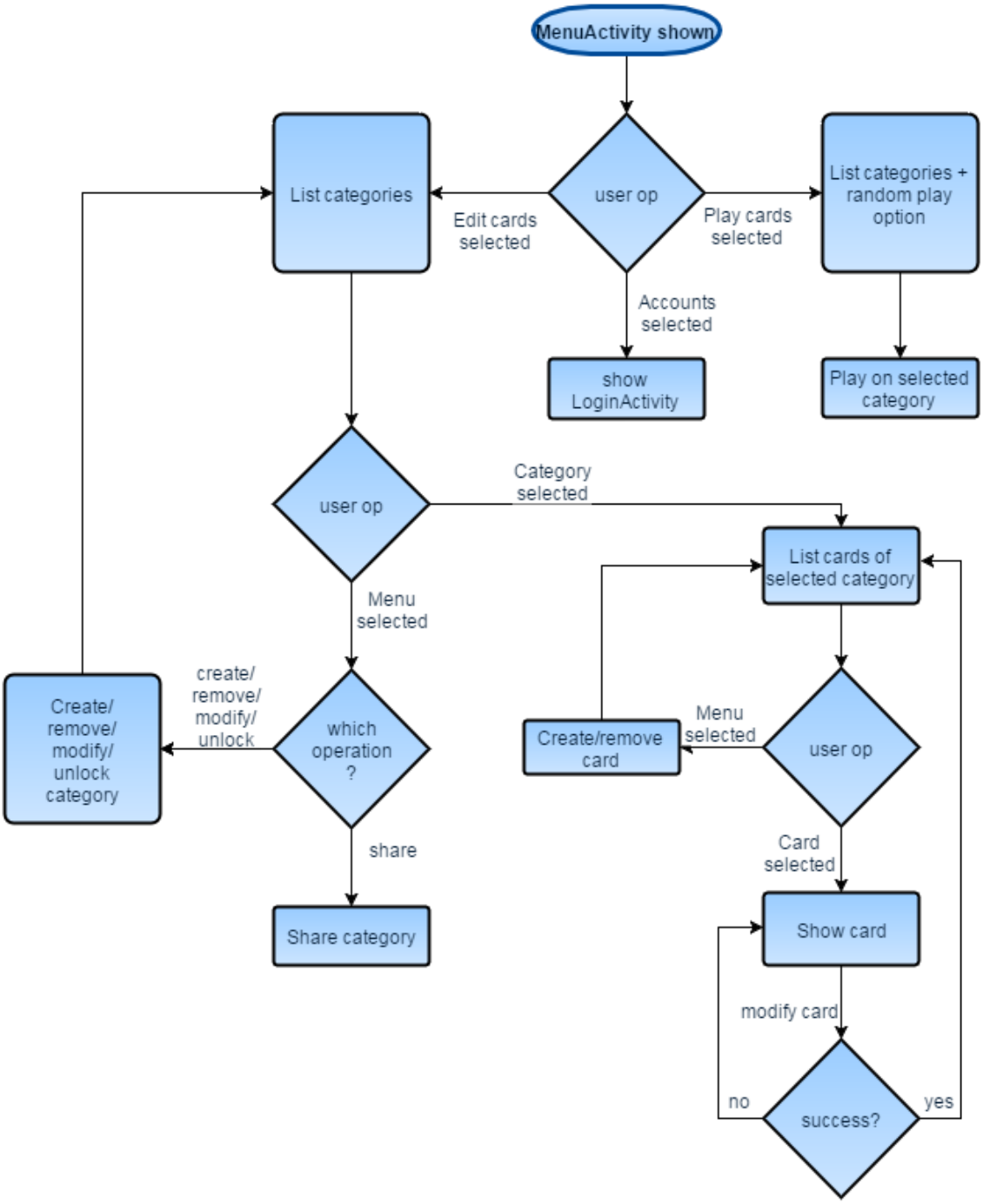
- Játék
- Kártyák szerkesztése
- Kijelentkezés

A *Játék* opciót választva az alkalmazás lekéri a felhasználó által látható kategóriákat. A felhasználó választhat, hogy melyik kategórián szeretne játszani, továbbá a helyi menüben kiválaszthatja a *Játék véletlen kártyákon* lehetőséget is. Választása után elindul a játék.

A *Kártyák szerkesztése* lehetőséget választva az alkalmazás lekéri a felhasználó által szerkeszthető (azaz a saját) kategóriákat. Itt a felhasználó a helyi menüből új kategóriákat adhat hozzá. Vagy kiválaszthatja az egyik kategóriát, ekkor megjelennek a kategória kártyái, a helyi menüből kártyát adhatunk hozzá, törölhetünk, vagy a kategóriát módosíthatjuk, megoszthatjuk, törölhetjük.

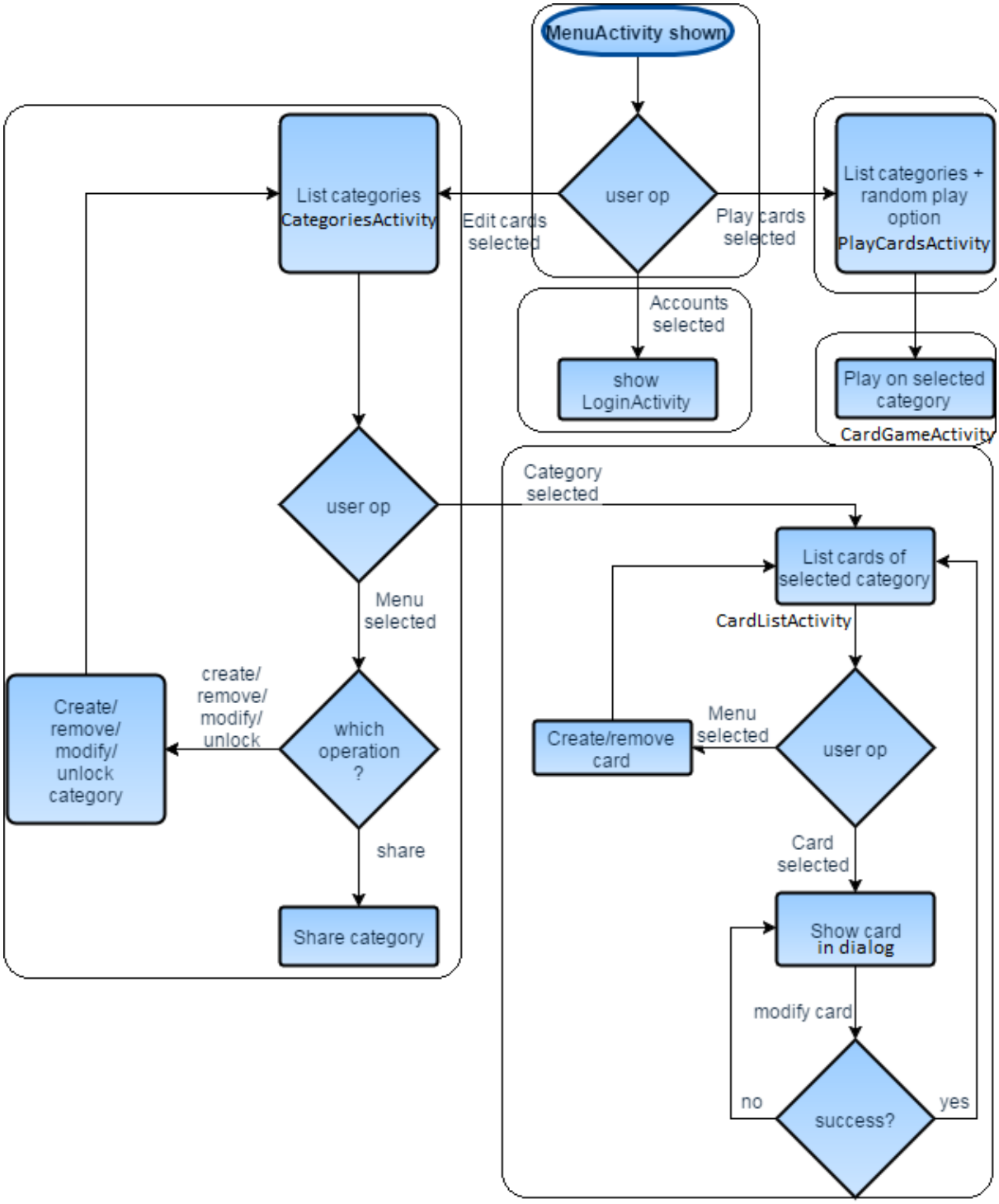
A *Kijelentkezés* opciót választva az alkalmazás elfelejti az azonosító tokent és a bejelentkező oldal jelenik meg.

Az előbbieket a következő folyamatábra szemlélteti:

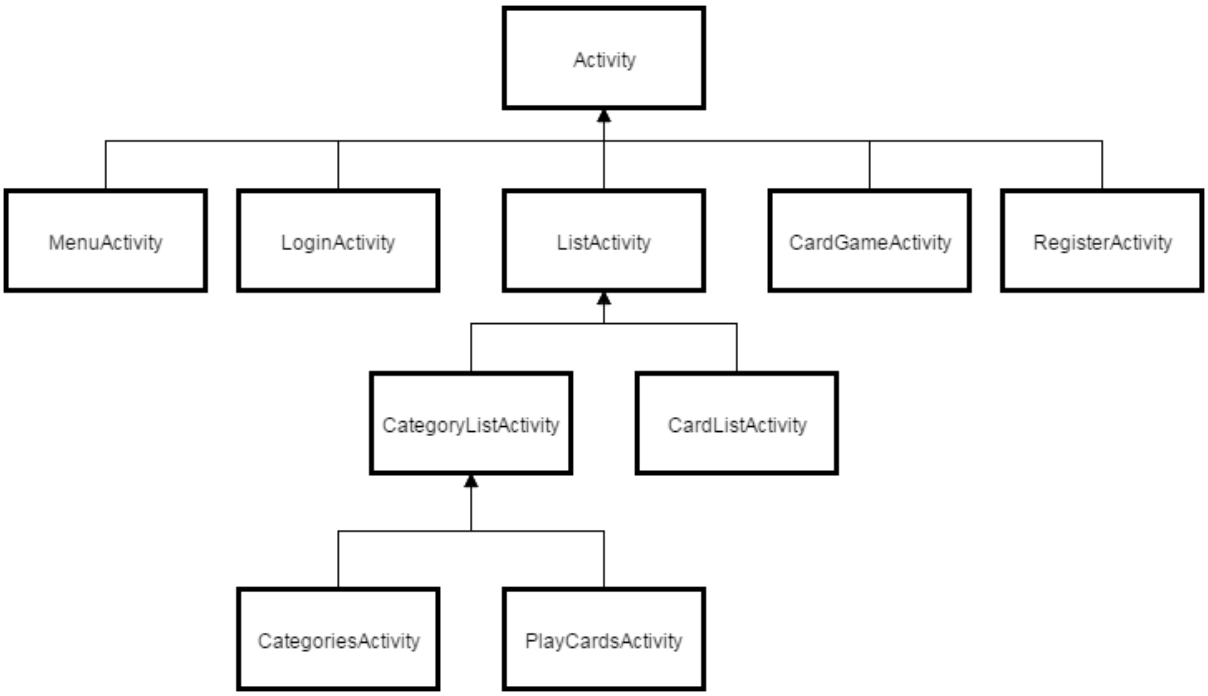


5.2. Alkalmazás architektúra

Az alkalmazás képernyőit (Android Activity-k) 5.1. fejezetben látható folyamatábrák funkcióit csoportosítva határoztam meg, ahogy az a következő ábrán látható:

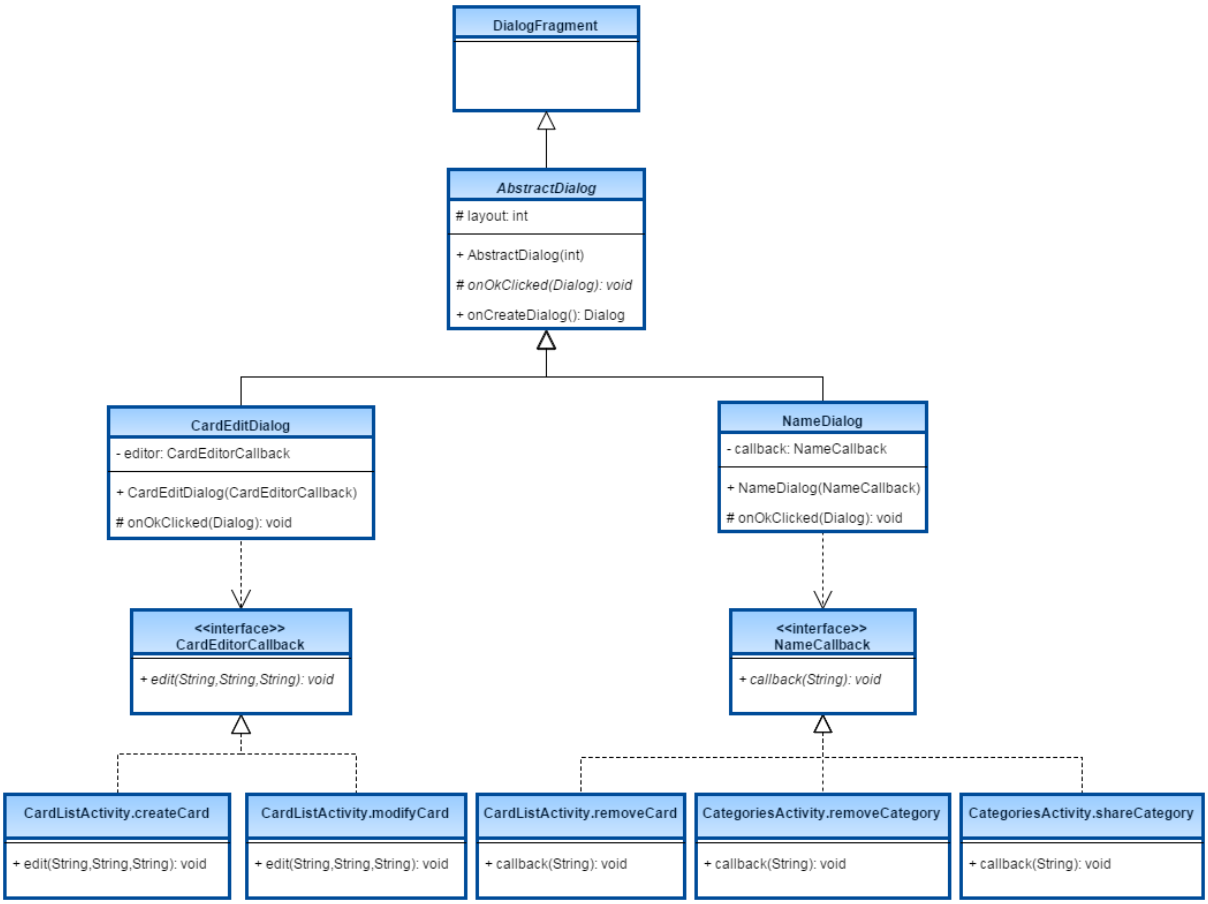


A különböző funkciókhoz tartozó Activity-knek azonban vannak közös műveleteik. Listát kell megjelenítenie például a CardListActivity-nek és a CategoriesActivity-nek is. Kategóriákat tartalmazó listát kell megjelenítenie a CategoriesActivity-nek és a PlayCardsActivity-nek is. Ezért az Activity-khez az alábbi ábrán látható osztályhierarchiát készítettem el:



Az Activity és ListActivity osztályok az Android API osztályai. A CategoryListActivity kifejezetten Kategóriák listázásához szükséges közös kódot tartalmazó osztály.

Az Activity-k mellett az alkalmazás egyes funkcióihoz tartozó dialógusai is hasonló szerkezetűek. A kódismétlés csökkentése érdekében a dialógusok kezelésére az alábbi architektúrát terveztem meg:



Az AbstractDialog absztrakt osztályban kerül megvalósításra a dialógusablakok közös funkcionalitása. Az osztály konstruktora paraméterül kapja a megjelenítendő layout azonosítóját, az onCreateDialog() metódus pedig inicializálja a közös komponenseket. A konkrét dialógus osztályoknak az absztrakt onOkClicked() metódust kell felüldefiniálniuk. Ebben a metódusban a konkrét osztályok lekérdezik a felhasználói inputot a dialógusablak mezőiből, és átadják a megfelelő Callback interfész metódusának. A Callback interfészt megvalósító osztályt a konkrét dialógus osztályok példányosításakor a létrehozó komponens adja át a dialógusnak. A Callback interfészt megvalósító osztály hajtja végre a dialógus OK gombjának kiválasztásakor végrehajtandó funkciókat.

5.3. Kommunikáció

A kliens alkalmazás a szerverrel Https protokollal, REST API segítségével kommunikál. Az üzleti objektumok struktúráiból a szerver automatikusan sorosít JSON objektumokat, ezeket a kliensnek vissza kell állítania. A szerver üzleti objektum struktúrái a kliens kódjába minimális változtatással átemelhetőek (C# struct -> Java public class), ezeket kliens oldalon a model package tartalmazza.

A Retrofit library-t választottam a automatkus JSON sorosításra és a REST API kéréseinek elvégzésére. Ez az OkHttp library-t használja, így ez utóbbira is szükség volt.

A Retrofit használatakor a REST API hívásoknak megfelelő interfészeket kell definiálni. Ehhez a REST API funkcióit 3 részre osztottam:

- Felhasználókezelés: AccountAPI
- Kategóriák kezelése: CategoryAPI
- Kártyák kezelése: CardAPI

Az API interfészek metódusai és a REST API közötti kapcsolatot annotációkkal lehet megadni, az alábbi példában látható módon:

```
public interface AccountAPI {

    @POST("/account/register")
    public Call<Void> registerUser(@Body RegisterParams params);

    /**
     * @param grantType must be "password"
     */
    @FormUrlEncoded
    @POST("/token")
    public Call<LoginResponse> loginUser(
            @Field("grant_type") String grantType,
            @Field("username") String email,
            @Field("password") String pass
            );

}
```

Az API interfészekhez a Retrofit create() metódusával lehet létrehozni az implementációkat. Mivel ezeket az API-kat a teljes alkalmazásból el kell tudni érni, ezért ezeket a statikus RetrofitWrapper osztályba csomagoltam.

A felhasználó bejelentkezéskor egy autentikációs token-t kap, amit minden, a Kategóriákon és Kártyákon végzett REST híváshoz mellékelnie kell a Http kérés fejlécében. Ezt nem célszerű az API interfészek minden metódusához külön hozzáadni, helyette a Retrofit által a Http kommunikációra használt OkHttpClient osztályhoz kell hozzáadni egy Interceptor-t, ami így automatikusan hozzáfűzi az autentikációs fejlécet minden kártyákon és kategóriákon végzett kéréshez:

```
// Init retrofit for authorized requests
// Create OkHttp interceptor to add Authorization header to all requests
OkHttpClient httpClient = new OkHttpClient();
httpClient.networkInterceptors().add(new Interceptor() {
    @Override
    public Response intercept(Chain chain) throws IOException {
        Request request = chain.request().newBuilder()
                .addHeader("Authorization", "Bearer " + authenticationToken)
                .build();
        return chain.proceed(request);
```

```
    }
});
Retrofit authorizedRetrofit = new Retrofit.Builder()
.baseUrl(baseURL)
.client(httpClient)
.addConverterFactory(GsonConverterFactory.create())
//.addConverterFactory(SimpleXmlConverterFactory.create())
.build();
// Init APIs
cardAPI = authorizedRetrofit.create(CardAPI.class);
categoryAPI = authorizedRetrofit.create(CategoryAPI.class);
```

A Http kommunikáció sok időt vehet igénybe, a felhasználói felület azonban ez alatt az idő alatt sem blokkolódhat. A probléma megoldására kényelmes lehetőséget nyújt a Retrofit aszinkron hívása, ami a UI szálról a Http híváskor azonnal visszatér, majd a sikeres és sikertelen eredmény esetén meghívódó callback metódusaiban automatikusan visszatér a UI szálra, így a hívások eredményének kijelzéséhez nem kell manuálisan szálat váltani:

```
// Send request to server
Call<LoginResponse> call =
        RetrofitWrapper.getAccountAPI().loginUser("password", email, pass);
call.enqueue(new Callback<LoginResponse>(){
    @Override
    public void onFailure(Throwable arg0) {
  Toaster.error(LoginActivity.this, getString(R.string.login_failed));
    }

    @Override
    public void onResponse(Response<LoginResponse> arg0, Retrofit arg1) {
        if(arg0.isSuccess()) {
   Toaster.info(LoginActivity.this, getString(R.string.login_successful));
            onSuccessResponse(arg0.body());
        }
        else {
   Toaster.error(LoginActivity.this, getString(R.string.login_failed));
        }
    }

});
```

5.4. Naplózás

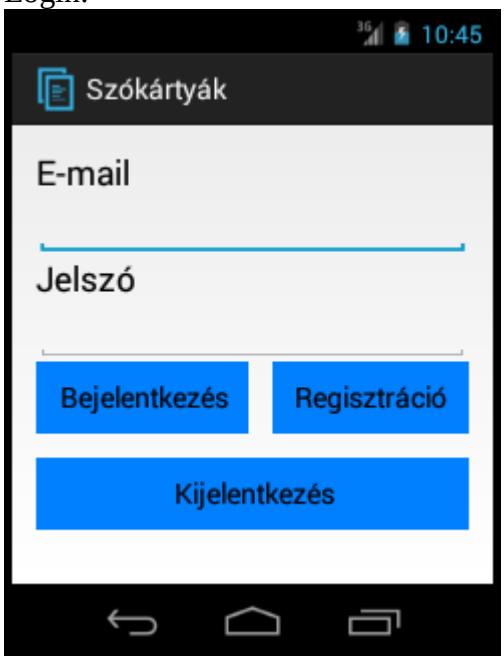
A kliens a hibakeresés megkönnyítése érdekében az alkalmazás fontosabb eseményeit és kivételeit naplózza. A napló fájl a kliens készülék háttértárán (SD kártya) perzisztensen van tárolva. A naplózást végző kódrészlet:

```
private static void appendLog(String level, String text){
        File logFile = new File("FlashCardAppLog.txt");
        if (!logFile.exists()){
            try{
                logFile.createNewFile();
            } catch (IOException e){
```

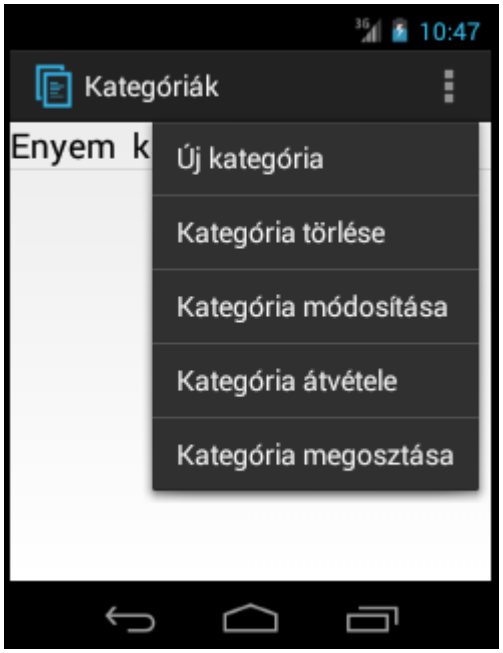
```
            e.printStackTrace();
        }
    }
    try{
     // BufferedWriter for performance, true to set append to file flag
BufferedWriter buf = new BufferedWriter(new FileWriter(logFile, true));
        buf.append(level + ": " + text);
        buf.newLine();
        buf.close();
    } catch (IOException e){
        e.printStackTrace();
    }
}
```

5.5. Képernyőképek

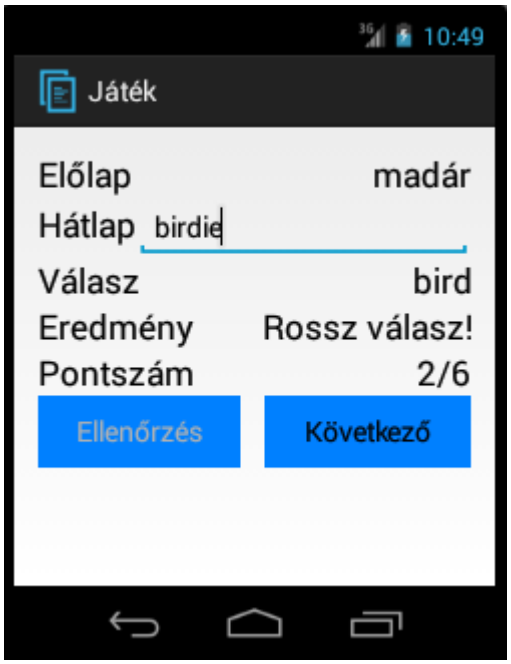
Login:



Kategóriák:


10:47
Kategóriák
Enyem k
Új kategória
Kategória törlése
Kategória módosítása
Kategória átvétele
Kategória megosztása

Játék:


10:49
Játék
Előlap
madár
Hátlap
birdie
Válasz
bird
Eredmény
Rossz válasz!
Pontszám
2/6
Ellenőrzés
Következő

6. Felhasznált eszközök

6.1. Szerver

- Visual Studio 2013 Ultimate
- IIS Server Express
- Microsoft Azure webalkalmazás platform + mysql adatbázis
- ASP.NET Web API (+ Individual User Accounts modul)
- Entity Framework + MySql.Data.Entity (driver)
- NLog

6.2. Kliens

- Eclipse ADT
- Android API
- Retrofit 2.0.0-beta2
- OkHttp 2.5.0
- OkIo 1.6.0
- Gson 2.3.1
- Converter-gson 2.0.0-beta2
- draw.io: diagramok készítéséhez
- Concorde Rookie tablet: teszteléshez

7. Összegzés

Munkánk során specifikáltuk, megtervesztük végül implementáltuk a Szókártya alkalmazást és az azt kiszolgáló szoftverrendszert. Az elkészített alkalmazás segítségével szókártyákat hozhatunk létre, ezeket kategóriákba rendezhetjük és megoszthatjuk őket a közösséggel. Ezeket a szókártyákat mások nyelvtanulásra használhatják fel.

A megvalósított alkalmazás 3 rétegű architektúrát használ. Az adatelérési és az üzleti logikai réteg a központi szerver alkalmazáson, a megjelenítési (a felhasználói felület) az Android-os kliens alkalmazáson található meg. A központi szerver a perzistens adatokat egy MySQL adatbázisban tárolja, de konfigurálható SQL Server-rel való kompatibilitásra is.

Munkánk során nagy mennyiségű tervezési és implementációs munkát végeztünk, melynek eredménye egy biztonságos és könnyen kezelhető egyszerű alkalmazás lett, mely a specifikációban leírt követelményeket kielégíti.

8. Továbbfejlesztési lehetőségek

8.1. Szerver

- Bináris mező segítségével megvalósítható lenne, hogy a szókártyákhoz az előlapra képet is tárolhassunk az a szöveg helyett vagy mellé.
- Megvalósítható lenne, hogy a hátlapon szinonímákat is tárolhassunk, így ezeket mind elfogadhatnánk. Ekkor az adatbázisban meg kell valósítani minden kártyához több szöveg tárolását.
- Kitalálási idő limittel is bővíthető lenne egy adott kártya.
- Az alkalmazás bővíthető lenne szerver oldali eredmény tárolással (ehhez biztonsági megoldások is kellenének, pl. a szerver ne küldje ki a kártya hátlapját idő előtt).

8.2. Kliens

- A kategóriák megosztásakor jelenleg a kapott linket a felhasználónak manuálisan kell megosztania a közösségi oldalakon (vagy egyéb módon). Ezt a közösségi oldalak által nyújtott API-k használatának segítségével lehetne automatizálni (pl. Facebook API).
- Kategóriák listájában egyéb adatokat is meg tudnánk jeleníteni (kártyák száma, tulajdonos, nyelv ikon).
- A szavak kitalálásakor lehessen segítséget kérni, amely megad néhány karaktert a szóból. Az így kitalált szavakért járhatna rész pont.

9. Hivatkozások

Retrofit library: http://square.github.io/retrofit/ - 2015.11.28.

NLog library: http://nlog-project.org/ - 2015.11

10. Telepítési dokumentáció

A következő fejezetben röviden ismertetjük, hogy hogyan lehet gyorsan telepíteni a teljes szoftverrendszert.

10.1. Szerver telepítése

A szerver forráskódját és bináris állományát csatoltuk. A sollution *FlashCardServer* projekt mappájának tartalmát kell átmásolni egy IIS Server gyökérkönyvtárába (pl. Azure webalkalmazás gyökerébe). Ezután ha elindítjuk az IIS Server-t, akkor várni fogja a kéréseket és a bináris állományok segítségével ki fogja őket szolgálni. Természetesen a Web.config-ban szükséges az adatelérés konfigurálása, azaz a **connectionString** és az **adatbázis driver** beállítása, hogy hozzá tudjon férni a perzisztens adatokhoz.
A szervert lokálisan is futtathatjuk és debugolhatjuk. Ehhez a feltöltött sollution gyökerében található *FlashCardServer.sln* fájlt kell megnyitni Visual Studio 2013 Ultimate segítségével.
Kipróbálás telepítés és fordítás nélkül: Egy szerverpéldány elérhető a testweb-gyurisb.azurewebsites.net címen.

10.2. Kliens telepítése

Ha a szervert telepítettük valahova akkor a kliensnek ismernie kell ezt a címet. A forrásban, 2 helyen, a RetrofitWrapper.baseURL statikus sztringjét, valamint a AndroidManifest.xml-ben a .LoginActivity intent filterében az android:host mezőt kell átírni. Ezután természetesen újra kell fordítani, majd exportálni kell a projektet egy .apk fájlba és ezt kell egy android készüléken futtatni a telepítéshez. A fordításhoz és exportáláshoz egy Android SDK-val rendelkező Eclipse-re van szükség. Ezután az alkalmazás működőképes lesz.
Telepítés fordítás nélkül: Alapértelmezetten a *testweb-gyurisb.azurewebsites.net* cím van beállítva a szervernek, ha a *FlashCardAndroid* mappa gyökerében található *WordCardApp.apk* előre lefordított fájlt telepítjük egy készülékre, akkor ott ezen a címen fogja elérni a webszolgáltatását.