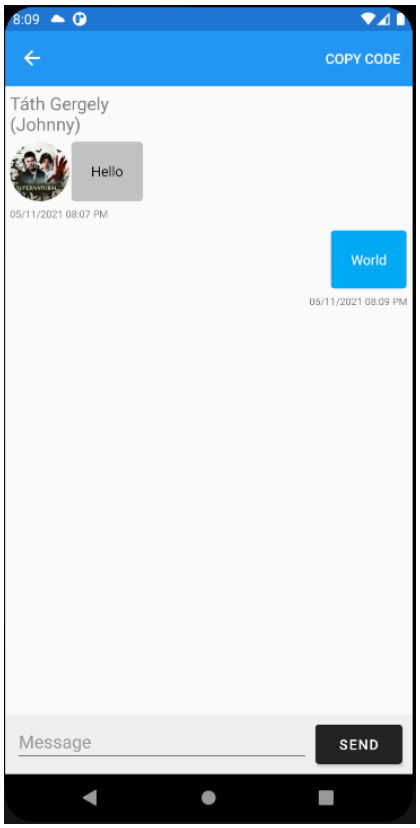
Secret Messenger

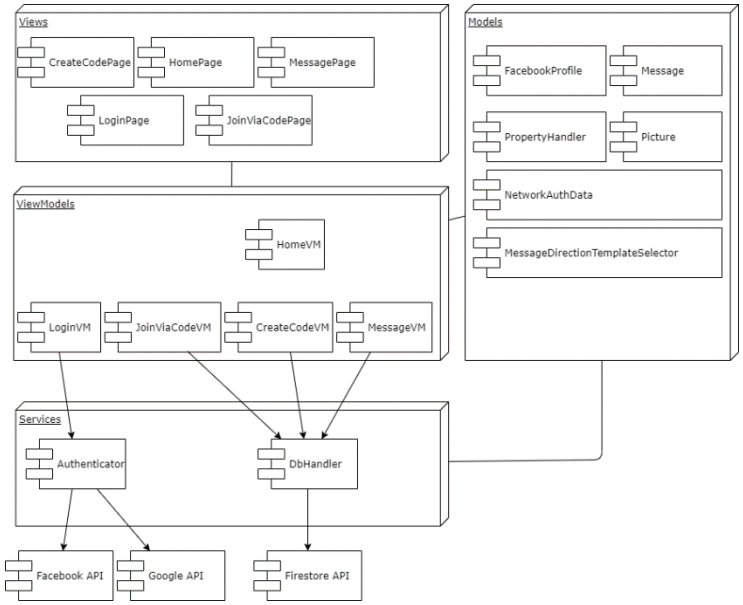
Kliensoldali technológiák házi feladat - Készítette: Táth László Gergely



A Secret Messenger alkalmazás csoportos beszélgetési lehetőséget biztosít, a már meglévő Facebook vagy Google fiókunk segítségével. Az alkalmazás nem igényel semmilyen veszélyes engedélyt, csupán internethozzáférésre van szüksége. Az egyes csoportos beszélgetésekhez való hozzáféréshez szükség van egy kódra, amit az alkalmazás generál nekünk. Ha a kódot lementettük, utána bármikor visszacsatlakozhatunk a beszélgetésbe annak segítségével. Az alkalmazás lehetővé teszi, hogy a titkos üzeneteink ne csak a szélben szálljanak, hanem megmaradjanak, és visszakövethetőek legyenek a számunkra.

Architektúra

Az alkalmazást Xamarin Forms segítségével készítettem el, az MVVM (Model View ViewModel) architektúrát követve. Az alkalmazás Material Dizájnt használ.



Szolgáltatások

Authenticator: Wrapper pluginokon keresztül kommunikál a Facebook és Google Apikkal.

DbHandler: Firebase Cloud Firestore NoSQL alapú adatbázzisal kommunikál. Az adatok JSON formátumban tárolódnak.

Komponensek

Megjegyzés: A Page-ViewModel páros egy komponensként jellemezve, a könnyebb érthetőség kedvéért.

Login: A bejelentkezést biztosítja Facebook vagy Google fiókkal.

Home: Köszönti a bejelentkezett felhasználót, és felkínálja a következő lehetőségeket:

- **CreateCode:** Új csoport létrehozása
- **JoinViaCode:** Csatlakozás meglévő csoporthoz
- Kijelentkezés

Message: A beszélgetést vezényli. Minden új üzenet esetén automatikusan frissíti a UI-t.

Modellek

FacebookProfile: A Facebook API JSON válaszának deszérializáláshoz használt osztálya.

Picture: Kép URL-t tárol.

MessageDirectionTemplate: Egy üzenet irányától (kimenő/bejövő) függően más felületet csatol hozzá.

PropertyHandler: INotifyPropertyChanged interfészt valósítja meg. A ViewModellek és a Message ebből származik.

NetworkAuthData: A FacebookProfile és GoogleUser-ből a számunkra releváns adatokat (Id, Név, Kép) tárolja.

Message: Egy adott üzenet adatait tárolja úgy, mint Szöveg, küldő és üzenet iránya (kimenő/bejövő).

Működés

Authentiákció

Mindkét esetben egy Delegate segítésével feliratkozunk az OnLogin eseménykezelőre, majd elindítunk egy bejelentkezési kérelmet. Amennyiben a kérelem nem sikerül (például a felhasználó bezárja az ablakot), akkor arról egy Toast üzenet értesít. Amennyiben sikerül a bejelentkezés, akkor egy NetworkAuthData példányába másolódnak a már bejelentkezett felhasználó számunkra fontos adatai.

Google API

Facebook API-hoz nagyon hasonló működésű. Az egyetlen különbség, hogy nem JSON, hanem GoogleUser választ ad sikeres bejelentkezés esetén, amit szintén NetworkAuthData-vá alakítunk.

Plugin: Plugin.GoogleClient (Wrapper Plugin)

Facebook API

Feliratkozik a változásokra, majd a beépített böngészőn keresztül indítja el a Facebook bejelentkezési felületet. A Delegate függvény pedig meghívódik a feliratkozás által. Amennyiben nem sikerült (például Cancelled), akkor arról a felhasználó egy „Login Failed" értesítést kap. Amennyiben sikerült, a kapott JSON-t NetworkAuthData-vá alakítja(Id, név, kép) majd átnavigál egy Home képernyőre.

Plugin: Plugin.FacebookClient (Wrapper Plugin)

```
if (_facebookService.IsLoggedIn) { _facebookService.Logout(); }
_facebookService.OnUserData += FacebookDataDelegate;
await _facebookService.RequestUserDataAsync(_fbRequestFields, _fbPermissions);
```

Szálkezelés

Csak a UI kezelés fut a főszálon. A bejelentkezéstől kezdve az üzenetek betöltéséig minden külön async szálon fut, ami értesíti a főszálat, ha szüksége van rá.

Hibakezelés

Minden kritikus pontban található hibakezelés, így véletlenül sem állhat le az alkalmazás. A hiba valós okát elrejti a felhasználótól (ahogy ezt a biztonsági elvek megkövetelik), és egy egyszerű Toast üzenetben közli az információt.

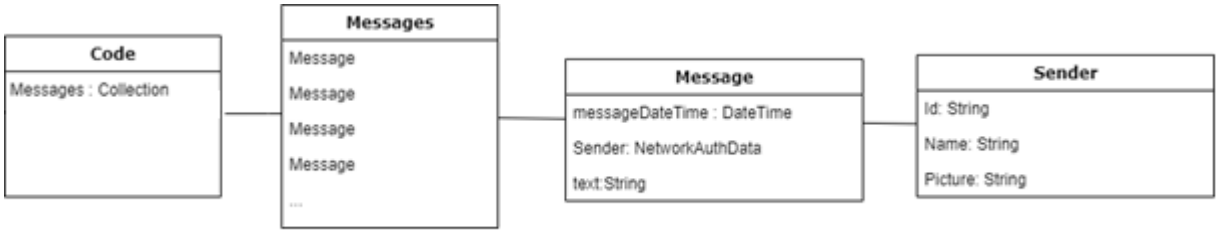
```
catch (Exception)
{
    MainThread.BeginInvokeOnMainThread(()=>{
        DependencyService.Get<IToast>().Show("Login Failed");
    });
}
```

Üzenetküldés

Amint a felhasználó el szeretné küldeni az üzenetét, akkor az elküldendő szöveghez hozzácsatolódnak a küldő adatai, majd JSON formában a DbHandler segítségével a Firestore adatbázisába kerülnek. Minden módosításról értesülnek az aktuális csoport aktív használói, akiknek egy lekérdezés után frissül a felülete és megjelenik az új üzenet. Minden üzenetről el kell dönteni, hogy bejövő vagy kimenő, majd a MessageDirectionTemplateSelector a megfelelő felületet csatolja hozzá (kimenő: jobb oldalt, kék színnel; bejövő: bal oldalt, küldő adataival)

Adatbázis struktúrája

Kollekció: Code → Dokumentumok: Messages → Kollekció: Message → Dokumentum: Sender



Frissítés

A Firestore sajátos tulajdonsága, hogy fel tudunk iratkozni egy kollekció vagy dokumentum változásra, hasonlóan, mint egy Livedata/Lifecycle folyam Android esetén. A Callback értesíti a kérőt (esetünkben a MessageViewModel-t) a változásról. Változás esetén újratölti az összes üzenetet. Azért nem csak egyet, mert ha sok üzenet jön egy periódus alatt, akkor elveszhetnének egyes üzenetek.

```
_firestore.Collection(CodesCollection).Document(code).Collection(MessagesCollection)
.AddSnapshotListener(callback);
```

Források

Draw.io – Ábrák készítése

A felhasznált Xamarin pluginok jelentős részét MIT hallgatók készítették