Iváncsics Barnabás Pál

# MOBIL ALKALMAZÁS FEJLESZTÉSE IOS PLATFORMRA

MSc. Önálló laboratórium 1. dokumentáció

KONZULENS

Albert István

BUDAPEST, 2024

# Tartalomjegyzék

# 1 Bevezetés

Az Önálló laboratórium 1. keretein belül feladatom egy mobilalkalmazás elkészítése volt iOS platformra saját ötlet alapján. A témán belül egy turisztikai applikációt készítettem, aminek a neve: „Fedezzük fel!” Az alkalmazás használatával magyarországi városokat járhatunk be, fedezhetünk fel. A felhasználó a különböző városokon belül helyszínek és séták közül válogathat, amelyek felkeresésével, bejárásával megismerheti az adott településeket.

# 2 Funkcionális követelmények

Az alkalmazással szembeni funkcionális követelményeket a konzulensemmel egyeztetve én határoztam meg a félév elején. Ezek a következők voltak:

- Több képernyő létrehozása
  - Az alkalmazásban legyen több képernyő, oldal, amik között lehessen navigálni.
  - A városoknak, helyszíneknek és sétáknak legyen külön listája.
    - A séták listáját lehessen távolság alapján szűrni
  - Legyen egy részletező oldal a helyszíneknek és sétáknak, ahol a felhasználó hasznos információkat tudhat meg.
- Navigálás az állomások között
  - Egy térkép segítségével meg lehessen nézni, hogy az egyes állomások hol találhatók és a felhasználó (azaz a készüléke) hol található ahhoz képest.
  - A térképen a séták állomásai legyenek csoportosítva valamilyen módon, például legyenek összekötve egy vonallal.
- Szerveroldal
  - Az alkalmazásban használt adatok szerveroldalon legyenek tárolva, beleértve a képi tartalmakat is.
- Felhasználókezelés
  - Legyen megvalósítva valamilyen felhasználóhoz köthető funkció, amit csak a regisztrált és bejelentkezett felhasználók tudnak igénybe venni.
- Platformok
  - Az alkalmazásnak fordulnia és futnia kell iOS platformon.

# 3 Áttekintés

Ebben a fejezetben röviden bemutatom képekkel illusztrálva, hogy a fent meghatározott követelmények közül mit és hogyan sikerült megvalósítanom.

## 3.1 Betöltő képernyő

Az alkalmazás elindítása után, az inicializálás alatt egy indító képernyő jelenik meg (*1. ábra*).

## 3.2 Kezdő képernyő

Miután a háttérben megtörtént az inicializálás megjelenik a Kezdő képernyő *(2. ábra)*. Ezen az oldalon található a városok listája.

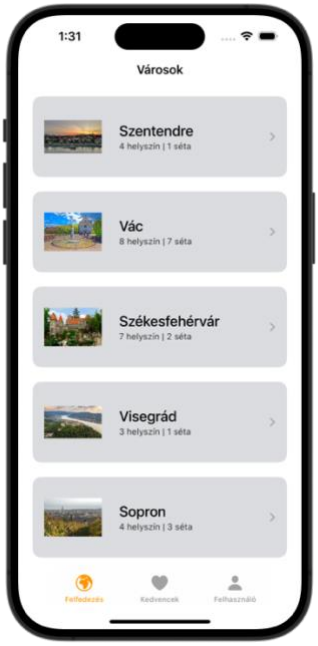
## 3.3 Menü

A 2. *ábrán* látható, hogy az alkalmazás alján található egy menü, ahonnan a 3 fő menüpont érhető el. Ez a menü mindegyik képernyőn megjelenik függetlenül attól, hogy éppen hol vagyunk a navigációs stack-en belül.

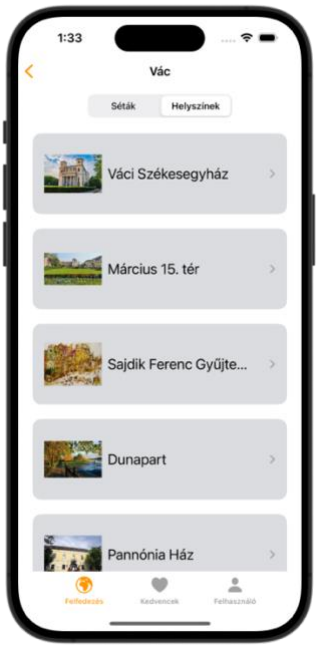
Az egyes menüpontokra kattintva az alábbi képernyők jelennek meg:

- Főoldal: Kezdő képernyő *(2. ábra)*
- Kedvencek: Kedvencek képernyő *(13. ábra, függelék)*
- Felhasználó: Felhasználó képernyő *(12. ábra, függelék)*


*1. ábra:Indító képernyő*




*2. ábra: Kezdő képernyő*



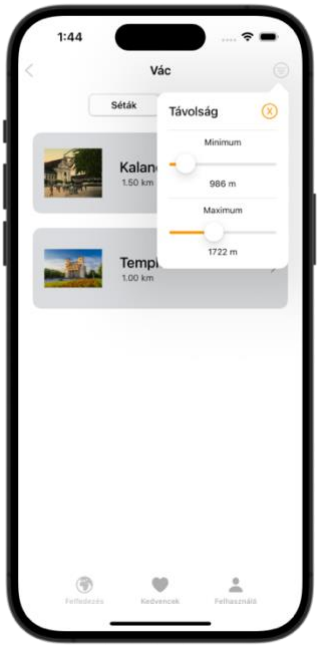

*3. ábra: Helyszínek lista*

## 3.4 Helyszínek és séták lista képernyő

A városok listából egy elemet kiválasztva jut el a felhasználó a séták és helyszínek *(3. ábra)* listához. A séták és a helyszínek között a képernyő tetején található gombokkal lehet váltani. A séták listáját távolság szerint lehet szűrni *(4. ábra)*.

## 3.5 Séta és helyszín részletei képernyő

A fent említett listákból egy elemet kiválasztva a séta *(5. ábra)*, vagy helyszín *(6. ábra)* részleteit bemutató képernyőre lehet jutni. A két képernyő hasonlóan lett felépítve azzal a különbséggel, hogy sétáknál található egy plusz lista is, amiben a sétához tartozó helyszínek is megjelennek.




*4. ábra: Állomások lista képernyő*


*5. ábra: Állomás részletek képernyő 1.*


*6. ábra: Állomás részletek képernyő 2.*

## 3.6 Térkép képernyő

A részletező oldalon a térképre kattintva a térkép képernyőre lehet jutni, ahol az egész oldalon egy térkép jelenik meg *(8. ábra)*. A térképen a helyszínhez, illetve a sétákhoz tartozó markerek, pontok jelennek meg. A markereket ki lehet jelölni, ekkor alul egy ablak jelenik meg *(9. ábra)*, ahol további funkciók érhetőek el. A térképet ábrázoló gombra nyomva az alapértelmezett alkalmazás nyílik meg a markerhez tartozó helyszín koordinátáival. A sétáló embert ábrázoló gombra rányomva pedig a legrövidebb utat jeleníti meg a helyszínhez a térkép. Sétát megjelenítő térképnél ez a gomb a képernyő jobb alján található és megnyomására az alkalmazás legenerálja az egyik legrövidebb utat a készülék helyzetétől úgy, hogy mindegyik állomást érintse az. Ezen kívül lehetőség van „körbe nézni” *(10. ábra)* az adott helyszínnél (amennyiben elérhető az utca képe).

## 3.7 Felhasználó képernyő

Ezen az oldalon tud bejelentkezni, illetve regisztrálni *(11. ábra)* a felhasználó. Miután bejelentkezett az oldalon megjelenik egy profil nézet. Innen tud kijelentkezni az alkalmazásból, illetve a fiókját is itt tudja törölni *(12. ábra)*.

## 3.8 Kedvencek képernyő

A kedvencek képernyőn jelenik meg a felhasználó mentett helyszínei és sétái városok szerint csoportosítva *(13. ábra)*.

# 4 Megvalósítás

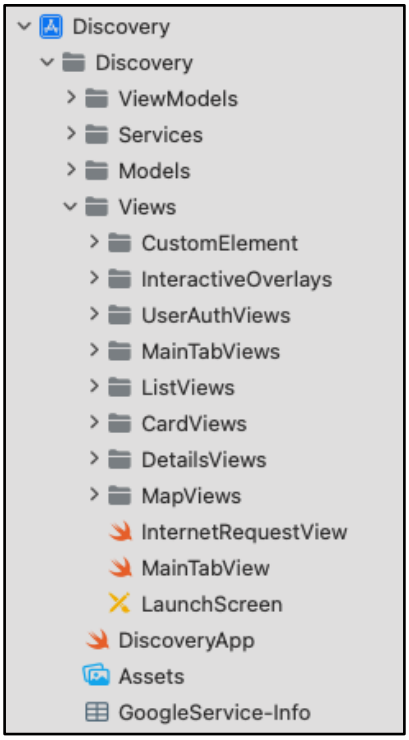
## 4.1 Keretrendszer

Az app elkészítéséhez a SwiftUI[1] keretrendszert használtam. A SwiftUI egy modern, deklaratív keretrendszer, amely intuitív és hatékony iOS-alkalmazások létrehozását teszi lehetővé. Segítségével gyorsan és hatékonyan tudtam létrehozni a felhasználói felületet.

## 4.2 Projekt Könyvtár hierarchia

Az alkalmazás elkészítése során az alábbi könyvtár hierarchiát alakítottam ki a projekthez *(7. ábra)*:

- ViewModel: Az alkalmazás ViewModel rétegét megvalósító osztályok.
- Services: Hálózat, helyzetmegosztás szükséges osztályok, illetve útvonal készítő segédosztály.
- Models: Model réteghez szükséges adatstruktúrák.
- Views: Alkalmazásban lévő nézetek.
- CustomElement: Egyéni View-k és alap SwiftUI view felüldefiniálását megvalósító View.
- InteractiveOverlays: PopOver és Sheet View-k.
- UserAuthViews: Felhasználókezeléshez tartozó View-k.
- MainTabViews: Fő oldalak az alkalmazásban.
- ListViews: Lista nézetek.
- CardViews: Lista elem, kártya nézetek.
- DetailsViews: Séta, helyszín részletező nézet.
- MapViews: Térkép View-k.
- InternetRequestView: Offline állapot esetén megjelenő nézet.
- MainTabView: Az alkalmazás fő menüjét valósítja meg.
- LaunchScreen: Indító képernyő
- DiscoveryApp: Ebben található az alkalmazás belépési pontja és a Firebase konfiguráció




*7. ábra: Projekt könyvtár hierarchia*

[1] https://developer.apple.com/xcode/swiftui/

## 4.3 MVVM architektúra

Az alkalmazás fejlesztése során törekedtem az MVVM (Model-View-ViewModel) architektúra megvalósítására. A modell réteget az adat osztályok és Firebase alkotja. A ViewModel réteget az alkalmazásban található ViewModel osztályok alkotják. A View réteget, pedig az alkalmazás felületét leíró nézet (View) osztályok alkotják.

A továbbiak a struktúra három rétege mentén fogom részletezni az elkészült alkalmazást.

### 4.3.1 Model réteg

Ez a réteg biztosítja az adatok tárolását, valamint az adatmodellek definiálását, amelyek a felhasználói felületen megjelenített információkat reprezentálják.

Az adatok tárolására (és kezelésére) a Firebase [2]szolgáltatásait használtam. Firebase-en belül a Cloud Firestore szolgáltatást használtam a városok, helyszínek séták, illetve a felhasználói adatok tárolására. Az alkalmazásban megjelenő képek tárolásához a Cloud Storage szolgáltatást vettem igénybe.

#### 4.3.1.1 Cloude Firestore

A Cloud Firestore-ban gyűjteményeket (collection) és dokumentumokat (document) lehet létrehozni azok egymásba ágyazásával.

A városoknak, helyszíneknek, sétáknak és felhasználói adatoknak külön-külön létrehoztam egy-egy gyűjteményt. Ezekben a gyűjteményekben pedig létrehoztam a dokumentumokat. Minden dokumentum egy város, vagy helyszín, vagy séta, vagy felhasználó adatait tárolja el. A dokumentum mezői teljesen megfelelnek a később, 4.3.1.3 fejezetben bemutatott adatmodellek attribútumainak.

#### 4.3.1.2 Cloud Storage

A Firestore-ban minden dokumentum 1 MB, így a nagyobb méretű tartalmak, mint például a képeket nem tudjuk a dokumentum belül tárolni. Az ilyen nagyobb tartalmú fájlok tárolására szolgál a Cloud Storage. Itt az alkalmazásban látható képeket tárolom a következő mappa struktúra alapján:

- Sétákhoz tartozó képek
  - coverPhoto/routes/[séta azonosítója]/[kép neve, kiterjesztéssel]
- Helyszínekhez tartozó képek
  - coverPhoto/locations/[helyszín azonosítója]/[kép neve, kiterjesztéssel]
- Városokhoz tartozó képek elérése
  - coverPhoto/cities/[város azonosítója]/[kép neve, kiterjesztéssel]

Egyelőre nem indokolt az, hogy séták, helyszínek és városok mappán belül legyen még egy mappa külön azok azonosítóival, mert mindegyikben csak egy kép van. Viszont a későbbiekben, amikor több kép is lesz az alkalmazásban, sokkal átláthatóbb és rendszerezett marad a struktúra.

[2] https://firebase.google.com

#### 4.3.1.3 Adatmodellek

Városohoz, helyszínhez, sétához és felhasználóhoz létrehoztam az adatmodelleket. Ezek a projekten belül a Models mappában (csoportban) találhatóak. Mindegyik adatmodell a Firestore-ban lévő dokumentumoknak felel meg, így könnyedén tudtam map-elni az adatokat az egyes lekérdezések során. A legtöbb modell megvalósítja Codable és Identifiable protokollokat. A Codable használatával könnyedén tudjuk map-elni az adatokat az adatbázis lekérdezése és írása során. Az Identifiable pedig, a SwiftUI és a Codable-nek segít abban, hogy a létrehozott modell példányok azonosíthatóak legyenek. Ez például a lista nézetekhez, vagy a térképen látható helyszínekhez elengedhetetlen.

Az alkalmazásban a következő adatmodelleket hoztam létre:

- City: Egy várost reprezentál (Codable, Identifiable).
- Location: Egy helyszínt reprezentál (Codable, Identifiable).
- Route: Egy helyszínt reprezentál (Codable, Identifiable).
- UserModel: Felhasználó mentett adatait (kedvencek funkcióhoz) reprezentálja (Codable, Identifiable).
- IdentifiableMapRoute: Azonosítható, térképen megjelenő útvonal (Identifiable)
    - Létrehozására azért volt szükség, mert a sétát megjelenítő útvonal (kék vonal), több ilyen útvonalból áll össze és így szükséges volt arra, hogy azonosíthatóak legyenek.

### 4.3.2 ViewModel réteg

A ViewModel réteg a Model és a View rétegek közötti kommunikációt biztosítja. A ViewModel lekérdezi az adatokat a Model rétegből, és biztosítja azok helyes megjelenítését a View rétegben.

A ViewModel-eken belül létrehoztam @Published változókat, tulajdonságokat, amik használatával a felhasználói felület automatikusan frissül, amikor azok változnak. A ViewModellek a projekten belül a ViewModels mappában (csoportban) találhatóak meg.

Az alkalmazásban a következő ViewModel-eket hoztam létre:

- CitiesViewModel
    - Felelős a városok adatainak kezeléséért és megjelenítéséért.
    - A Firestore adatbázisból lekéri a városok adatait
    - Fontosabb változók, függvények:
        - @Published var cities tömbben tárolja a városok adatait
        - fetchData() függvény biztosítja az adatok letöltését
- LocationsViewModel
    - Helyszínek adatainak kezeléséért felelős.
    - Firestore adatbázisból lekéri a helyszínek adatait
    - Fontosabb változók, függvények:
        - @Published var locations tömbben tárolja a helyszínek adatait.
        - fetchData() függvény biztosítja az adatok letöltését

- RoutesViewModel
  - Útvonalak adatainak kezeléséért felelős.
  - A Firestore adatbázisból lekéri az útvonalak adatait
  - Fontosabb változók, függvények:
    - @Published var routes tömbben tárolja az útvonalak adatait
    - fetchData() függvény biztosítja az adatok letöltését
- UserViewModel
  - Felhasználói adatok kezeléséért felelős.
  - Firestore adatbázisból lekéri a felhasználó adatait,
  - Fontosabb változók, függvények:
    - @Published var user változóban tárolja azokat a felhasználó adatait
    - fetchData() függvény biztosítja az adatok letöltését
    - createNewUser(): függvény új felhasználó létrehozására
    - updateUserData(): függvény felhasználói adatok frissítésére
    - deleteUserData(): függvény felhasználói adatok törlésére
- AuthenticationViewModel
  - Felhasználói hitelesítési folyamat kezeléséért felelős. Az osztály felelős a felhasználó bejelentkezéséért, regisztrációjáért, kijelentkezéséért és a hitelesítési állapot kezeléséért.
  - Fontosabb változók, függvények:
    - @Published var flow: Hietelesítési folyamat állapotát tárolja
    - @Published var authenticationState: Hitelesítés állapota
    - @Published var user: bejelentkezett felhasználót tárolja
    - signInWithEmailPassword(): függvény a bejelentkezéshez (email cím – jelszó párossal)
    - signUpWithEmailPassword(): függvény a regisztrációhoz (email cím – jelszó párossal)
    - signOut(): függvény a kijelentkezéshez
    - deleteAccount(): függvény a fiók törléséhez
  - A felhasználó adatainak kezelésére egy UserViewModel példányt is tartalmaz. Mivel, hogy a felhasználói hitelesítés és a felhasználói adatok összefüggenek, ezért úgy döntöttem, hogy itt hozom létre a UserViewModel példányt (és máshol nem). Ezáltal ezt csak az AuthenticationViewModel-en keresztül lehet elérni. Későbbiekben a két ViewModel osztályt lehet, hogy össze fogom vonni.

Itt fontos megjegyeznem, hogy a UserViewModel-en kívül az adatok lekérése során nem használtam végül snapshot listenert. Az adatok így nem frissülnek valós időben, amikor változás van az adatbázis, vagy a nézet felől. Az adatokat akkor kérem le az adatbázisból, amikor szükség van azokra például egy lista megjelenítéséhez (illetve ez esetben is csak akkor, ha még viewModel-ben található adatok hiányosak). Azért jutottam erre megoldásra, mert nem tartom relevánsnak az alkalmazás felhasználás során azt, hogy szükség legyen arra, hogy az adatbázisban történő változásokat lekezelje valós időben. Továbbá azzal számolok, hogy így a forgalmat is csökkenteni tudom. Ez pedig azért fontos, mert a Firebase az úgynevezett „Pay as you go” árazási rendszer szerint, dinamikus a forgalommal arányosan számláz.

### 4.3.3 View réteg

A View a rétegben a SwiftUI segítségével hoztam létre az alkalmazás nézeteit, amelyek a ViewModel-ekből származó adatokat használják a dinamikus felhasználói felület kialakításához. Az előző, 4.3.2 fejezetben bemutatott ViewModel-ek @Published változóinak köszönhetően az adatok változásai automatikusan frissítik a nézeteket.

Az alkalmazás fejlesztése során törekedtem arra, hogy a kinézet egységes és letisztult maradjon. Az alkalmazás UI részét és drótvázát a Figma [3]programban terveztem meg.

Fejlesztés közben figyeltem arra is, hogy minél több nézetet újra fel lehessen használni. Ilyenek voltak például részletező nézet és a listában megjelenő listaelemek, card-ok.

Az oldalak, képernyők közötti navigációhoz a NavigationStack-et és NavigationLink-et használtam.

A nézetek (View) a projekten belül a Views mappában (csoportban) és annak az almappáiban találhatóak meg (projekt struktúra leírása: 4.2 fejezet).

Az következőkben a projekt mappa struktúrája szerint mutatom be a létrehozott View struktúrákat.

#### 4.3.3.1 Views

A nézetek gyökér mappájában két nézet és az indítási képernyő található. Ezek a következőek:

- MainTabView
    - Az alkalmazás gyökér nézete. Ez egy MainTabView-t definiál, amely egy tab nézetet (az összes képernyő alján látható menüt) jelenít meg három alnézettel.
    - Emellett figyeli a hálózati kapcsolat állapotát, és az alapján dönt, hogy a tab nézetet vagy egy internetkapcsolatot igénylő nézetet jelenít meg
- InternetRequestView
    - A fent említett internetkapcsolatot igénylő nézetet valósítja meg.
- LaunchScreen
    - Az indítási képernyő található benne
    - Az Info.plist fájlban kellett beállítani, hogy ezt a storyboard-ot jelenítse meg indításkor

#### 4.3.3.2 CustomElement

Ebben a mappában egy olyan nézet van, amit az alkalmazásban több helyen is felhasználok. Ezen kívül itt található egy másik fájl is, ami SwiftUI View egyik elemét írja felül.

- UITabBarController
    - Ez a kód egy UITabBarController kiterjesztést definiál, amelyben beállítom a képernyő alján megjelenő tabBar kinézetét. Ehhez a UITabBarController osztályon belül a viewDidLayoutSubviews-t írtam felül.
    - Erre azért volt szükség, mert a SwitUI még nem biztosít olyan modifiereket a tabBar-hoz, amikkel a kívánt kinézetet megkaphattam volna.

[3] https://www.figma.com

- EmptyCoverPhotoView
    - Egy üres fedőképet jelenít meg egy kör alakú ikon formájában.
    - Ez a nézet szolgál alapértelmezett nézetként azokhoz a képekhez, amiknek a tartalmát nem lokálisan tárolom (jelen esetben az összes képnek). Ennek használatával mindig meg lesz valami jelenítve a képek helyén, akkor is ha azok nem töltődnek be rendesen.

#### 4.3.3.3 InteractiveOverlays

Itt olyan nézetek vannak, amik az éppen megjelenített nézetek fölött jelennek meg és valamilyen felhasználói interakciót várnak.

- DistanceFilterTabItem
    - Navigationbar-hoz hozzáadható gomb, aminek a megnyomásakor egy felugró ablakot jelenít meg, amelyben a felhasználó beállíthatja a minimális és maximális távolságot.
    - popover nézetet jelenít meg úgy, hogy az a gomb alatt legyen (és ne a képernyő alján)
- LocationMapSheetView
    - Egy olyan nézetet definiál, amely egy helyet jelenít meg egy interaktív térképnézettel és gombokkal a navigációhoz és a térkép megnyitásához.
    - A térképen ez a nézet jelenik meg a felugró sheet nézetben.

#### 4.3.3.4 UserAuthViews

Felhasználói fiók kezeléséhez szükséges nézeteket tartalmaz.

- AuthenticationView
    - Ez a nézet határozza meg, hogy melyik nézetet kell megjeleníteni a bejelentkezési vagy regisztrációs nézetek közül. Ehhez a viewModel flow @Published változóját használja
- SignupView
    - Egy űrlapot jelenít meg a felhasználói regisztrációhoz.
- LoginView
    - Egy felhasználói bejelentkezési űrlapot jelenít meg az alkalmazásban.
- A UserProfileView
    - Egy felhasználói profil nézetet jelenít meg az alkalmazásban
    - Lehetőség van az alkalmazásból kijelentkezni, illetve törölni a felhasználói fiókot.

#### 4.3.3.5 MainTabViews

Az alkalmazás fő menüjében található, három „gyökér” nézet van itt. Illetve a mentett állomások fő Tab nézete is itt található.

- CitiesTabView
    - Ez a kód egy olyan nézetet valósít meg, amely lehetővé teszi a felhasználók számára, hogy válasszanak a városban található séták és helyszínek közötti megjelenítésen.
- AccountView
    - Ez a kód egy felhasználói fiók gyökér nézetet valósítja meg.

    - AuthenticationViewModel authenticationState változója állapota alapján vagy a profil nézetet jeleníti meg, vagy egy másik nézetet, ami jelzi azt, hogy nincsen bejelentkezett felhasználó. Ezen a nézeten található egy gomb, aminek a megnyomása esetén a korábban, 4.3.3.4. fejezetben említett AuthenticationView jelenik meg egy felugró sheet nézeten belül.
- FavouritesView
    - Hasonlóan működik, mint a fenti AccountView. Ha be van jelentkezve a felhasználó, akkor megjeleníti a mentett elemeket tartalmazó listákat. Amennyiben nincs bejelentkezve a felhasználó egy olyan nézetet jelenít meg, ami jelzi azt, hogy nincsen bejelentkezett felhasználó.
- SavedItemsTabView
    - Egy olyan nézetet valósít meg, amely lehetővé teszi a felhasználók számára, hogy válasszanak a mentett séták és helyszínek közötti megjelenítésén.

#### 4.3.3.6 ListViews

Itt találhatóak az alkalmazásban látható lista nézetek.

- generalListItemStyle
    - Kiegészíti a SwiftUI nézetét egy egyedi stílussal.
    - Ezt a stílust használom a lista elemein.
- CitiesViews
    - Városok listáját jeleníti meg.
- RoutesViews
    - Séták listáját jeleníti meg.
- LocationsViews
    - Helyszínek listáját jeleníti meg
- savedListItemStyle
    - Kiegészíti a SwiftUI nézetét egy egyedi stílussal.
    - Ezt a stílust használom a mentett lista elemein.
    - Kicsit mást stílust kellett itt használnom, mint a sima listákban, hogy a swipe action-nél rendesen jelenjen meg a lista elem.
- SavedLocationsListView
    - Mentett helyszínek listáját jeleníti meg.
    - Azért volt szükség egy külön nézetre, mert ebben a listában városok szerint vannak csoportosítva a mentett helyszínek.
- SavedRoutesListView
    - Mentett útvonalak listáját valósítja meg.
    - Ugyanabból az okból hoztam létre külön ezt a nézetet, mint az előző SavedLocationsListView-t.

#### 4.3.3.7 CardViews

Itt találhatóak az alkalmazásban látható lista elem (card) nézetek.

- ListItemCardView
    - Általános lista elem nézet, ami a többi card-ban felhasználható.
    - Minden kártya ezt a nézet jelenít meg a szükséges adatokkal.
- CityCardView

    - Megjeleníti a ListItemCardView-t a séta adataival és hozzáad egy plusz Text nézetet, amin a séták és helyszínek száma jelenik meg.
- RouteCardView
    - Megjeleníti a ListItemCardView-t a séta adataival és hozzáad egy plusz Text nézetet, amin a séta hosszúsága jelenik meg.
- LocationCardView
    - Megjeleníti a ListItemCardView-t a helyszín adataival és hozzáad egy üres nézetet.
        - Ennél a kártyának nem szerepel plusz adat, így egy üres nézetet adtam hozzá.
- HorizontalLocationCardView
    - A séta részleteit megjelenítő oldalon található horizontális lista.
    - Ez nem a korábban említett ListItemCardView-t jeleníti meg, mert a részletező oldalon egy teljesen más nézetű kártyát kellett megjeleníteni.

#### 4.3.3.8 DetailsViews

A részletező oldalakat megvalósító nézetek találhatóak ebben a mappában.

- GeneralDetailsView
    - Mindkét részletező oldal tetején megjelenő view-kat (hasonló részeket) tartalmazza.
    - Ez egy általános részletező nézet, ami megkapja a szükséges adatokat, illetve egy másik View-t, amit a leírás alatt fog megjeleníteni.
        - A kapott View rész az, ami a két nézetben különbözik.
- LocationDetailsView
    - Megjeleníti a GeneralDetailsView nézetet és hozzáadja ahhoz még a helyszín részletező oldal alján megjelenítendő térképet.
- RouteDetailsView
    - Megjeleníti a GeneralDetailsView nézetet és hozzáadja ahhoz még a séta részletező oldal alján megjelenítendő helyszín listát (HorizontalLocationCardView) és a térképet.

#### 4.3.3.9 MapViews

Itt található a két térkép nézet. Sokat gondolkodtam azon, hogy ennek a két nézetnek is kellene egy közös nézetet létrehozni, de végül a nagyobb eltérések miatt (különböző akciókat futtatnak le az eseményeknél, a térképen megjelenő gombok is különböznek funkcionalitásban) úgy éreztem, hogy egyelőre jobb, ha még külön hagyom a kettő térképet. Ezen struktúrák leegyszerűsítésén még a jövőben dolgoznom kell.

A térképek, az azokon lévő pontok és útvonalak, illetve a felhasználói helyzet megjelenítéséhez az apple által biztosított MapKit keretrendszert használtam.

- RouteMapView
    - Sétákat megjelenítő térképes View
- LocationMapView
    - Helyszínt megjelenítő térképes View

## 4.4 Szolgáltatások, segéd osztályok

Az alkalmazási logikai rétegének megfelelő működéséhez létrehoztam még három osztályt. Ezek a projekten belül a Services mappában találhatóak meg. A következőkben ezeket fogom bemutatni röviden:

- LocationManager
  - Felhasználó helyzetének meghatározásáért és a telefon helymeghatározási engedélyeinek kezeléséért felelős osztály.
  - Fontosabb függvényei:
    - checkIfLocationServicesIsEnabled:
      Ellenőrzi, hogy a helymeghatározási szolgáltatások engedélyezve vannak-e.
    - getUserLocation: Visszaadja a felhasználó helyzetét, ha az elérhető.

- NetworkConnectionManager
  - Hálózati kapcsolat figyeléséért felelős osztály.
  - Inicializálásakor elindít egy NWPathMonitor típusú monitort, ami a hálózati kapcsolat változásait figyeli.
  - @Published var isConnected: A hálózati kapcsolat állapota itt tárolódik, és automatikusan frissül, amikor változás történik.

- RouteCreator
  - Használatával több helyszínből álló útvonalakat lehet létrehozni és tervezni a távolságok alapján.
  - Fontosabb változók és függvények:
    - @Binding var allRoutes: A generált útvonalakat tartalmazó tömb.
    - @Binding var distance: Az összesített távolság hossza az útvonalak alapján.
    - createMultiStopsRoute: A paraméterekben egy tömböt (amiben helyszínek vannak) és a felhasználó lokációját várja. Amennyiben a felhasználó lokációja nil értékű, a tömb első elemétől hozza létre az útvonalat. Amennyiben nem nil, akkor pedig a felhasználó helyzetétől tervezi meg azt. Mindkét esetben a tömbben szereplő összes helyszínt érinteni fogja.

## 4.5 Alkalmazás elindítása

Az alkalmazás belépési pontja a DiscoveryApp.swift file-ban található @main actor-nál.

Itt történik a Firebase konfigurációja, illetve a legtöbb ModelView osztály példányosítása is. A létrehozott viewModel példányokat környezeti objektumként továbbítom az alkalmazás többi része felé.

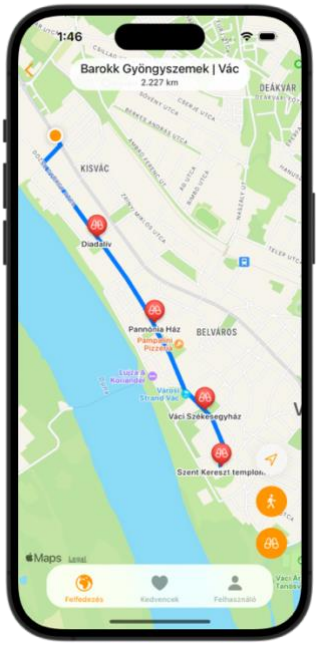
# 5 Összefoglalás, jövőbeni tervek

Az alkalmazást a funkcionális követelmények alapján sikeresen megterveztem és megvalósítottam. Kipróbáltam, teszteltem egy iOS alapú készüléken is és a tesztelés után elmondható, hogy az alkalmazás megfelelően működik.

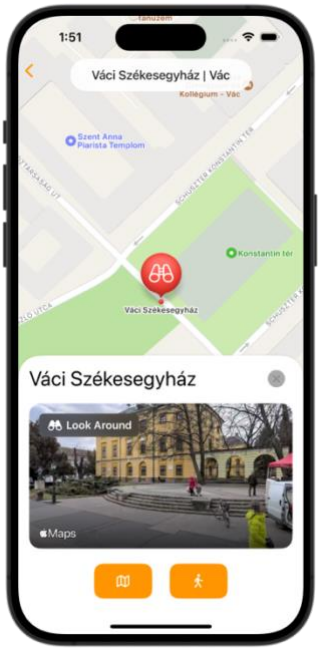
Úgy gondolom, hogy a megvalósított funkciók mellett még számos módon lehetne kiegészíteni az alkalmazást, hogy a felhasználók még jobb élményben részesülhessenek használata közben. A projekttel kapcsolatos jövőbeni terveim:

- Felhasználói funkciók bővítése
  - Séta értékelések.
  - Saját séták létrehozása, megosztása.
- Képgaléria
  - A helyszín részleteit megjelenítő nézetben szeretnék egy képgaláriát létrehozni.
- Navigációs funkciók bővítése
  - Megtekintett állomások megjelölése.
- Tesztek írása
  - Felhasználói felület helyes megjelenésének tesztelések.
  - Adatbázis kommunikációs tesztek írása.

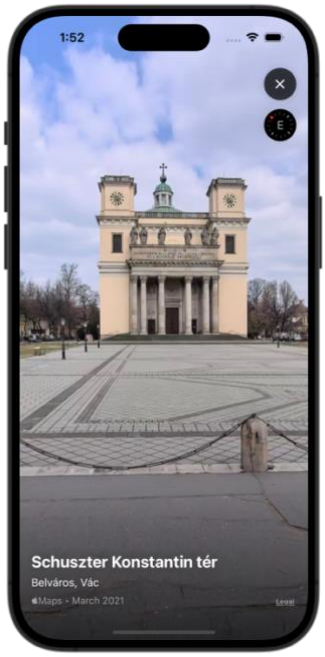
# 6 Függelék




*8. ábra: Térkép képernyő 1.*

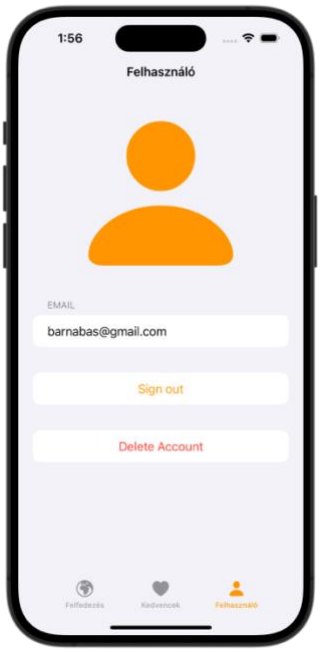



*9. ábra: Térkép képernyő 2.*

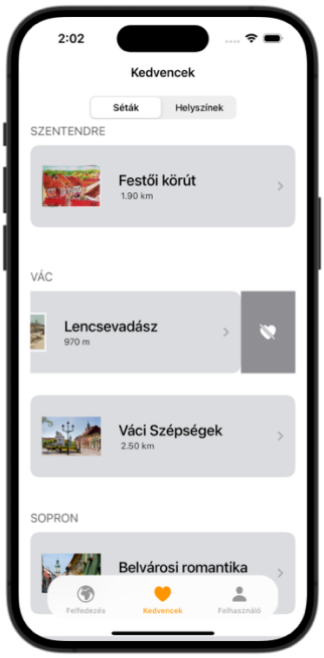



*10. ábra: LookAround képernyő*


*11. ábra: Regisztráció képernyő*




*12. ábra: Profil képernyő*




*13. ábra: Kedvencek képernyő*