Haladási napló – Diploma tervezés 2

Október 3.

Sajnos az Xcode-dal problémák adódtak, ezért előbb azt kellett újra telepíteni, de mivel ez sem oldotta meg a problémát, ezért újra telepítettem a gépet a macOS Sonoma frissítés után, úgy tűnik, ez jelenleg orvosolta a problémát.

Emiatt a képernyőkkel nem igazán tudtam haladni ezen a héten, viszont elkezdtem beleásni magam abba, hogy milyen lehetőségek vannak chat alkalmazások/funkciók megvalósítására, és kettő relevánsabb videót is találtam:

<https://www.youtube.com/watch?v=6v4fmg9iRSU>: régebbi videó (2020-as), storyboardos alkalamzást csinál benne, de bemutatja a MessageKit library alapjait, ami biztosítja az alapokat egy chat funkciót megvalósító alkalmazáshoz, viszont a csatorna sok érdekességet bemutat Swift fejelsztésben, pl. push notification használattal kapcsolatban is van videó

<https://www.youtube.com/watch?v=Zz9XQy8PRpQ>: friss videó, SwiftUI-ban építenek chat alkalmazást Firebase backendhez – a saját alkalmazásom szempontjából ez a relevánsabb találat, de ezen a csatornán is sok érdekes dolog van, leginkább az alkalmazás esztétikája szempontjából, illetve design tutorialok is vannak (Figma, Sketch)

Október 10.

Implementálásra került a chatnézet alapja. Ez az alap módosításra, illetve bővítésre kerül majd az üzleti logika implementálása során. A chat nézet alapja a következő komponensekből tevődik össze:

* létrehozásra került egy „ChatTitleRow” nevű osztály, amely a chat nézetben a beszélgetés headerjéül fog szolgálni. Ez az komponens megjeleníti a beszélgetésben részt vevő másik fél profilképét (jelenleg system image-ek vannak beégetve az adott helyeken), nevét, és amennyiben rendelkezik felhasználónévvel, akkor abban az esetben azt is. A profilkép fix mérettel, lekerekítve jelenik meg (frame; width: 50, height: 50) (cornerRadius: 50)
* létrehozásra került az üzenetbuborék megjelenítéséért felelős komponens („MessageBubble”), amely implementálásához szükség volt egy Model osztály létrehozására („ChatMessage”), ebben az üzenetek a következő property-kkel rendelkeznek: id (azért, hogy ForEach függvénnyel bejárható legyen az ilyen elemeket tartalmazó tömb), az üzenet, egy „received” boolean property, amely azt hivatott jelezni, hogy az adott felhasználó kapta-e az üzenetet (true esetben igen, false esetben nem) és egy „timestamp” mező. A „MessageBubble” példány eltárol egy ChatMessage példányt, majd attól függően rendezi bal vagy jobb oldalra, illetve színezi szürkére vagy narancssárgára a buborékot, hogy az adott user kapta, illetve küldte az üzenetet. Az üzenetbuborékokat 25-ös cornerRadius értékkel láttam el. A buborékra való tap gesztúrát követően a buborék alatt megjelenik az üzenet timestamp-je.
* A chat nézethez létrehoztam egy külön TextField komponenst is. Ehhez első körben létrehoztam egy „CustomTextField” osztályt. (Ennek az oka az volt, hogy ha a későbbiekben szükség volna még egyedi TextField osztályokra, akkor ezt a komponenst fel lehessen használni.) Ebben az osztályban eltárolásra kerül property-ként a placeholder szöveg, illetve @Binding változóként a begépelt szöveg (azért @Binding változó, hogy külső komponensből is módosítható legyen a tartalma), továbbá eltárolásra került két függvény típusú változó, az egyik az onEditingChanged esemény handlerje (boolean -> void típussal), a másik az onCommit esemény handlerje (void -> void típussal). Az osztály body property-jében elhelyeztem egy ZStack elemet, amelyben üres szöveg string esetén megjelenítésre kerül a placeholder, amelyre egy TextField került, amelybe a user beviheti a szöveget.
* A chatben használt bevitel kezelésére szolgáló komponensben felhasználtam az előző pontban leírt CustomTextField osztályt, ahol egyelőre csak a szöveges paraméterek lettek megadva a konstruktorban (üzleti logika hiányában még nem adtam értékül event handler függvényeket). A TextField széleit lekerekítettem és adtam neki egy bordert is, amely megjelenítéséhez egy workaround megoldást alkalmaztam: overlay modifiert alkalmaztam, amelyben értékül adtam egy RoundedRectangle elemmel. Elhelyezésre került egy gomb is, amely jelen pillanatban funkciót még nem lát el.
* Végül az elkészült komponenseket elhelyeztem a „ChatConversationView” komponensben, amely a teljes, beszélgetéshez tartozó képernyőt leírja. Jelen pillanatban dummy adatok szolgálnak a partner adataiként és az üzenetek adataiként is. A képernyő kialakításához létrehoztam a megfelelő VStack struktúrát, amelyekben elhelyeztem a korábban leírt elemeket. Az üzenetbuborékokat egy ScrollView-ban helyeztem el, így azok görgethetővé váltak.

Október 17.

Carousel view-hoz tutorial: <https://www.youtube.com/watch?v=6gAqdamj6tg>

Az adott szálláshoz tartozó képernyő implementálása következett, amelyre akkor jut el a felhasználó, ha a szállás listából kiválasztja az adott szállást. Ennek a képernyőnek az egyik fontos és talán legösszetettebb eleme a carousel, amelyben megjelenítésre kerülnek az adott szálláshoz feltöltött képek.

A carousel elemnek létrehoztam egy külön View komponenst, amelynek a CarouselView nevet adtam. Ez egy generikus komponens, amely rendelkezik egy View típusú „Contenttel”, valamint egy „Itemmel” és egy „ID-val”. Az „Item” megvalósítja a RandomAccessCollection protoclt, az ID pedig a Hashable és Equatable protocollokat. Property-ként a következő adatok vannak eltárolva: „content”, amely egy olyan closure, ami az „Item.Element” alapján ad vissza „Contentet”; „id” „KeyPath<Item.Element, ID>” típussal, illetve a kártyákhoz tartozó property-k mint „spacing” és „cardPadding” „CGFloat” típussal, items „Item” típussal, illetve egy @Binding Int típusú „index”, amely az aktuálisan kiválaszott Item.Element (kártya) indexe.