Haladási napló – Diploma tervezés 1

Március 19.

Elkezdtem implementálni a hiányzó képernyőket a kliens alkalmazásban. Ezt a folyamatot a bejelentkezési flow képernyőinek az implementálásával kezdtem. A bejelentkezési képernyő majdnem teljesen el is készült, csak sajnos egy picit elszoktam a SwiftUI keretrendszerrel való frontend implementálástól, és emiatt egy kicsit nehézkesen mentek az elemek elrendezései. A jelenlegi állapot sem végleges, próbálok egy olyan megoldást kitalálni, amellyel a majd a következő lépésben megvalósított alkalmazás logó beillesztése után a TextFieldeket tartalmazó List és a környezete hátterének a színe megegyezik. Illetve a List viselkedése is valamelyest megváltozott a verzió emelés következtében, ezt a későbbiekben figyelembe kell majd venni.

Illetve gondolkoztam a dipterv feladatkiírásának a szövegén, szerintem a következő megfelelő lenne:

Ha utazásról van szó, nem minden ember vágyik arra vagy engedheti meg magának, hogy hatalmas luxus szállodákban töltse el pihenését, illetve sokan úgy gondolják, hogy sokkal autentikusabb élményben lehet részük, ha megtapasztalhatják azt, milyen az adott országban eltölteni néhány napot egy helybéli szemszögéből szemlélve. Ezekre az esetekre kínálhat megoldást a couchsurfing.

A couchsurfing alapötlete az, hogy ha valakinek van egy felesleges szobája, ágya, kanapéja, akkor azt megoszthatja különböző utazókkal. Általában a vendéglátók nem csak a helyet biztosítják az utazóknak, hanem idegenvezetővé is válnak és megmutatják a kedvenc helyeiket az adott városban, cserébe illik valamivel meghálálni a házigazda vendégszeretetét.

A hallgató feladata egy olyan alkalmazás fejlesztése iOS platformon, ahol a regisztrált felhasználóknak lehetőségük van mások által kiadott szobát, kanapét lefoglalni. Bármelyik felhasználónak lehetősége van szobát kiadni, a felhasználók értékelhetik egymást, tudnak egymással chaten keresztül beszélgetni. Bizonyos szempontok alapján lehet szűrni a kiadó szobákra (pl. város alapján). Az alkalmazás képes értesítéseket küldeni a felhasználóknak.

A hallgató feladatának a következőkre kell kiterjednie:

* Mutassa be... (ennél a pontnál kellene egy kis segítség)
* Ismerje meg és röviden mutassa be a szükséges iOS technológiát!
* Tervezze meg az alkalmazás architektúráját!
* Valósítsa meg az alkalmazás kliens oldali és backend oldali komponensének prototípusát!

Március 26.

Ezen a héten elkészült az alkalmazás logója, illetve az app ikon – ezutóbbi skálázásával még vannak problémák, az ingyenes eszköz nem tette lehetővé, hogy több felbontásban is letölthető legyen az elkészített logó. Továbbá majdnem teljesen elkészült a bejelentkezési képernyő. A képernyőn szereplő formot ki kellett szervezni külön View fájlba, mert több alkalmazott modifier („scrollDisabled” és „scrollContentBackground”) csak iOS 16.0-tól volt elérhető, és ahhoz, hogy fallbackeljem alacsonyabb verziók esetén a formon alkalmazott modifiert, célszerűnek láttam újbóli deklaráció helyett csak egy példányosítást megvalósítani. Amire még rá kell jönnöm, hogy milyen módon lehet egy tartalmazott komponensből (jelen esetben a formból) eseményt „kibuborékoztatni” (Angular-os analógia az @Output property-kkel) – erre azért van szükség, hogy a formot befoglaló NavigationView megkapja a gombnyomás eseményt, és a bejelentkezés sikeressége után tovább navigáljon a flow-ban szereplő következő képernyőre. Vagy nem tudom, hogy ez az output szükséges-e egyáltalán, vagy a befoglalás miatt automatikusan érzékeli-e a Swift, hogy érkezett egy navigációra való kérés a gombtól, és emiatt tovább lép.

<https://medium.com/devtechie/disable-scroll-programmatically-in-swiftui-e3a380449f49>

<https://www.hackingwithswift.com/quick-start/swiftui/how-to-detect-dark-mode>

<https://stackoverflow.com/questions/57254461/how-to-change-the-background-color-for-a-form-in-swiftui>

<https://www.avanderlee.com/swift/dark-mode-support-ios/>

<https://stackoverflow.com/questions/56457395/how-to-check-for-dark-mode-in-ios>

Április 2.

Újabb képernyők készültek el. Elkészült az alkalmazás első indításakor megjelenő elosztó képernyő (ami kijelentkezés esetén is megjelenik). Illetve elkészült a fiókot létrehozó regisztráló képernyő is.

Amire megoldást kell találni:

* igényes scrollozhatóság a bejelentkező és regisztráló képernyőkön

További teendő, hogy az elosztó képernyő legyen az alkalmazás belépési pontja, a korábban elkészült, és amúgy a bejelentkezett felhasználó esetén belépési pontként szolgáló foglalási képernyő helyett.

A telefonszám megadós és módosítós mezőkhöz fel kell venni egy országot/országkódot kiválasztó Pickert, és akkor csak a textfieldben megadott számot könnyű lesz hozzáfűzni, ezáltal a csak numerikus billentyűzettel is bevihető a megfelelő adat és a user nem kap betűket -> kisebb az esélye a rossz formátum megadásának – ennek ellenére ellenőrizni kell, utólag, hogy csak numerikus-e a bevitt adat, és ha nem, akkor vagy error dialógusablakot kell majd feldobni akkor, amikor be szeretné küldeni az adatot, vagy azonnali validálás és jelölni, hogy nem megfelelő a formátum.

<https://stackoverflow.com/questions/39717964/how-to-get-todays-date-less-18-years>

<https://saikoneru.medium.com/working-with-dates-in-swift-ffabc22305db>

Április 16.

Teendő:

* Szobák árának a pénznemét is el kell tárolni az adatbázisban – lekérés esetén a frontendnek ezt megfelelően kell majd mappelnie, és feldolgoznia
* Az „OtherNeed” típusban lévő „name” mezőnek lehet, hogy érdemes volna bevezetni egy enum típust, ahonnan kiválasztható lenne, hogy milyen igényről van szó. Magába a típusba lehet, hogy nem lenne célszerű belenyúlni, csak egy nagyobb refaktor esetén, mert a többi mezőre szükség van for cikluson belüli iterációhoz stb.
* Backend válaszokban szereplő adattípusoknak megfelelő típusok bevezetése frontend oldalon, ezek használata a View osztályokban.

Bevezettem egy kisebb refaktort az „AccommListItemView” osztályban, amely az egyes szállások megjelenítéséért felelős a keresési eredmény listában. Először is bevezetésre került egy külön „Accommodation” típus, amely a backendről visszaküldött, szállással kapcsolatos infromációk frontend oldalon történő összefogásáért felelős. Ennek a típusnak egy példányát tárolja el az előbb említett View osztály, a korábban egyesével eltárolt megjelenítendő információk helyett. Bevezetésre került továbbá egy enum is, amely a lehetséges szállás típusokat hívatott biztosítani. (Nem tudom, hogy ezek a típus osztályok mennyire véglegesek – előfordulhat, hogy a backend fejlesztése során ezek még módosításra fognak kerülni.)

A refaktor után elkészítettem az „AccommListView” osztályt, amely a szabad szobák keresésekor indított backend hívás eredményében kapott listát hivatott megjeleníteni. Ehhez az előző bekezdésben leírt osztály egyes példányait használja, amelyeket az egyes „Accommodation” elemekkel példányosít. Az egyes példányokat egy „List”-ben jeleníti meg, amely pedig egy „NavigationView”-ba van ágyazva, hogy ezáltal megvalósíuljon az iOS-en jól ismert navigációs flow. Ennek megfelelően módosításra került a „BookingViewForm” osztályban (ami a foglalás adatainak megadásáért felelős) a „Keresés” gomb – a Button elem ki lett cserélve egy „NavigationLink” elemre, amely „destination” paraméterének meg lett adva az újonnan létrehozott lista oldal (egyelőre dummy adatokkal). Ennek a gombnak a megnyomása fogja elindítani a backend hívást, azután, hogy a megfelelő végpontok implementálásra kerültek.

Április 20.

Elkezdtem az alkalmazás backendjének a refaktorálását. Első körben létrehoztam egy „application.yaml” fájlt, amely a Spring alkalmazás indításakor az alkalmazás konfigurációjának a beállítását végzi, illetve az alkalmazás indítását hivatott elvégezni egy Docker konténerben. Ezután elkezdtem felülvizsgálni az maven projektleíró fájlban lévő dependenciákat és megszabadultam azoktól, amelyekre jelen pillanatban nincs szükség.

A refaktorálás következő lépéseként elkezdtem felülvizsgálni a már meglévő osztályokat, és megszabadultam azoktól a felesleges annotációktól, amelyekre jelen pillanatban (vagy úgy általánosságban nincs szükség).

Május 2.

Létrehoztam egy docker-compose.yml fájlt, amelyben definiálom az adatbázist. Ennek a fájlnak a futtatásával egy Docker konténerben hozom létre és futtatom az adatbázist a korábban létrehozott sémával. Továbbá létrehoztam a Spring alkalmazás „resources” mappájában egy adatbázis erőforrást: ez az SQL fájl felelős a táblák létrehozásáért. (Korábban a MySQL workbenchen történt a táblák létrehozása, és lokálisan futott az adatbázis. Ez a létrehozás került át a Spring alkalmazásba és az adatbázis futtatása ezentúl egy konténerben lesz megvalósítva.)

Ezen felül létrehozásra került minden táblának a megfelelő Entity (model) objektum. Ezen kívül felvételre került egy „PaymentMethod” nevű enum típus is, amely a lehetséges fizetési módokat írja le. A korábban létrehozott DTO osztályok jelenleg deprecated státuszba kerültek, mivel létrehozásra kerültek külön request és reponse objektumok az Entity objektumokhoz. Az eddig megvalósításra került entity-request-response típusokhoz létrehozásra kerültek mapper objektumok is. A request és response objektumoknál fontos odafigyelni arra, hogy elkerüljük a körkörös függőségeket (tehát, ha pl. létrehozunk egy RoomResponse példányt, akkor az abban lévő BookingResponse példányok is tartalmaznak referenciát ugyanerre a RoomResponse példányra). Ezeket a függőségeket majd még fel kell oldani, ennek a feloldásnak az egyik lehetséges útja, ha bevezetésre kerülnek olyan request és response osztályok, amelyek nem tartalmaznak referenciát arra a típusra, amelyikben megjelennek mint mező. Másik lehetséges megoldás, hogy csak az adott mező típusának az azonosítóját tartalmazza az adott típus. Ez viszont további végpont hívásokhoz és lekérésekhez vezethet abban az esetben, ha a felhasználó szempontjából fontos megjeleníteni az azonosítóval ellátott adat további tulajdonságait. Ez a probléma további átgondolást igényel.

Május 11.

Request jön a springben: security megfogja, security filter chainen végig kell mennie a requestnek ahhoz, hogy feldolgozzák -> meggyőződünk, hogy tényleg jó-e. Középen egy filter chain proxy, ide custom filter chaineket (OncePerRequestFilter interface-t implementálják) lehet tenni, hogy ezen is menjen át a request. Security config-ban filter chain (sessionmanagement az stateless, cors disable, stb), az általunk létrehozott filtert feliratkoztatjuk, ebben a filterben a request headerből kiszedjük a jwt tokent (bearer prefixű tokent), ha nincs lejárva és validációknak megfelel, akkor a username-t/email-t kiszedjük belőle. A kiszedett információval betöltjük -> user details-t kapjuk (interface implementáció), ezután új authentication-t létrehozunk. Végén a security contextben az authentication-t beállítjuk arra, amit létrehoztunk, majd tovább adjuk a filter chainben.

Május 21.

A kezdetleges Spring Security flow-n történtek módosítások. Most az alkalmazás indításakor ellenőrzi az authentikációban szereplő végpontokra intézett kéréseket a spring security, de jelenleg van valahol egy körkörös dependencia, amelyben a benne szereplő entitások körbe hívják egymást, amely egy Stack Overflow errort eredményez. A Spring Security a fentebb leírt JWT token alapú megvalósítást valósítja meg, amely a security filter chainben ellenőrzi a hívásban szereplő tokent, és ez alapján eldönti, hogy a kérés

Június 1.

Első lépésként a SecurityConfig fájlban a HTTP kérések authorizácjóában a lambda kifejezésben beállítottam, hogy minden kérést hagyjon jóvá („permitAll()” függvény hívása). Erre azért volt szükség, mert a Spring Security-ben valahol van egy körbe hivatkozás, amely StackOverflow exception dobását eredményezi, és ezzel a megoldással jelenleg működőképes az alkalmazás, a megvalósításra került végpontokat lehet hívni, és ki lehet őket próbálni. Az a terv, hogy miután minden végpont elkészült, a Spring Security configban visszakapcsolásra kerül a kérések authorizációja, amely során megoldásra kerül majd a jelenleg fennálló probléma. Addig ezzel a kis kerülővel tettem elérhetővé és kipróbálhatóvá a végpontokat.

Elkészült továbbá az „/api/room” végpont, amely az adatbázis Room táblájával való műveleteket hivatott végrehajtani. A végponthoz tartozó minden Spring komponens elkészült, és működőképes.

Megpróbálkoztam továbbá egy saját, „CustomEmailValidator” osztállyal, amely a HTTP kérésben kapott body-ban található „email” property-t hivatott vizsgálni (a „userRequest” osztályban lett emiatt alkalamzva), azonban ez az annotáció egyelőre nem a vártnak megfelelően működik, ezt majd még a jövőben ki kell javítani.