EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Média- és Oktatásinformatikai Tanszék

Pszichoterápiás ellátótérkép Spring és React alapokon

Bende Imre

Tanársegéd

Csuvik Gábor

Programtervező informatikus BSc

Budapest, 2021

Tartalomjegyzék

1. Bevezetés 5
1.1. A dolgozat felépítése 5
2. Felhasználói dokumentáció 7
2.1. A webalkalmazás elérése 7
2.1.1. Gépigény 7
2.2. Az alkalmazás főbb funkcióinak rövid ismertetése 8
2.2.1. Felhasználói jogosultságok 8
2.2.2. Intézménykereső funkciók 9
2.2.3. Tudásbázishoz kapcsolódó funkciók 9
2.2.4. Egyéb funkciók 10
2.3. A felhasználói felület bemutatása 10
2.4. Használói esetek 15
2.5. Felhasználói élmény 15
2.5.1. Különböző képernyőméretek 16
2.5.2. Offline-first megközelítés 20
2.6. Geolokáció 21
2.7. Userkövetés 21
2.8. Adatbiztonság 21
2.9. Észrevételek visszajelzése 21
3. Fejlesztői dokumentáció 22
3.1. Fejlesztői környezet 22
3.1.1. Virtualizációs lehetőségek 22
3.1.2. A forráskód 22
3.1.3. Függőségek a kliensoldali kódhoz 22
3.1.4. Függőségek a szerveroldali kódhoz 23
3.2. Frontend technológiák 24

3.2.1. A React könyvtár 26
3.2.2. Tesztelői eszközök 27
3.2.3. Állapotkezelés 27
3.2.4. Progresszív webapplikáció, offline first eszközök 30
3.2.5. Más fontos könyvtárak 33
3.3. Backend technológiák 33
3.3.1. Keretrendszer 33
3.3.2. Külső könyvtárak 35
3.3.3. Perzisztencia 36
3.3.4. Modellek, adatelérési (DAO) réteg 36
3.3.5. Kiszolgálók (Service-ek) 37
3.3.6. Vezérlő (Controller) osztályok 37
3.4. Autentikáció és autorizáció 37
3.4.1. A tokenek anatómiája 38
3.5. Végpontok dokumentációja 39
3.5.1. Megfontolások 39
3.6. Üzemeltetés 40
3.6.1. A szerveroldali kód automatizációja 41
3.6.2. A kliensoldali kód automatizációja 43
3.7. Tesztelés 44
3.7.1. Eszközök 44
3.7.2. Tesztesetek és teszteredmények 44
4. Összefoglalás 47
5. További fejlesztési lehetőségek 48
6. Mellékletek 50
6.1. 1. sz. melléklet: A szolgáltatókhoz kapcsolódó osztályok UML diagramja 50
6.2. 2. sz. melléklet: A szerveroldali alkalmazás docker-compose fájlja 51

1. Bevezetés

Segítő szakemberként több évet dolgoztam terápiás munkatársként a mentális egészségvédelem különböző területein, mielőtt informatikus pályát választottam. Munkám során gyakorta találkoztam olyan elképzelésekkel, amik meggátoltak nehéz élethelyzetekkel küzdőket abban, hogy segítséget kérjenek. A stigmatizációtól való félelmen és az ellátórendszerrel kapcsolatos információk hiányán túl a kapcsolatfelvételt az a közvélekedés is nehezíti, miszerint a pszichoterápiás ellátás kevéssé hozzáférhető, drága lehetőség. Szakdolgozatomban arra vállalkozom, hogy egy olyan rendszert fejlesszek, ami segít a rászorulóknak eligazodni az alapítványi, egyházi, valamint az állami, társadalombiztosítás által finanszírozott sokszínű intézményrendszerben, remélve, hogy így több ember elől hárul el a segítségkérés ilyenfajta akadálya. A cél megvalósulását az alkalmazás egy térképes intézménykeresővel, valamint egy információbázissal segíti.

Az alkalmazáshoz kapcsolódó teljes funkcionalitás fejlesztése túlmutat a szakdolgozati kereteken, de célom egy olyan nyilvánosan hozzáférhető rendszer átadása, ami segíthet a szakembereknek klienseik továbbításakor, illetőleg a rászorulóknak lehetőségeik megismerésében. A webalkalmazás elkészítésekor fontos szempont, hogy azt mind asztali, mind mobileszközökről használni lehessen, akár internetkapcsolat nélkül is. Ezt modern eszközök, technológiák segítségével érem el mind kliens-, mind pedig szerveroldalon. A kliensoldalon a felhasználó különféle felbontásoknak is megfelelő, reszponzív webdizájnnal találkozik, amit az első használat után a progresszív webalkalmazásokhoz szükséges technológiák segítségével internetkapcsolat nélkül is betölthet. A szerveroldalon felhőtechnológiák segítik az adatokhoz való gyors, megbízható hozzáférést. A technológiai célokon túl fontos számomra, hogy a projekt idővel önfenntartó, közösségileg adminisztrálható legyen. Ennek az alapjait is szeretném beleilleszteni ebbe a prototípusba. A későbbiekben a felhasználói igények, visszajelzések szerinti fejlesztés mellett ennek bővítése kerül a fókuszba, hogy egy valóban közösségi alkalmazás születhessen, amit könnyen és szívesen használ bármely rászoruló, vagy szakember egyaránt.

1.1. A dolgozat felépítése

Dolgozatom második fejezetében részletesen dokumentálom az alkalmazás használatát. Ez minden olyan olvasónak hasznos lehet, aki felhasználói oldalról találkozik az applikációval, vagy ha az alkalmazás felhasználói élményt illető megfontolásaiban

érdekelt. A részletesebb technikai kérdéseket, megvalósításokat a harmadik fejezet taglalja. Olyan olvasóknak szól ez a fejlesztői dokumentáció, akiket érdekelnek az alkalmazás mögötti informatikai megoldások, technológiai megfontolások, vagy akik fejlesztőként dolgoznának az alkalmazás további életútján. Ebben a fejezetben nem csak programozói, de üzemeltetői témákat is bemutatok. Az összefoglalást követően beszélek a fejlesztés következő mérföldköveiről, illetve azokról a nem-fejlesztői feladatokról, amik még az alkalmazás előtt állnak, hogy egy valóban jól használható, fontos eszköz lehessen a magyarországi mentális egészség-védelemben.

2. Felhasználói dokumentáció

Ebben a fejezetben részletesen vázolom a Pszi-Infó alkalmazás használatát, főbb funkcióit, szereplőit és jogosultságaikat. Végezetül kitérek a felhasználói élmény releváns vonatkozásaira.

2.1. A webalkalmazás elérése

A kliensoldali webalkalmazás böngészőből a https://pszi.info címen érhető el. Első betöltést követően internetkapcsolat nélkül is működőképes. A modern böngészők mind asztali, mind mobilos eszközökön lehetőséget adnak a program telepítésére, hogy azt a felhasználó natív alkalmazásként is használhassa. A szerveroldali kéréseket az https://api.pszi.info szolgálja ki. Mindkét végpontot érvényes SSL tanúsítvány védi, hogy az adatküldés biztonságos maradhasson kliens és szerver között.

2.1.1. Gépigény

Az alkalmazás minden nagyobb böngésző minden támogatott verziójában fut, eléréséhez az első alkalommal, valamint a frissítések telepítéséhez internetkapcsolat szükséges. A modernebb operációs rendszerek, böngészők lehetőségeket kínálnak a webalkalmazás elhelyezésére a kezdőképernyőn, vagy natív alkalmazásként való telepítésére és használatára is. Ezek a lehetőségek operációs rendszer és böngészőfüggők, javasolt a felhasználó tájékozódása eszközei képességeit illetően.

A fejlesztéshez és teszteléshez használt operációs rendszerek és böngészők verziói:

- Windows 10 Home 20H2:
  - Google Chrome 90.0.4430.212
  - Microsoft Edge 90.0.818.66
  - Mozilla Firefox 88.0.1
- Ubuntu 20.04.2 LTS:
  - Google Chrome 90.0.4430.212
  - Mozilla Firefox 88.0.1
- Android 10 QKQ1.190828.002:
  - Chrome for Android 90.0.4430.91

2.1.1.1. Támogatott képernyőméretek

A minimális képernyőméret 280px szélességű, de ilyenkor egyes funkciók (pl. cikk szerkesztése) csak alapfunkciókkal használhatók. 360 képpont feletti szélességnél már

minden funkció használható. Az okostelefonokon túl a táblagépek, laptopok, monitorok felbontásai is megfelelő használatot biztosítanak, akár ultrawide képerány mellett is.

2.2. Az alkalmazás főbb funkcióinak rövid ismertetése

Az applikációnak két fő funkciója van; egyrészt intézménykeresőt, másrészt tudásbázist biztosít. A többi funkció ezekhez kapcsolódó. Mielőtt részletesebben beszélek a lehetőségekről, ismertetem az alkalmazás felhasználó jogosultságait, hogy a webalkalmazás funkcionalitásainál rámutathassak a jogkörökhöz kapcsolódó megkötésekre.

2.2.1. Felhasználói jogosultságok

A prototípusban két jogosultsági kör szerepel, ami a későbbiekben továbbiakkal fog kiegészülni. A jogosultsági körök:

2.2.1.1. Látogató

A látogató böngészheti az intézményeket és az információs portált. Nem szerkesztheti az adatokat, csak olvasási jogköre van.

2.2.1.2. Adminisztrátor

Szerkesztheti mind az intézményeket, mind a cikkeket. Újakat hozhat létre, és törölhet elemeket.

2.2.1.3. További tervek

Az ütemtervben szerepelnek további jogkörök, amelyek a közösségi szerkesztési elvet valósítják meg:

- A *felhasználó* jogkör a látogató jogosultságait intézmények ajánlásával egészíti ki, valamint szerkesztési javaslatok tételével a már meglévő intézményeket illetően
- A *szerkesztő* elbírálhatja a tett javaslatokat, valamint változtathat intézményi adatokat, és új intézményt adhat meg
- A *szerző* cikkeket írhat

Az adminisztrátor szerepkör így főleg jogosultságkezelésre, üzemeltetéshez kapcsolódó feladatokra szűkül. Felhasználó regisztrációt követően válik valakiből. Megfelelő kontribúciót követően ajánlja fel neki a rendszer a szerkesztővé válást. Szerzővé kapcsolatfelvétel, megkeresés útján válhat valaki. Ez a rendszer még kidolgozásra, implementálásra szorul, de fontos része lesz az oldal működésének és filozófiájának.

2.2.2. Intézménykereső funkciók

Az oldal alapfunkciója a pszichiátriai- és pszichoterápiás intézménykereső. A következő lehetőségeket támogatja az alkalmazás:

- Intézmények, ellátóhelyek részletes keresése szűrők alapján. Jelenleg támogatott szűrők:
    - Település
    - Előjegyzés szükségessége
    - Beutaló szükségessége
    - Várólista hossza
    - Van-e sürgősségi ellátás
    - Célcsoport
    - Csak telefonszámmal rendelkezők
    - Csak email címmel rendelkezők
    - Csak weblappal rendelkezők
    - Szűrés keresőszó alapján
- Az intézmények, találatok térképen való megjelenítése, kattintható jelölőkkel, amik több információt adnak az ellátóról
- Az intézmények, találatok listában való megjelenítése, ahol további információk olvashatók az egyes ellátókról
- Térképen talált intézmény listában való kiemelése
- Listában talált intézmény térképen való kiemelése
- Listában talált intézmény felkeresése telefonon, emailben vagy webcímen
- Intézmények keresése a keresőmező használatával
- Az adminisztrátor számára elérhető az intézmény hozzáadása egy külön oldalról
- Az adminisztrátor törölheti az intézményeket a listanézetből indulva

2.2.3. Tudásbázishoz kapcsolódó funkciók

Az alkalmazás másik fontos pillére a tudásbázis, ami szakemberkereséshez, problémák felismeréséhez, krízishelyzetek kezeléséhez ajánl hasznos, közérhető tanácsokat, forrásokat. Az ehhez kapcsolódó funkciók:

- A tudásbázis böngészése
- Cikkek keresése a keresőmező segítségével
- Kiválasztott cikk olvasása

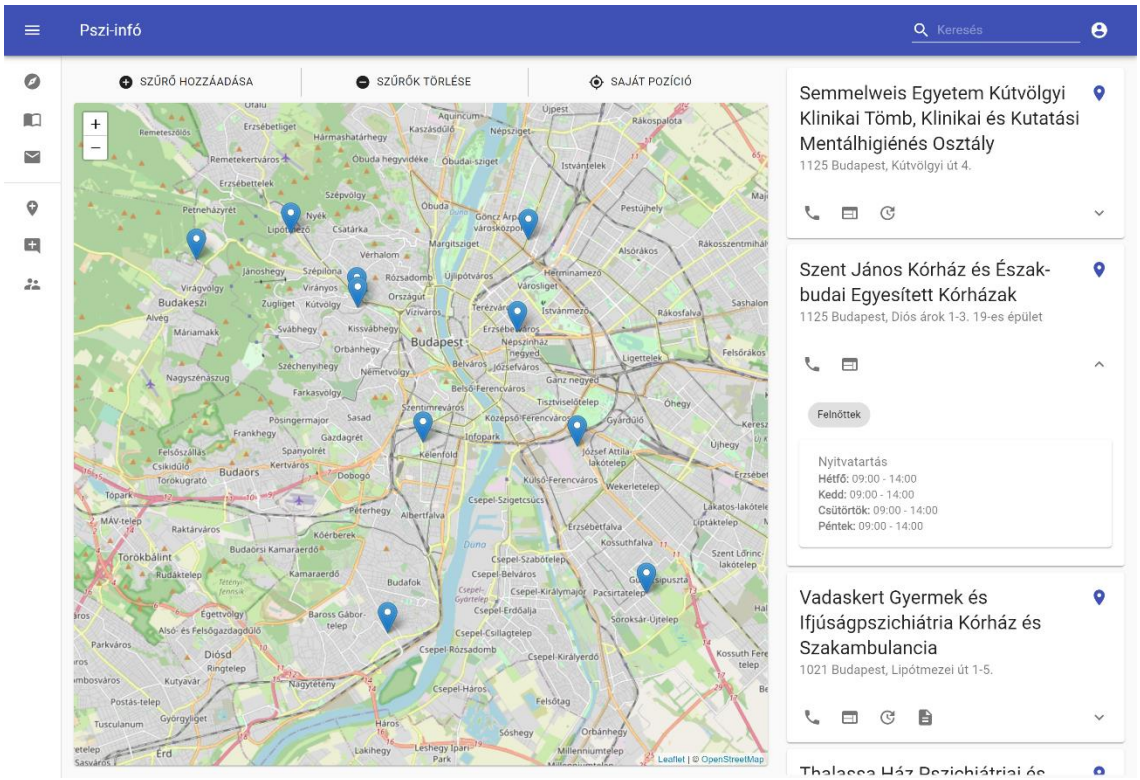
- Az adminisztrátornak lehetősége van cikkek hozzáadására
- Az adminisztrátornak lehetősége van cikkek törlésére

2.2.4. Egyéb funkciók

Az alkalmazás lehetőséget ad emailcímmel és jelszóval való bejelentkezésre, valamint kapcsolatfelvételre a megadott email címen keresztül.

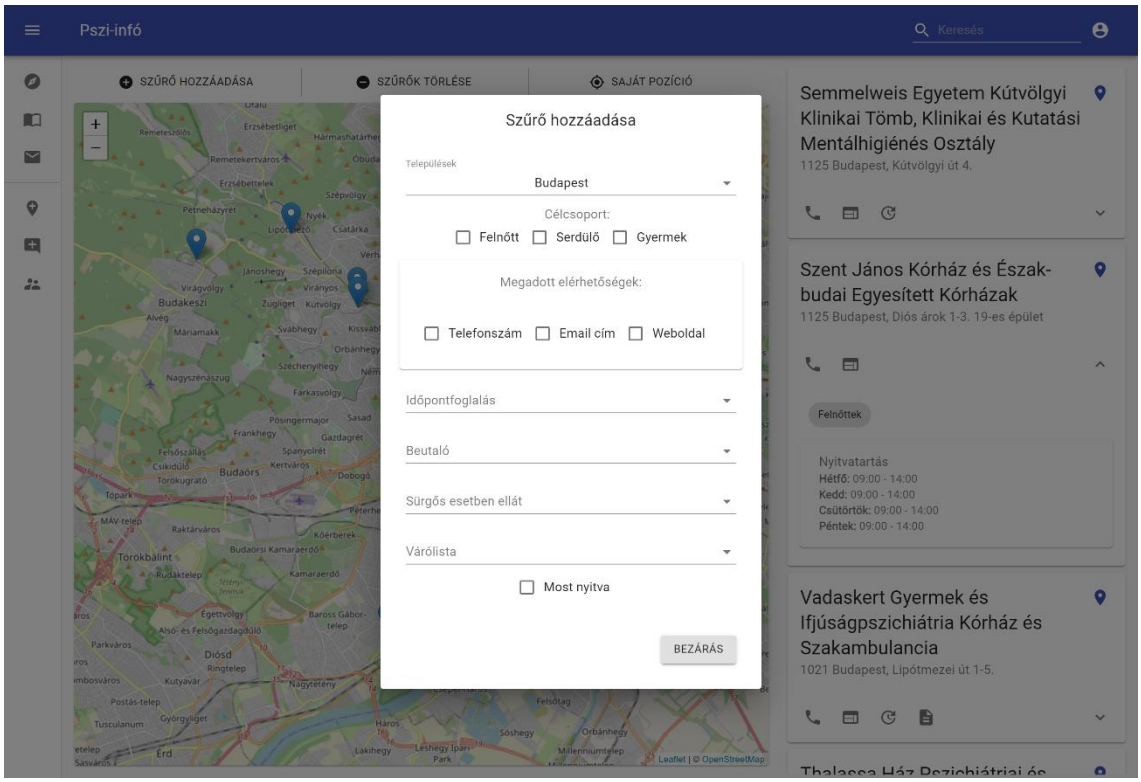
2.3. A felhasználói felület bemutatása

Képernyőképek segítségével vázlatosan ismertetem az elérhető fontosabb nézeteket. A képernyőképek 1440x900 képpontos felbontás mellett készültek. Interaktív módon a https://pszi.info címen ismerhető meg az alkalmazás.

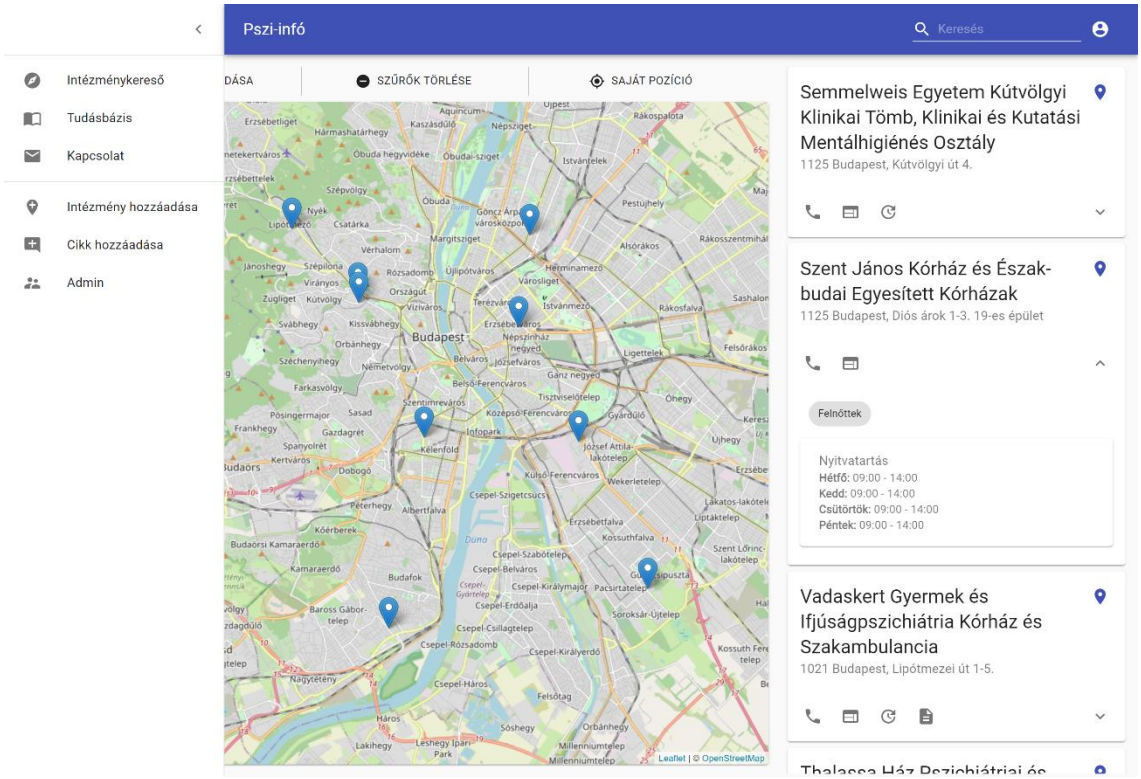


ábra 1. – Intézménykereső

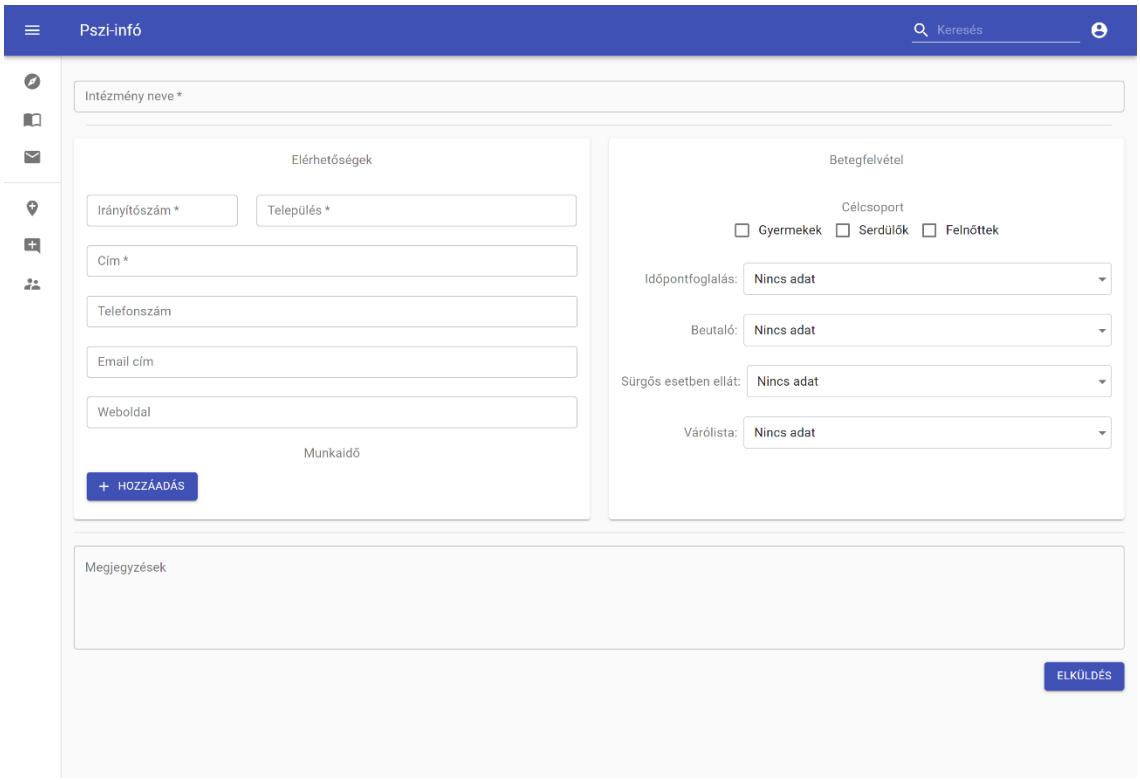
Az intézménykereső bal oldalán felül a szűrők kezelése, és a saját földrajzi pozícióra való közelítés szerepel, alatta pedig a térkép látható. Jobb oldalon az intézménylista szerepel. Más felbontásokon ez az elrendezés ettől eltérő, hogy jobban segítse az adott képernyőméreten az applikáció használatát. Erről példák később szerepelnek.



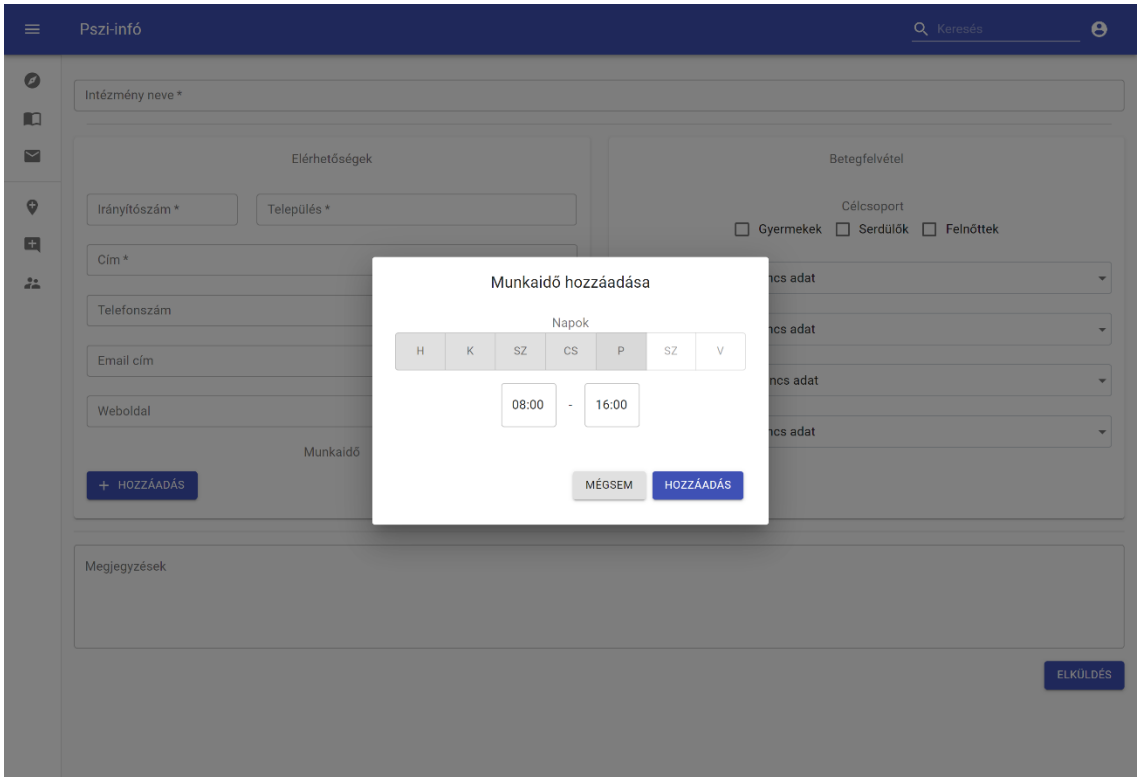
ábra 2. - Intézménylista szűrői



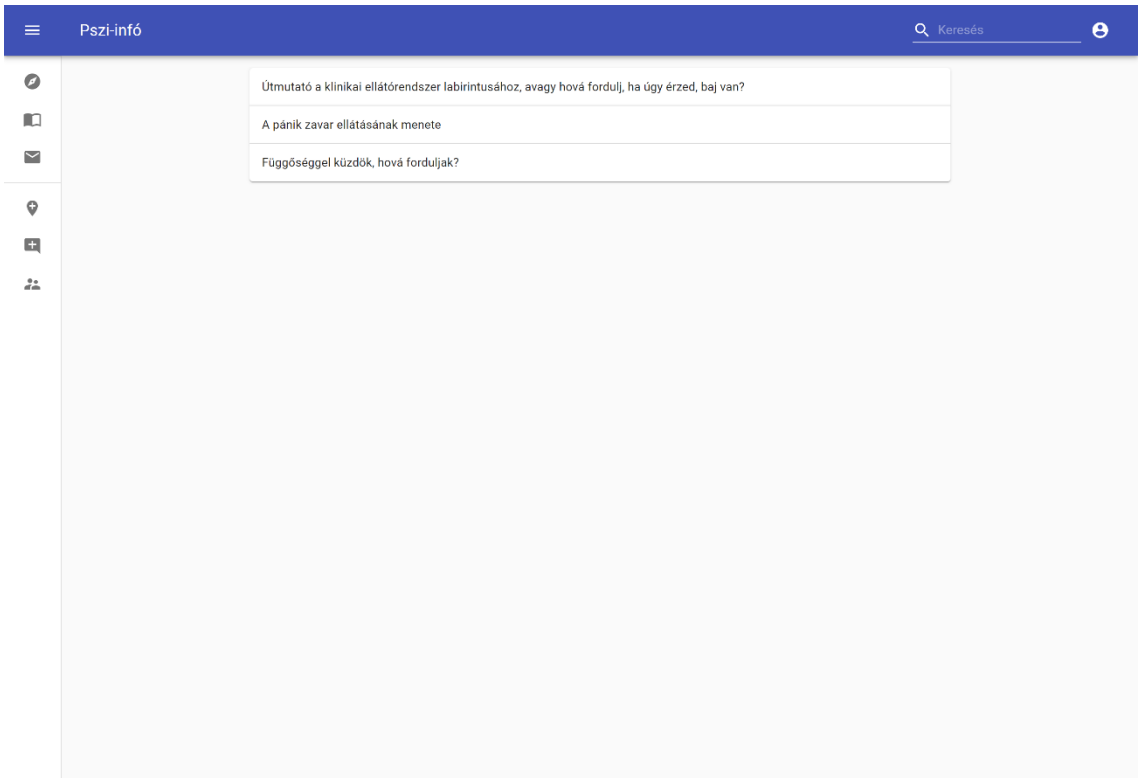
ábra 3. - Nyitott navigációs oldalsáv (balról)



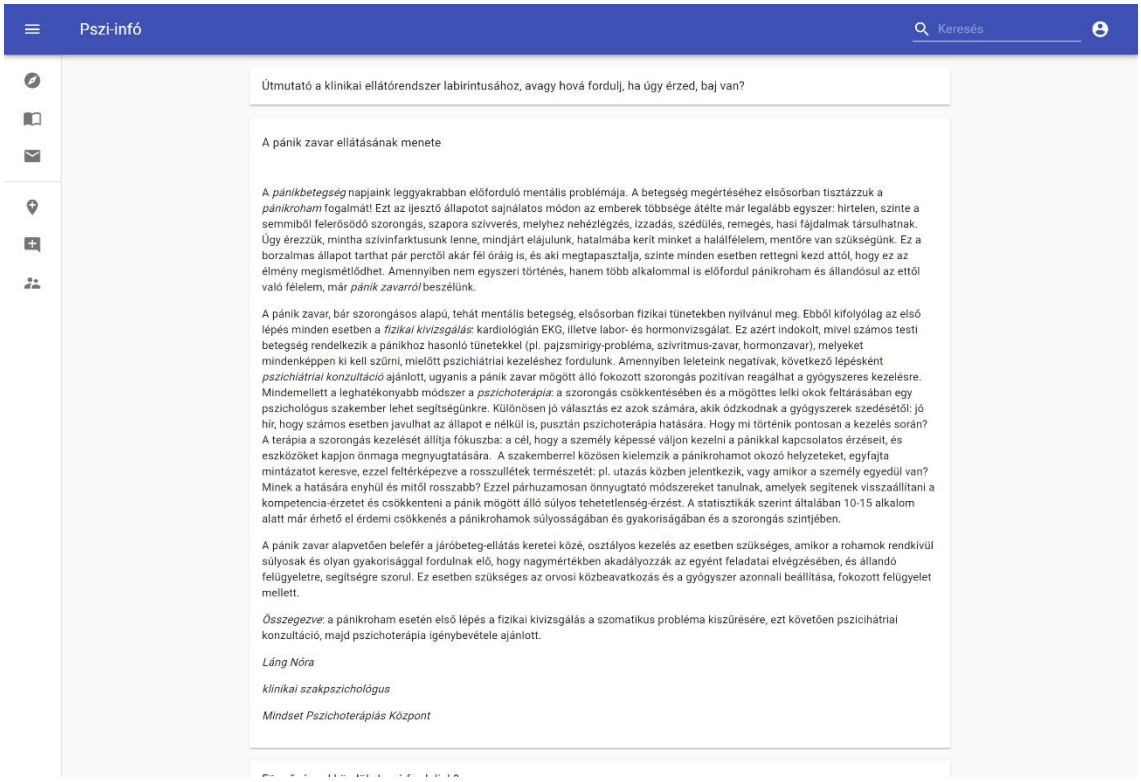
ábra 4. - Intézmény hozzáadása



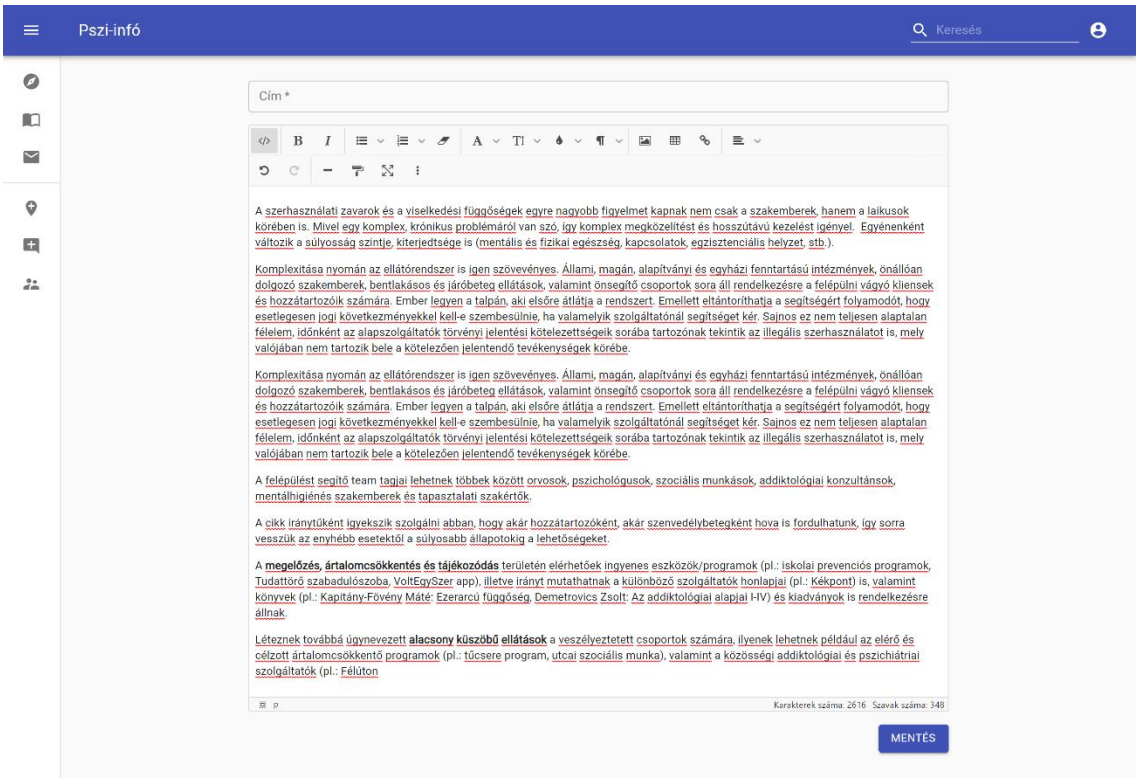
ábra 5. - Munkaidő hozzáadása



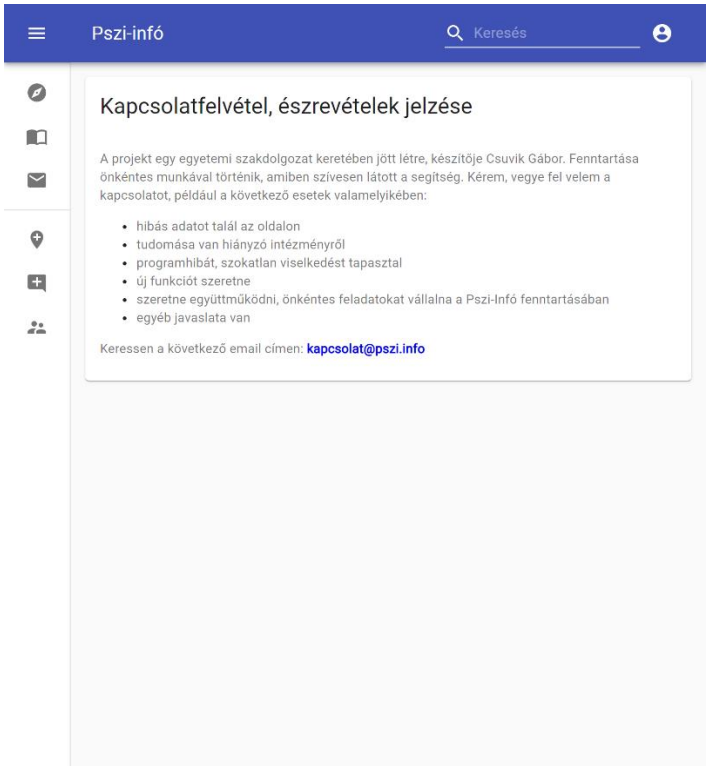
ábra 6. - Tudásbázis főképernyője becsukott cikkekkel



ábra 7. - Tudásbázis főképernyője nyitott állapotban

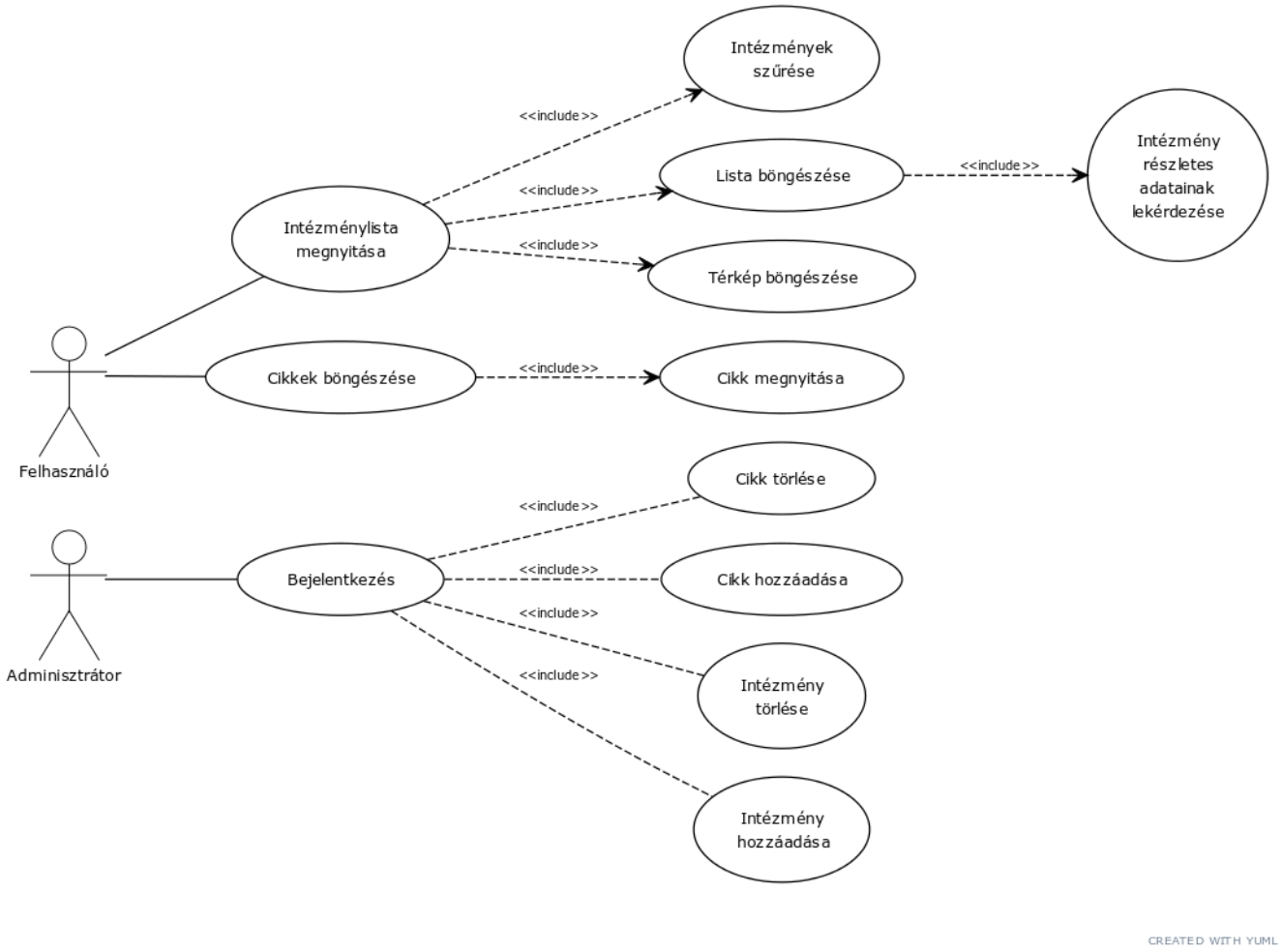


ábra 8. - Cikk hozzáadása



ábra 9. - Kapcsolatfelvételi oldal

2.4. Használói esetek



ábra 10 - Használói eset diagram

2.5. Felhasználói élmény

Az alkalmazás tervezésénél kiemelt jelentőségű volt, hogy egy intuitív, jól használható, korszerű felületet kapjon a felhasználó, ami megfelel a modern elvárásoknak. A mai internethasználók gyakran mobileszközökről is fogyasztanak tartalmakat, sőt, ez a trend drasztikusan erősödik: az elmúlt tíz évben az elérések aránya mobileszközökről 6.09%-ról 55.56%-ra nőtt.[1] Ez új kihívások elé állítja a fejlesztőt. Ha nem natív mobilalkalmazással támogatják az asztali alkalmazásokat, olyan rugalmas, dinamikus webalkalmazásokat kell előállítani, amik a legkülönfélébb képernyőméreteken is használhatók maradnak. Az

[1] „What Percentage of Internet Traffic Is Mobile? [Feb 2021]”. https://www.oberlo.com/statistics/mobile-internet-traffic (elérés máj. 02, 2021).

internetkapcsolat ugyanakkor gyakorta instabil, vagy időszakosan nem elérhető. Ezekhez a modern problémákhoz kerestem olyan modern eszközöket, amik ilyenkor is segítenek a felhasználóknak az adatelérésben.

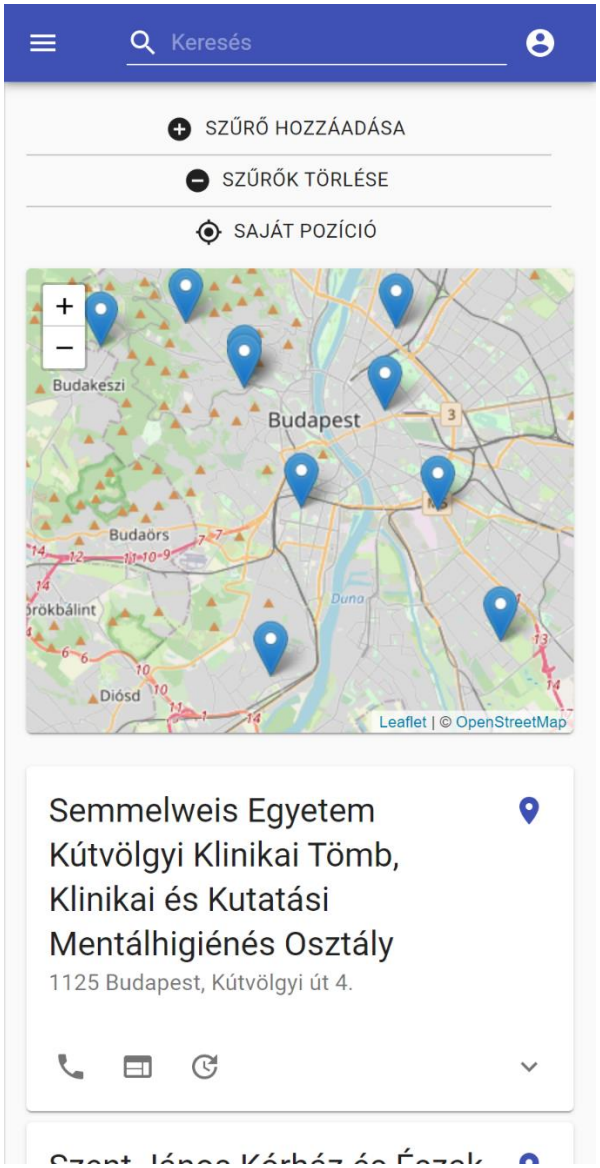
2.5.1. Különböző képernyőméretek

Az alkalmazás optimálisan legalább 280 képpont szélességű képernyőn fut. Ezalatt jelentősen romlik a felhasználó élmény. Az oldal dizájnja, elrendezése nem csak a kis kijelzőket, de a szintén modern, nagyfelbontású, 4k vagy ultrawide kijelzőméretet is támogatja, jelentőségteljesen használva a rendelkezésre álló helyet. A következő töréspontok mentén változik az oldal elrendezése:

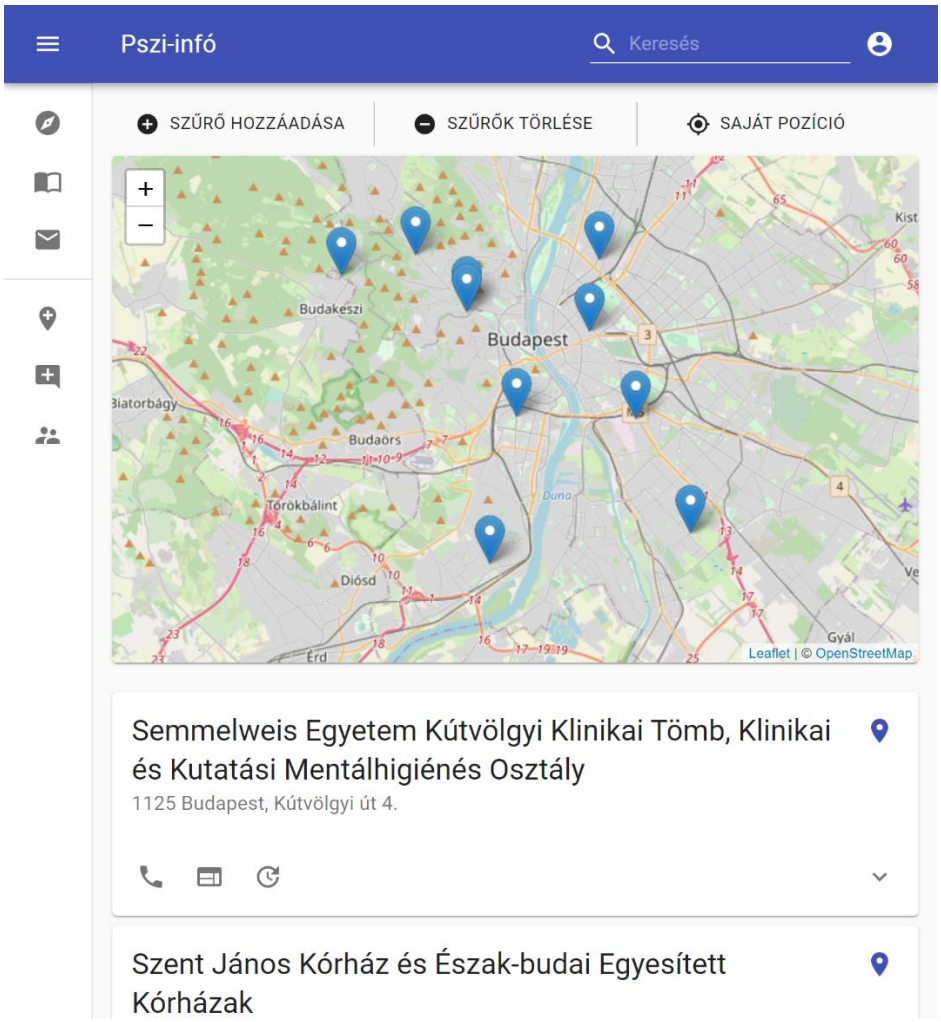
táblázat 1 - Töréspontok

Töréspont neve	*Szélesség felső határa*	*Felhasználás*
xs	600 képpont	Telefonok, okostelefonok
sm	960 képpont	Tabletek
md	1280 képpont	Tabletek, laptopok, kisebb monitorok
lg	1920 képpont	Laptopok, monitorok
xl	-	Nagyméretű monitorok

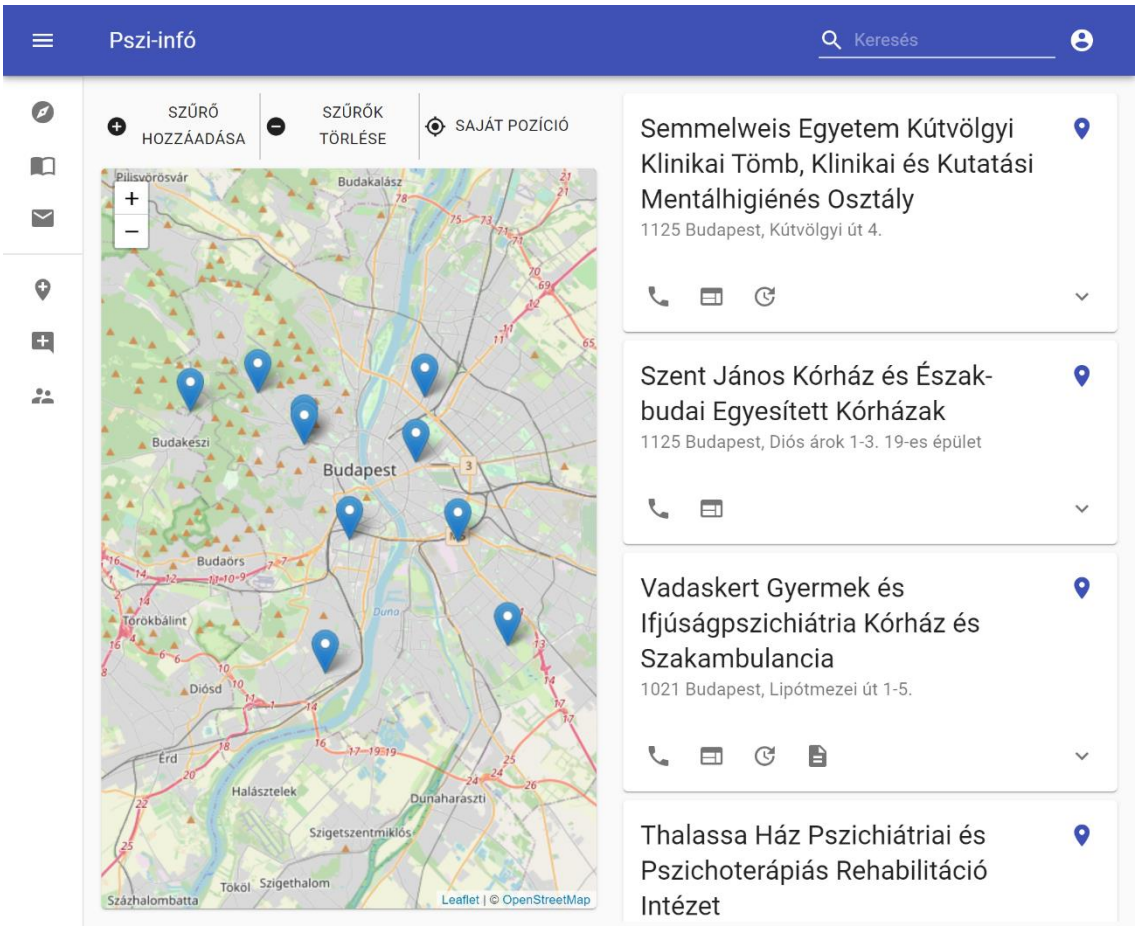
2.5.1.1. *Képernyőképek a különböző felbontások szemléltetésére*



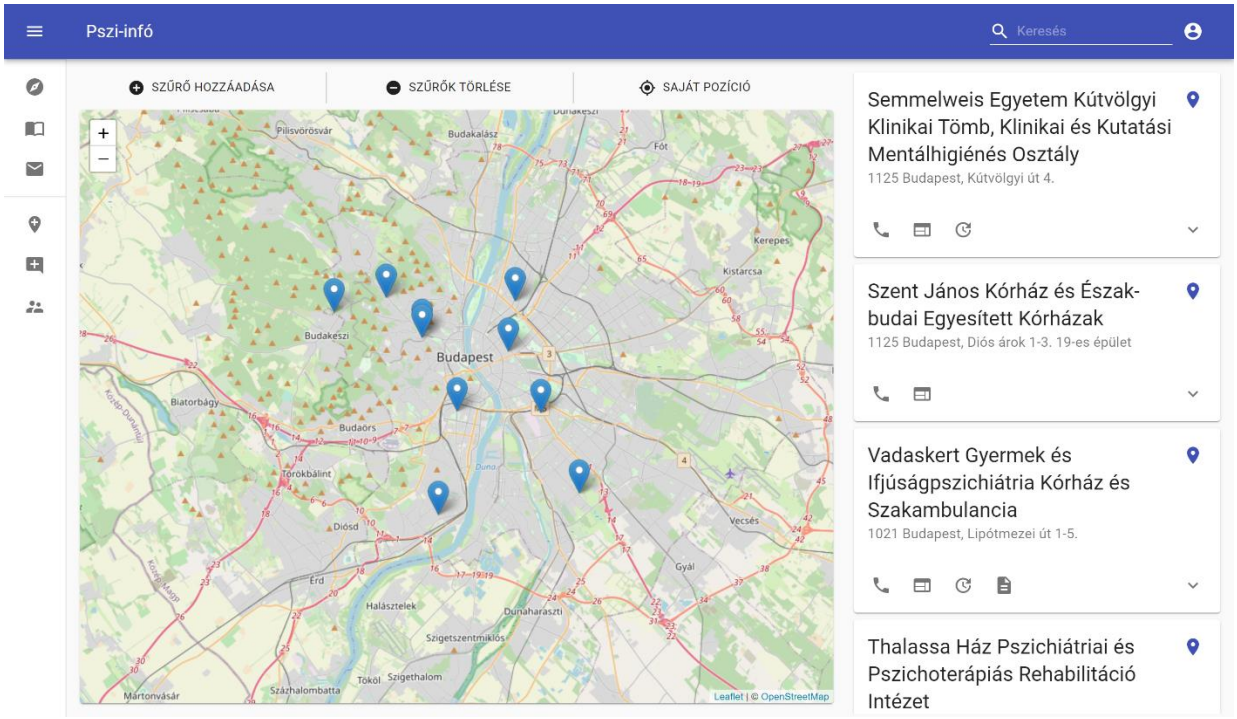
ábra 11. - Intézménykereső XS felbontásban



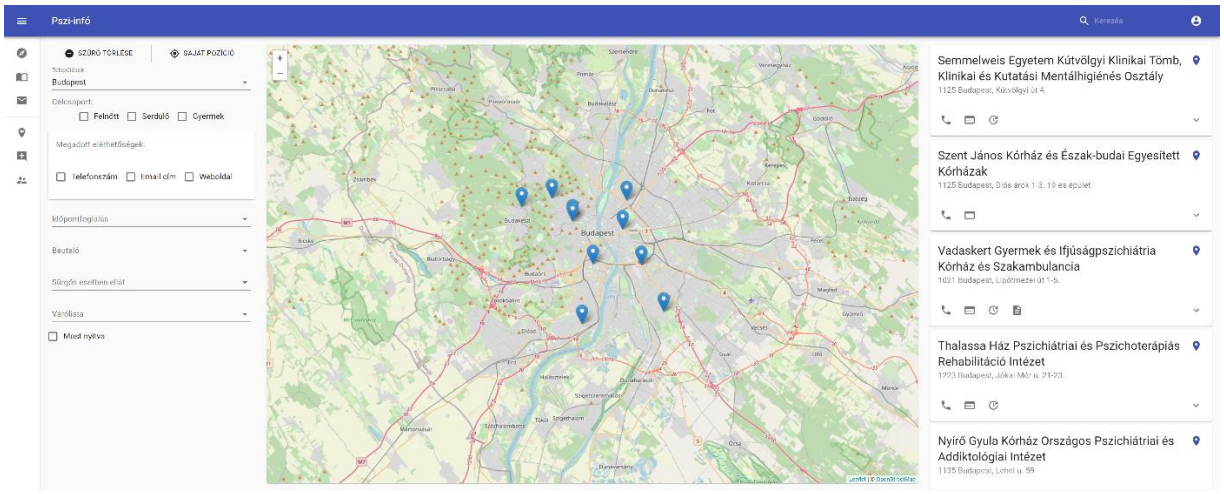
ábra 12. - Intézménykereső SM felbontásban



ábra 13. - Intézménykereső MD felbontásban



ábra 14. - Intézménykereső LG felbontásban



ábra 15. - Intézménykereső XL felbontásban

2.5.2. Offline-first megközelítés

A progresszív webalkalmazások térnyerése annak köszönhető, hogy akár egy vonatúton, változó jelminőség mellett, vagy akár lejárt adatkerettel is használhatók, biztosítva a folyamatos, megszakítás nélküli hozzáférést. Ilyenkor a tartalmak lokális tárban helyeződnek el, a frissítések pedig a legközelebbi internetre csatlakozáskor töltődnek le, például ha a felhasználó wifi-közelbe kerül. Fontos ésszerű keretek között tartani, mennyi adatot tárol az alkalmazás, hiszen az könnyen nőhet olyan méretűre, amit már nem vesz szívesen a felhasználó egy általa csak ritkán látogatott oldaltól. További előnye, hogy a modern operációs rendszerek támogatják a progresszív webalkalmazások natív alkalmazásként való telepítését, ami a teljesképernyős módon túl a legújabb API-knak köszönhetően valóban natív alkalmazásként tud működni, pl. küldhet értesítést a felhasználónak a háttérből is. Mivel szigorúbb biztonsági előírásoknak kell megfelelnie (például kizárólag biztonságos, https csatornán tölthet le adatokat), biztonságosabb is, és a gyorsítótárba töltött tartalmak miatt rövidebb betöltést is biztosít általában.

A tárolt adatokon túl a technológia hátránya, hogy a változtatások, fejlesztések nem azonnal tükröződnek a felhasználó eszközén, hiszen az oldalbetöltéskor még a korábbi service worker gyorsítótára érvényesül. Háttérfolyamatban töltődik le az új verzió, ami akkor veszi át a régi helyét, amikor az már nem szolgál ki betöltött oldalakat többé.

Az alkalmazásban a Google Workbox megoldásai segítették az üzleti standardok szerinti implementációt. Az offline-first élményről első sorban a Service Worker és a Cache API gondoskodik. Ezekről bővebben a fejlesztői dokumentációban írok.

2.6. Geolokáció

A felhasználónak lehetősége van saját pozíciójának megjelenítésére az intézménykereső térképen. Ehhez engedélyeznie kell a helyzetmeghatározást. Az engedély ezután visszavonásig érvényes. A visszavonásra natív eszközök is rendelkezésre állnak a böngészőkben, eszközökön.

2.7. Userkövetés

Az oldal a Google Analytics analítikáját használja annak követésére, milyen interakciók történnek felhasználó és webalkalmazás között.

2.8. Adatbiztonság

Az alkalmazás személyes adatokat nem kezel. A tartalmak teljes mértékben hozzáférhetőek regisztráció nélkül is. Az ott megadott email cím a felhőben, a Firebase rendszerében kerül tárolásra.

2.9. Észrevételek visszajelzése

Az oldal kapcsolat fülén feltüntetett kapcsolat@pszi.info elérhetőségen lehetőség van kapcsolatfelvételre, javaslatok tételére, visszajelzés küldésére.

3. Fejlesztői dokumentáció

A fejezetben a Pszi-Infó alkalmazás fejlesztői rétegeit mutatom be, többek között kitérve a projekt telepítésére, a használt technológiák bemutatására, az alkalmazásspecifikus osztályok és kapcsolatok ismertetésére, valamint az üzemeltetés szempontjából releváns témákra.

3.1. Fejlesztői környezet

A fejlesztéshez IntelliJ IDEA Ultimate integrált fejlesztői környezetet használtam, de nem szerepel a kódbázisban olyan projektfájl, ami megkívánná a program jelenlétét. Tetszőleges IDE használható, ami támogatja mind a Java, mind pedig a JavaScript kódot, és lehetőség szerint kellő támogatást is nyújt a React és a Spring keretrendszerekhez.

3.1.1. Virtualizációs lehetőségek

A projektben Dockerfile található mind a kliensoldali, mind a szerveroldali részhez, illetve docker-compose file-ok a fejleszői és az éles környezethez. Ezért a fejlesztéshez javasolt a docker virtualizációs környezet használata. Amennyiben ez nem teljesül, natív módon is futtathatók a függőségek, a Dockerfile-okban található konfigurációk és parancsok mentén. A dokumentációban hagyatkozom a docker jelenlétére.

3.1.2. A forráskód

A forráskód publikusan elérhető a GitHub verziókezelőn[2]. Letöltéséhez git szükséges.

3.1.2.1. A kód struktúrája

A kliensoldali kód a ui könyvtárban, míg a szerveroldali kód az api könyvtárban van. Két különálló projekt van az adattárban így, amiknek a telepítése, használata, fejlesztése egymástól függetlenül is lehetséges virtualizáció segítségével, amiről később írok. A gyökérkönyvtárban alapszintű dokumentáció, valamint virtualizációs leírók vannak a docker-compose orkesztrátor eszköz számára.

3.1.3. Függőségek a kliensoldali kódhoz

A projekt fordításához, fejlesztői futtatásához node (v14+) és a vele csomagolt npm szükséges. A react-scripts parancsaira hivatkozom a futtatáshoz, teszteléshez, és az éles verzió csomagolásához.

[2] https://github.com/csuvikg/pszi-info

táblázat 2 - Főbb parancsok a kliensoldali kódhoz

parancs	*funkció*
npm build	a package.json-ben leírt projekt buildelése
npm start	fejlesztői szerver indítása a 3000-es porton
npm test	automatizált tesztek futtatása interaktív módban
npm run build	éles verzió csomagolása

3.1.3.1. Futtatás konténerben

Lehetőség van docker konténerként is futtatni a kliensoldali alkalmazást. Olyankor nem igényel telepített node-ot, vagy a dockeren kívül más függőséget, de kevés interaktív lehetőséget ad a fejlesztéshez. Akkor javasolt így használni az alkalmazást, ha valaki csak a szerveroldali kódot fejleszti, de azt a kliensoldali kezelőfelülettel szeretné kipróbálni. A ui könyvtár Dockerfile állományával építhető meg a konténer.

3.1.3.2. Szoftvertelepítés a szerverre

Ugyan a master ágra kerülő, sikeres tesztekkel futó kód automatikusan települ a szerverre, lehetőség van manuális telepítésre is. Az automatizált megoldásról később írok. A manuális módhoz szükség van a firebase parancssori eszközre, és megfelelő szintű hozzáférésre a Firebase projekthez. Ilyenkor az *npm run build* parancs elkészíti az éles csomagot, a *firebase deploy --only hosting:pszi-info* pedig telepíti a szerverre. Ezután az új verzió azonnal elérhetővé válik.

3.1.4. Függőségek a szerveroldali kódhoz

A projekt a JDK 15-ös verzióját használja, ennek megléte szükséges a kód fordításához. A build eszköz gradle, ami futtatható a gradlew fájllal. A szerveroldali kódhoz is tartozik Dockerfile, ilyen módon elégséges a futtatáshoz docker is. Az adatbázis PostgreSQL, ami natívan is telepíthető, de a projekt gyökerében van egy fejlesztői docker-compose fájl, ami megfelelően felparaméterezett adatbázist indít konténerben. Ezek segítségével fut az API szerver is, aminek a docker-compose file-ja szintén a gyökérkönyvtárban van. Igényel egy megfelelően kitöltött .env file-t is, amit mellette kell létrehozni. Példa egy konfigurációra:

táblázat 3 - Környezeti változók a szerveroldali kódhoz

kulcs	*érték*
POSTGRES_HOST	db
POSTGRES_PORT	5432
POSTGRES_DATABASE	pszi_info
POSTGRES_USERNAME	postgres
POSTGRES_PASSWORD	tetszőleges jelszó
GOOGLE_MAPS_API_KEY	Google Maps API kulcs

Lokális fejlesztéshez ezeket a környezeti változókat be kell állítani. A modern fejlesztői eszközök biztosítanak ehhez egyszerű lehetőségeket. Látható, hogy szükséges Google Maps API kulcs a teljes funkcionalitáshoz. Ezt a szerveroldali kód a geocodinghoz használja, a címek koordinátákká alakításához. Enélkül is futtatható a kód, ha ez a funkcionalitás nem fontos.

A projektben Lombok van a boilerplate kód (getterek, setterek, konstruktorok, stb.) generálásához, így fejlesztéshez be kell kapcsolni az annotation processing beállítást.

3.1.4.1. Szoftvertelepítés a szerverre

A verziókezelőben a master ágra kerülő kódból automatikusan docker konténer épül. Erről bővebben később írok, a folyamatos intergrációról (CI, continuous integration) szóló résznél. A telepítés további része manuális, az éles docker-compose fájlt kell elindítani a szerveren, hogy az letöltse az új konténert, és a megfelelő beállításokkal (a .env fájlban előkészített, a szerveren elhelyezett változókkal) elindítsa azt. A REST API a https://api.pszi.info címen érhető el.

3.2. Frontend technológiák

A kliensoldali felhasználói felületért, logikáért a React JavaScript könyvtár, míg a tárhelyszolgáltatásért a Google Cloud és a Firebase felel.

A React kiváló lehetőséget biztosít a modern, komponensalapú tervezéshez, fejlesztéshez. Ez az elképzelés biztosítja a kód modularitását, fenntarthatóságát, tesztelhetőségét és újrafelhasználhatóságát. A kód struktúráját illetően fontos megjegyezni, hogy Reactben a HTML és a JavaScript nem válik szét, a jsx formátum lehetőséget ad a HTML sablonok JavaScriptbe való beágyazására, például:

```
const element = (
  <div>
    <h1>Cím</h1>
    <h2>Alcím</h2>
  </div>
);
```

Ami aztán html-be renderelhető:

ReactDOM.render(element, document.getElementById('root'));

Ez némileg szokatlan lehet első hallásra, de a megközelítés elsajátításával jól strukturált, átlátható kód írható. A stílusok, stíluslapok kezelésére manapság többféle megoldás is létezik. Én a CSS-in-JS megközelítés mellett döntöttem (pl. styled-components, emotion), és a @material-ui/styles könyvtárat használtam fel. Ilyen módon nagyfokú dinamizmust, örökölhetőséget, ütközésmentes stílusdefiníciókat biztosító érhető deklaratív leírók felelnek a stílusokért, melyekből végül hagyományos CSS generálódik. Példa:

```
import { makeStyles } from '@material-ui/core/styles';
const useStyles = makeStyles({
  root: {
    backgroundColor: 'red'
  }
});

export const MyComponent = () => {
  const classes = useStyles(props);
  return <div className={classes.root} />;
}
```

A program egy egyoldalas alkalmazás (SPA, single-page application) a modern trendeknek megfelelően. Ezzel együtt a funkcionalitásukban különálló oldalak eltérő útvonalon könyvjelzőzhetők, hash alapú útvonalválasztással, ami a # jel utáni részt (URL fragment) használja az routingnál. Mivel ezt a részt a böngésző figyelmen kívül hagyja, így nem igényel szerveroldali konfigurációt az ilyen útvonalválasztás, és biztosan visszafelé kompatibilis. A fragment mindazonáltal JavaScriptben hozzáférhető, feldolgozható. Az implementációhoz a React Router könyvtárat használtam. Példa URL:

https://pszi.info/#/providers/add

Az alkalmazásban a globális állapotokat a Redux könyvtárral használtam, melyet a Thunk middleware-rel egészítettem ki, hogy kezelhessek benne aszinkron hívásokhoz kötődő eseményeket is. A felhasznált technológiákról külön-külön részletesebben is írok.

3.2.1. A React könyvtár

A React a manapság legnépszerűbb kliensoldali könyvtár[3], melyben egyre több frontend-fejlesztő szerez tapasztalatot. Kezdeti népszerűségét a virtuális DOM ötletének köszönheti, melynek köszönhetően az igazán költséges DOM-manipulációkra csak akkor került sor, ha azokra valóban szükség volt. A Facebook inkubációjaként biztosított volt a túlélése, fejlődése. A komponensalapú ideológiája tesztelhetővé, újrafelhasználhatóvá tette a kódot. Állapotkezelési modellje érhető, befogadható volt a fejlesztők számára. Az idők folyamán kialakult, mára jelentőssé duzzadt közössége, ökoszisztémája pedig továbbra is fenntartja a könyvtárat övező népszerűséget.

Kódomban a 2019-es évben bevezetésre került hookokat (eseményhorgokat) használtam, ami egy újabb interfész a React belső mechanizmusaihoz. Ezzel az eszközzel funkcionális komponensek készíthetők, melyek a JavaScript modern nyelvi elemeit jól hasznosítják. Példa:

[3] „State of JS 2020: Front-end Frameworks”. https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/ (elérés máj. 02, 2021).

```
import { useState } from 'react';

const Example = () => {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>{count} alkalommal kattintottál!</p>
      <button onClick={() => setCount(count + 1)}>
        Kattints ide
      </button>
    </div>
  );
}
```

A useState hookról az állapotkezelés alatt írok.

3.2.2. Tesztelői eszközök

A testing-library könyvtárra hagyatkoztam, mely lehetőséget kínál jól bevált eszközök integrált használatára. A kódban a *.test.js osztályok tartalmazzák az egységteszteket, amiket a CI automatikusan futtat is. Inkább csak demonstrációs jelleggel kerültek a kódba a kliensoldali tesztek, és csak kevéssé jó lefedettséget biztosítanak. A manuális tesztelésé volt a főszerep fejlesztés közben.

3.2.3. Állapotkezelés

A komponensek szintjén háromféle állapot különböztethető meg:

- Belső állapot:
    - A React terminológiájában state, melyet a useState hook kezel. Ezt a belső állapotot a React biztosítja komponensei számára. Változása kiváltja a komponens újrarenderelését. Példát fentebb közöltem, a React könyvtár általános ismertetésénél.
- Külső állapot:
    - A React terminológiájában props, a komponens argumentumként kapott változói. HTML attribútumokként adhatók át a jsx-ben a komponensnek.

Változása szintén rendereléssel jár. A prop-types könyvtár lehetőséget biztosít típusellenőrzésre, amelyet kódomban én is használok. Példa:

```
import PropTypes from 'prop-types'

const HelloWorldComponent = ({ name }) => {
  return (
    <div>Üdv, {name}</div>
  )
}

HelloWorldComponent.propTypes = {
  name: PropTypes.string
}
```

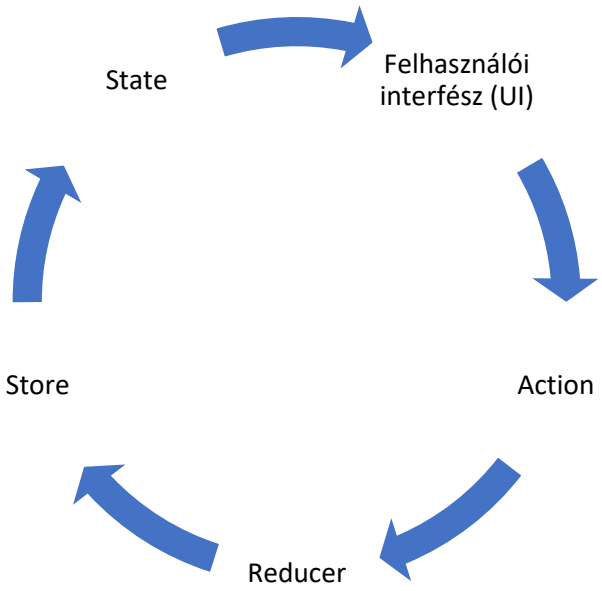
- Ebben a példában a props a funkcionális komponens paraméterében megkapott objektum (itt a name), melynek típusjelölését (PropTypes.string) a kód második fele biztosítja. Hozzáadható továbbá az .isRequired is, ha kötelezővé akarjuk tenni a komponens argumentumát.

- Globális állapot:
  - Globális állapotként arra a külső állapotra hivatkozok, amelyet nem közvetlen módon, a szülőn keresztül kap a komponens. Ehhez a kódban a redux könyvtárat használom, aminek működése igen bonyolult, ezért külön alfejezetben tárgyalom részletesebben.

3.2.3.1. Redux

A redux a megfigyelő (observer) tervezési mintát megvalósító alkalmazásállapot-tároló, melyben actionnek hívott eseményeken keresztül módosítható az állapot. Ilyen módon egy központi tároló, ami az alkalmazás bármely komponenséből elérhető. Ezen túl segíti az alkalmazás debugolhatóságát úgy, hogy láthatóvá, visszajátszhatóvá, visszaállíthatóvá teszi az állapotváltozásokat. Reactben a redux használatához szükség van a react-redux adapterhez is, ami eszközöket ad a redux store-jához kapcsolódó manipulációkhoz.

Szigorúan definiált, hogyan módosulhat a reduxban tárolt állapot:



ábra 16 - Adatfolyam a Reduxban

Az ábrán feltüntettem a redux elemeit. A felhasználói interfész a React alkalmazás. Valamilyen itt definiált eseményhez kötötten váltható ki akció (action), melyhez használható a react-redux könyvtárban található useDispatch() hook is.

A dispatch egy actiont vár, aminek van típusa (type), valamint lehetnek ezen felül tetszőleges tagjai, amin keresztül adatokat lehet továbbítani rajta keresztül (payload). Az action a reducerbe érkezik meg, ahol megváltoztatja az állapotot a store-ban. A fenti példában használt useSelector hook ezt az állapotot olvassa ki a komponensen belül.

A példakód nem teljes, az action és a reducer leírása, valamint a store inicializálása hiányzik belőle. Az egyszerűség kedvéért ezeket nem tárgyalom, a react-redux dokumentációjában könnyen hozzáférhetők más, könnyen érthető példákon keresztül[4]. A következő, mellékhatásokról szóló részben bemutatok egy összetettebb actiont.

3.2.3.2. Mellékhatások, aszinkronicitás kezelése

A redux alapvetően nem alkalmas aszinkron akciók végrehajtására. Ezen probléma megoldására több middleware is született. Kódomban a Redux Thunk használata mellett döntöttem. Segítségével lehetőség van például ilyen actionök létrehozására:

[4] „Getting Started | React Redux”. https://react-redux.js.org/introduction/getting-started (elérés máj. 02, 2021).

```
export const createProvider = provider => async dispatch => {
    dispatch(createProviderRequest());
    try {
        await fetch("https://api.pszi.info/providers", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify(provider)
        });
        dispatch(createProviderSuccess())
    } catch (error) {
        dispatch(createProviderFailed(error.message));
    }
}
```

Látható, hogy az action lehet aszinkron, és hívhat más actionöket is. A createProviderRequest például beállíthatja az isLoading állapotot true-ra, ehhez pedig köthető a UI-on egy töltésjelző spinner. Az aszinkron hívás után annak eredményétől függően más action hívódik, amik hamisra állítják az isLoading értékét a reducerben. Látható továbbá, hogy az action kaphat argumentumot (itt a provider), és adhat is tovább a benne hívott actionnek (pl. az error.message-et).

3.2.4. Progresszív webapplikáció, offline first eszközök

Az alkalmazás az első betöltést követően offline is elérhető. Ezen felül modern környezetben lehetőség van a telepítésére is, okostelefonon és asztali számítógépen egyaránt. Ezeket olyan korszerű, szabványos API-k biztosítják, mint a Service Worker, Cache, IndexedDB, Local Storage.

Ahhoz, hogy egy webalkalmazás progresszív webalkalmazás (PWA) legyen, három minimális előfeltétel teljesülése szükséges:

- Biztonságos, https kapcsolaton keresztül kell kiszolgálni a tartalmakat rajta
- Legalább egy service workert kell telepítenie (erről később)
- Tartalmaznia kell egy webmanifest fájlt, például:

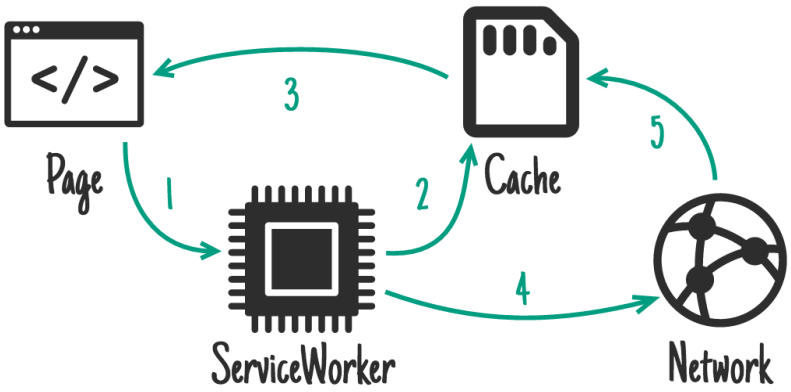
```
{
    "name": "Pszi-Infó",
    "short_name": "Pszi-Infó",
    "icons": [
        {
            "src": "/android-chrome-192x192.png",
            "sizes": "192x192",
            "type": "image/png",
            "purpose": "any maskable"
        },
        {
            "src": "/android-chrome-512x512.png",
            "sizes": "512x512",
            "type": "image/png"
        }
    ],
    "theme_color": "#3f51b5",
    "background_color": "#3f51b5",
    "start_url": "https://pszi.info",
    "display": "standalone"
}
```

Ez a fájl ad utasítást ahhoz, hogyan kell alkalmazásként telepíteni a webapplikációt. Megadja az alkalmazás nevét, felsorolja a használható ikonokat, megadja a színeket, amiket a natív alkalmazás használhat töltőképernyő, ablakkeret színezéséhez, és így tovább.

3.2.4.1. Service Worker API

Az API olyan interceptorok megírását teszi lehetővé, amik szabályozzák a böngésző hálózati kéréseit és caching szabályait az oldalra vonatkozóan. Lehetőséget teremt gyors

betöltődésű oldalak, vagy teljesen offline élmények létrehozására. Többféle mintázatban is használható, a stale-while-revalidate minta felépítése a következő[5]:



ábra 17 - Adatfolyam a Service Workerben

Ilyenkor az oldal a service workerhez fordul egy asset lekérdezéséért (pl. egy CSS fájl, vagy egy https kérés a backend felé), amit először a service worker cache-ből próbál kiszolgálni. Ha nem sikerül, a hálózathoz fordul, ami frissíteni fogja a gyorsítótárat is.

Mivel a service worker verziózott, és változások után frissíteni kell, előállhat olyan helyzet, hogy a gyorsítótárazott változatot tölti be a felhasználó számára az oldal a friss helyett. Ilyenkor egy háttérszálon letöltődik a legújabb verzió, ám nem kerül használatra, amíg nem zárja be a felhasználó az oldalt. Következő betöltésnél már a legújabb verzió áll majd rendelkezésre.

Alkalmazásomban a Google Workbox service worker könyvtárát használtam.

3.2.4.2. Cache API

Az API lehetőséget ad kérés-válasz párok eltárolására, biztosítva, hogy a kéréseket az internet helyett a gyorsítótárból szolgáljuk ki. Az invalidálásért, frissítésért a fejlesztő felel. Jól összefésüléhető az API használata service workerekkel is.

[5] „Workbox Strategies”, *Google Developers*. https://developers.google.com/web/tools/workbox/modules/workbox-strategies (elérés máj. 02, 2021).

3.2.4.3. IndexedDB API

Az API egy alacsonyszintű, kliensoldali tár, mely strukturált adatok mellett binárisokat is tárolhat. Tranzakcionális hozzáférést biztosít. Alkalamzásomban a Firebase API-jainak adatait tárolom benne.

3.2.4.4. Local Storage

A window.localStorage API lehetővé teszi az oldal originjéhez kapcsolódó Storage objektum kezelését. Az itt tárolt kulcs-érték párok tovább élnek a böngésző munkameneténél, biztosítva, hogy új betöltésnél is el tudjuk érni a korábban eltárolt adatokat. Alkalmazásomban a cache-ek verzióit tárolom itt, annak eldöntésére, kell-e új adatokat letölteni a backendről.

3.2.5. Más fontos könyvtárak

Alkalmazásomban központi szerep jut a térképes keresésnek. Ehhez az OpenStreetMap térképét használom, a Leaflet könyvtárral. A felhasználó pozíciójának meghatározásához a Geolocation API-t használom.

A dátumkezelésre Day.js-t használok. Jodit, és a jodit-react adapter biztosítja a wysiwyg editort, ami a cikkek szerkesztését segíti.

Az alkalmazásban Firebase-t használok a hostinghoz, az analitikához és a felhasználókezeléshez. A függőségek letöltéséhez, és API-jaival való interakciókhoz a Reactfire könyvtárat használom. Az alkalmazás a jövőben elhagyja majd a jelenlegi backendjét, Firebase adatbázisra és serverless backendre fog támaszkodni.

3.3. Backend technológiák

Ebben a fejezetben röviden bemutatom, milyen technológiák biztosítják az alkalmazás szerveroldali működését.

3.3.1. Keretrendszer

Egy Spring Boot keretrendszerben futó Java program szolgálja ki a szerverre érkező kéréseket. A továbbiakban az azt felépítő könyvtárakról, konfigurációjukról, az alkalmazásban jelenlévő fontosabb osztályokról lesz szó.

3.3.1.1. REST API

A szerveroldali alkalmazás egy REST API-t szolgáltat a kliensoldali alkalmazás felé. Ez egy olyan szoftverarchitektúra típus, mely lehetővé teszi, hogy a backend és a frontend egymástól függetlenül létezzen, és egy HTTP protokollra épülő adatcserével

kommunikáljanak egymással valamilyen leírónyelv szerinti (pl. XML, JSON) üzenetek segítségével. Az ilyen üzenetek előnye, hogy mind számítógépek, mind emberek számára jól olvashatók. A HTTP lehetőségeinek széles eszköztárát használja a REST, szemantikai jelentőséget társítva az elérhető HTTP-igékhez, válaszkódokhoz. Az évek alatt több ajánlás is született arra vonatkozóan, hogyan érdemes felépíteni az ilyen végpontokat. Az erről szóló résznél bemutatom a megfontolásokat, amik mentén elkészítettem az alkalmazás interfészét.

3.3.1.2. Spring Boot

A Spring Boot a Spring keretrendszer kiegészítése. Mivel a Spring az évek folyamán nagyon komplex, összetett rendszerré vált, egyre összetettebbé vált a konfigurálása is. Mivel a keretrendszer moduláris, a felhasználó dönthet arról, melyik elemeit használja alkalmazásában. A projektek esetén azonban tetemes mennyiségű munkát igényel a modulok igények szerint való összekonfigurálása. Ezt a problémát oldja meg a Spring Boot, olyan könyvtárak létrehozásával, amik az adott modult alapértelmezett konfigurációs osztályokkal tartalmazzák. A starter csomagoknak nevezett függőségek így nem igényelnek további konfigurációt, ha alapértelmezett módon használja őket a felhasználó. Mivel a Spring meghatározza a kód struktúráját (opinionated), bemutatom külön alfejezetekben az adatelérési (DAO), a kiszolgáló (service) és a vezérlő (controller) osztályokat.

3.3.1.3. Használt modulok

Mivel a Spring moduláris, és a fejlesztőre bízott, mely részeit használja, így ismertetem, alkalmazásomban mely elemek kerültek integrálásra.

3.3.1.3.1. spring-boot-starter-actuator

Az Actuator modul az alkalamzás monitorozását, kezelését segíti. Nem a felhasználókat, hanem az adminisztrátorokat támogatja. Az alkalmazásban a healthcheck végpontját használom. Ha Spring alapú marad a szerveroldali kód, a Prometheus pull végpontját is használni fogom.

3.3.1.3.2. spring-boot-starter-cache

Lehetőséget ad a Spring gyorsítótárazásának használatára. Néhány annotációval beállítható, melyik metódusok visszatérési értéke kerüljön cache-be, és milyen feltételek mellett invalidálódjon.

3.3.1.3.3. spring-boot-starter-data-jpa

A Spring Data JPA a Spring Data perzisztencia-kezelés része, és lehetővé teszi a JPA-alapú adattárak (repository-k) létrehozását. A könyvtárban Hibernate-támogatás is van a JPA adattárak kezeléséhez.

3.3.1.3.4. spring-boot-starter-oauth2-resource-server

Mivel az alkalmazás autentikációjáért első sorban a Firebase felel, így a Springnek resource server üzemmódban kell futnia. Ezáltal autentikációs tokeneket nem ad, ellenben képes azok hitelességének ellenőrzésére. Gyakori felhasználás ez mikroszerviz-architektúrák esetében. Én azért így valósítottam meg ezt a réteget, mert terveim között szerepel a jövőben nagyobb szerepet adni a Firebase-nek, és serverless backendet használni a kérések kiszolgálására. Ez úgy vélem, sokban segítené a fenntarthatóságot. Egy későbbi, dedikált fejezetben részletesebben írok az autentikációs rétegről.

3.3.1.3.5. spring-boot-starter-validation

Lehetőséget ad a Java Bean Validation és a Hibernate Validator integrált használatára, ezáltal biztosítja, hogy a kérésekben kapott objektumok különböző mezőit ellenőrizzük a kódban deklarált szabályok szerint.

3.3.1.3.6. spring-boot-starter-web

Egy igen nagy csomag, magába foglalja a webalkalmazások fejlesztéséhez szükséges összes alapfüggőséget. Lehetőséget ad REST, vagy MVC végpontok létrehozására, és azok kiszolgálására beágyazott webszerver segítségétvel.

3.3.1.3.7. spring-boot-starter-test

Tartalmazza az alapvető, teszteléshez szükséges eszközöket. Én az egységtesztekben a JUnit5 és a mockito csomagokat használom belőle.

3.3.2. Külső könyvtárak

Egyéb könyvtárak is segítik az alkalmazás különböző funkcióit. Ezeket röviden ismertetem.

3.3.2.1. Google Maps Services

Ez a függőség felelős a geocodingért, tehát a felhasználók által bevitt címek koordinátákká alakításáért. Ez azért szükséges, hogy az intézményeket el lehessen helyezni a térképen.

3.3.2.2. Lombok

Ez a projekt a biolerplate, tehát a feleslegesen bőszavú, gyakran ismétlődő kódok rövidítését szolgálja. Annotation processingen keresztül a megadott annotációk szerint legenerálja az osztályokhoz tartozó konstruktorokat, gettereket, settereket, egyenslőség-ellenőrző metódusokat, és hasonlókat. A lefordított bytecode tartalmazni fogja ezeket, de a java kód nem. Így sokkal tisztább, tömörebb kódot kapni.

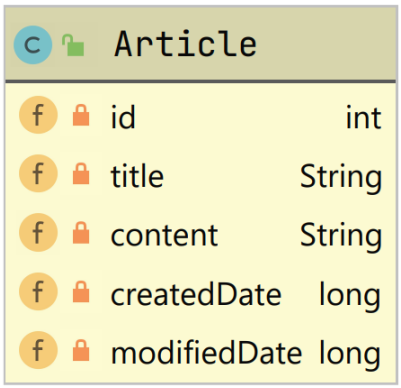
3.3.3. Perzisztencia

Az alkalmazás PostgreSQL adatbázist használ az intézmények, cikkek tárolására. Ez a modern adatbázis-rendszer széleskörű funkcionalitást biztosít nagyon szabad, open source liszenszelés mellett. Mindazonáltal az adatok formája kevéssé igényli a relációs modellt, így elképzelhető, hogy a jövőben Firebase Realtime Database vagy Firestore implementációra cserélődik, amik NoSQL adatbázisok. Míg előbbi egy JSON dokumentumban tárolja az adatokat, addig utóbbi egy dokumentum-kupacot tárol. Mindkettő jó kliensoldali offline-támogatással rendelkezik.

3.3.4. Modellek, adatelérési (DAO) réteg

A szolgáltatókhoz kapcsolódó osztályok UML diagramjai a mellékleteknél találhatók, az 1. sz. melléklet alatt. A diagramon nincs feltüntetve a default konstruktor, a getter, setter, valamint az equals-hoz kapcsolódó metódusok.

A cikkek osztálya önálló, ezért annak UML diagramját beágyazottan közlöm.



ábra 18 - A cikk osztály UML diagramja

A modellekből a javax.persistence annotációkkal generáltam adatbázis táblát. A hozzáférést a CrudRepository kiterjesztésével valósítottam meg, amely ezáltal biztosítja az alapvető, gyakori metódusokat. Nem volt szükség egyedi metódus, lekérdezés bevezetésére.

3.3.5. Kiszolgálók (Service-ek)

A modellekből Service osztály készült, ahol a CRUD metódusokat valósítom meg. Fontos ezen felül, hogy ezen a szinten implementáltam a Spring Cache gyorsítótárazását. Az ArticleService és a ProviderService mellett van GeocodingService is, ami a Google Maps API-ja segítségével valósítja meg a postai címek koordinátákká alakítását.

3.3.6. Vezérlő (Controller) osztályok

A három Service osztályra három Controller osztály épül, létrehozva a végpontokat, amikről külön fejezetben írok. Fontos a Controllerekben, hogy RestController annotációval vannak létrehozva, így a ResponseBody automatikusan rákerül a metódusokra. Ezáltal a visszaadott objektum mindig JSON-szerializációt követően, a body-ban kerül vissza a hívóhoz.

3.4. Autentikáció és autorizáció

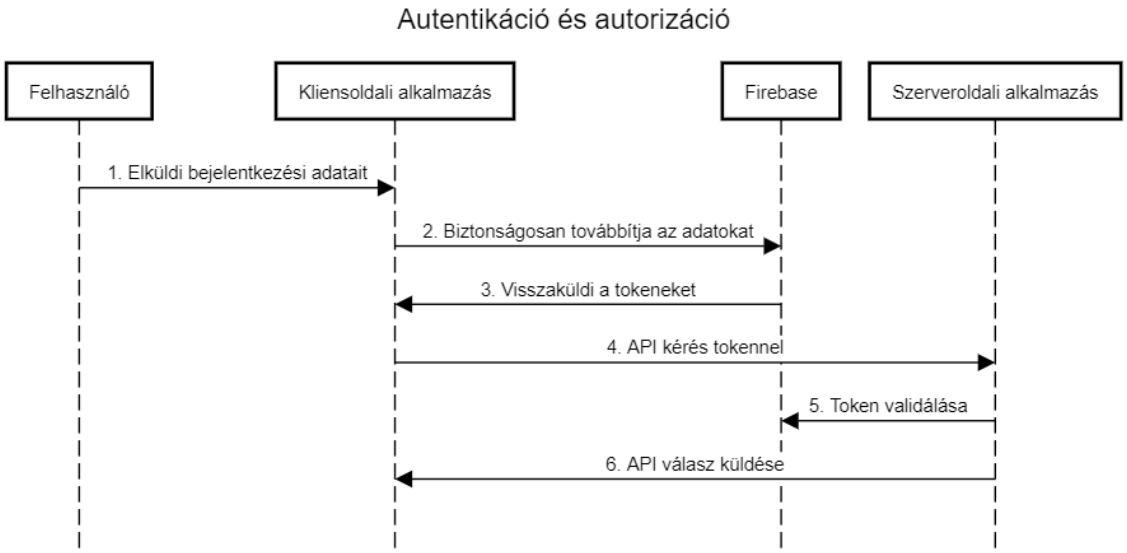
Az alkalmazásban OAuth2 biztosítja az autorizációt, OpenID Connect (OIDC) pedig az OAuth2 felett az autentikációt. Az OAuth2 főleg hozzáférés-delegálásból ismert, hiszen alkalmas arra, hogy feloldja azt a helyzetet, amikor sem a kliens, sem a szerver nem bízik a másikban: a szerver nem hiszi el, hogy a kliens az, akinek mondja magát, a kliens pedig nem bízik abban, hogy a szervernél biztonságban lenne a belépési adata. Az OAuth2 lehetőséget ad harmadik fél bevonására (pl. Google, Facebook, Amazon, Microsoft, stb.), ahol a felhasználó szokásos módon beléphet. A felhasználó kriptográfiailag aláírt tokent kap, ami igazolja a szerver felé, ki ő. A kriptográfia aszimmetrikus, a szerver ellenőrizheti a kiállító publikus kulcsával a token hitelességét. Mivel mindketten megbíznak a harmadik félben, így ezután, a hitelesített tokennel megbízhatnak egymásban is. A felhasználó nem adta meg jelszavát a szervernek a belépésnél. Az OIDC az access tokent identity tokennel egészíti ki, ami információkat tartalmaz a felhasználóról: jogosultságait, azonosítóit, egyéb adatait. Ez alkalmassá teszi az OAuth2 és az OIDC kombinációját mind autentikációra, mint autorizációra.

A szerver a harmadik fél publikus kulcsát eltárolhatja, és a megfelelő kriptográfiai eljárással ezután harmadik fél hívása nélkül is hitelesíthet azonos kiállítótól érkező tokeneket. Ez különösen azért fontos, mert a tokent minden kéréshez csatolni kell. Ez rámutat a módszer egyik fontos hátrányára: mivel a token hosszabb, mint egy munkamenet azonosító, ezért többletterhet ró a hálózati adatforgalomra. A másik hátránya, hogy nagyon nehezen vonható vissza egy már kiállított token, annak ellopása

így visszaélésre ad lehetőséget. Ennek megoldásaként a token érvényessége rövid, és bejelentkezésnél az access token és az identity token mellett egy harmadik, a refresh token is kiállításra kerül. Ennek lejárata hosszú, és ezt a kliens csak a harmadik fél felé küldi, akkor, ha a rövidlejáratú token már érvénytelenné válik. Ezzel új tokenek kerülnek kiállításra számára. A tokenek invalidálására nincs szabványos megoldás, de a legtöbb szolgáltató előállt valamilyen megoldással a problémára.

Az egyik fontosabb előnyről már írtam – a kölcsönös bizalmatlanság feloldásáról –, de emellett jelentős előny még a módszer skálázhatósága. Egy mikroszerviz-architektúra esetén hagyományos, munkamenet-alapú azonosításnál szükség van az adatok propagálására mindegyik szerviz felé. Erre többféle módszer létezik, de egyik sem igazán skálázódik jól. Ezzel szemben az oauth nem igényel állapottárolást szerveroldalon.

Az OAuth2 szabvány többféle flow-t kínál a különböző helyzetekre. Az általam választott esetében a felhasználó nem látogat meg harmadik felet bejelentkezés közben, hiszen a Firebase kliensoldali SDK-ja hívja meg a saját szerveroldali auth szolgáltatóját. Az alkalmazás a tokent kiállító fél is, a Firebase közreműködésével. A szerveroldali alkalmazás resource server üzemmódban fut – ilyenkor nem felel token kiállításáért, bejelentkezésért, csak kérésben érkező token hitelesítéséért. A folyamat így néz ki:



ábra 19 - Az autentikáció szekvencia-diagramja

3.4.1. A tokenek anatómiája

A token szabványos JWT, mely három enkódolt részből áll, pontokkal elválasztva, pl:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2Q
T4fwpMeJf36POk6yJV_adQssw5c
```

A három rész a fejléc, a tartalom és az aláírás. A fejléc (header) egy JSON-ben tartalmazza a kriptográfiai algoritmus (alg) és a token fajtáját (typ). A tartalom (payload) egy JSON objektum, ami kötelező (kiállító, kiállítás, lejárat, stb.), és szabadon hozzáadott kulcs-érték párokat tárol. Ez a token információtartalma. Az utolsó rész a kriptográfiai aláírás, ami a publikus kulccsal hitelesíthető, a szabványban leírt módon.

A tokent a kliensoldali alkalmazás minden kéréshez csatolja Authorization headerben, Bearer előtaggal.

Jelenleg nincsenek megkülönböztetett szerepkörök az alkalmazásban, de azt a tokenben hozzáadott további adatokkal lehet majd megvalósítani. Ezeket hívja claimnek a szabvány.

3.5. Végpontok dokumentációja

A végpontok részletes dokumentációja elérhető a backenden, többféle formátumban is:

- interaktív swagger felületen a https://api.pszi.info/swagger-ui.html címen
- openapi v3 szöveges formátumban a https://api.pszi.info/v3/api-docs címen

3.5.1. Megfontolások

A végpontok felépítésénél a következő szabályokat követtem:

- A végpont főnév, többes szám, ha valamilyen adatbázisban tárolt modellt kezel, pl.:
  - /providers
  - /articles
- Az ilyen végpont listázza az adott modellből található példányokat (GET ige), de alkalmas az adott modell új példányának létrehozására is (POST ige)
- Konkrét példány az azonosítója konkatenációjával kérhető le, pl.:
  - /providers/1
- Ez a végpont nem csak lekérdezésre (GET ige), de módosításra (PATCH ige) és törlésre (DELETE ige) is alkalmas
- Ha nem adatbázishoz kötődő a szolgáltatás akkor a neve a végpont, pl.:
  - /geocoding

- Ilyenkor a szolgáltatás metódusai vannak kiszervezve végpontként, pl.:
    - /geocoding/geocode

Az adott igékre vontakozó szabályok:

- Csak a POST metódus nem idempotens, az összes többi az, noha a DELETE meghívása is állapotváltozással jár először a háttérben

A válaszkódokra érvényes szabályok:

- 401-es kóddal válaszol a szerver, ha a token érvénytelen vagy hiányzik
- 500-as kóddal válaszol a szerver szerveroldali hiba esetén
- GET és PATCH metódus esetén:
    - 200-as kód sikeres kérés esetén
    - 404-es kód, ha nemlétező objektumra mutat a kérés
- POST metódus esetén:
    - 201-es kód, ha sikeres a kérés
    - 400-as kód, helytelen kérés esetén
- DELETE metódus esetén:
    - 204-es kód, bármely esetben

3.6. Üzemeltetés

Az egyes komponensek – a kliensoldali és a szerveroldali alkalmazás – éles szerverre való telepítését már részleteztem a Fejlesztői környezet fejezet "Függőségek a kliensoldali kódhoz" és "Függőségek a szerveroldali kódhoz" résznél. Az ott leírtak az irányadók itt is, ez a fejezet így nem a konkrét parancsokról szól, hanem főleg az automatizációra összpontosít, ami segíti a parancsok futtatását a kódváltozások függvényében. A most következő információk így feltételezik, hogy az olvasó megismerte az ott leírt, releváns build- és teszteszközöket és a szükséges parancsaikat.

A forráskód GitHub rendszerben tárolt, így a GitHub Actions CI rendszert használtam automatizáláshoz. Ez a lehetőség mindenkinek elérhető, ingyenes szolgáltatás a platformon, 2000 perc (33.3 óra) szerveridőig. A mostanában igen népszerű, Infrastructure-as-Code (IaC) elveket követi az eszköz, igy viselkedése teljes mértékben szabályozható a konvencióknak megfelelő, kódtárba elhelyezett szöveges leírókkal.

A .github/workflows alá elhelyezett yaml leírókat a GitHub automatikusan felismeri és lefuttatja, ha megfelelnek a szelektorai az adott kódváltozásra (branch megjelölése, pull requestben történő változtatás, és hasonlók szabhatók meg).

3.6.1. A szerveroldali kód automatizációja

```
name: Test backend
on: [push]
jobs:
  test-backend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up JDK15
        uses: actions/setup-java@v2
        with:
          java-version: '15'
          distribution: 'adopt'
      - name: Test with Gradle
        working-directory: ./api
        run: ./gradlew test
```

Ez minden kódváltozásra lefut, és egy ubuntu virtuális gépen checkoutolja a kódot, majd 15-ös JDK telepítése után lefuttatja a gradle wrapper segítségével a test parancsot az api könyvtárban. Ha hibakóddal tér vissza a parancs, a futás sikertelen lesz. A parancs kimenete is megtekinthető, így leolvasható, melyik teszt tört el.

A dockerhub platform szintén integrálható GitHubba. Ott összeköthető egy docker repository egy GitHub repository-val, és megadhatók build szabályok: melyik branchet figyelje, milyen elnevezéssel készítsen belőle image-et, hol található a Dockerfile, illetve hogy automatikusan vagy manuálisan történjen-e az image elkészítése. Mivel a szerveroldali kód változásai potenciálisan az adatbázis-struktúra változásával is járnak, így a frissítéseket az éles környezetben nem automatizáltam. Az itt elkészült docker image-dzsel konfigurált a szerveren egy docker-compose fájl. Ezt a második mellékletben közlöm.

A https kapcsolat miatt nem csak az alkalmazást és az adatbázist, de egy nginx http szervert és egy certbot konténert is tartalmaz. Utóbbi azért felel, hogy letsencrypten keresztül TLS tanúsítványt szerezzen. Ez egy másik projekt[6] alapján készült, és az ott található szkript futtatása szükséges az első indítás előtt.

[6] Philipp. *wmnnd/nginx-certbot*. Shell, 2021. https://github.com/wmnnd/nginx-certbot.

3.6.2. A kliensoldali kód automatizációja

```
name: Test and deploy frontend
'on':
  push:
    branches:
      - main
jobs:
  test-deploy-frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: npm ci && npm run build
        working-directory: ./ui
      - run: npm run test
        working-directory: ./ui
        env:
          CI: true
      - uses: FirebaseExtended/action-hosting-deploy@v0
        with:
          repoToken: '${{ secrets.GITHUB_TOKEN }}'
          firebaseServiceAccount:                              '${{
secrets.FIREBASE_SERVICE_ACCOUNT_PSZI_INFO }}'
          channelId: live
          projectId: pszi-info
          entryPoint: "./ui"
        env:
          FIREBASE_CLI_PREVIEWS: hostingchannels
```

Ez a forgatókönyv a forráskód letöltése után felépíti a projektet, majd teszteli. Ha sikeres, telepíti a Firebase segítségével. Ehhez a GitHub rendszerében tárolt kulcsot és azonosítót használja. Az alapvető összeállításban segíthet a Firebase parancssori eszköze, hiszen található benne támogatás, kódgenerálás a GitHub Actions CI eszköz felé. Mivel a Firebase Hosting szolgáltatását használom, nem szükséges egyéb teendő elvégzése. A TLS tanúsítványért a Firebase felel.

3.7. Tesztelés

Mivel a felhasználói élmény kiemelten fontos szerepet kapott az alkalmazásban, ezért a manuális tesztelés folyamatosan kísérte a fejlesztést. Gyakran a tesztelés mutatott rá arra, hol szorul finomításra az alkalmazás, hogy igazán intuitív lehessen a használata.

Az automatizált tesztek főleg demonstrációs céllal kerültek a kódba és a CI/CD pipeline-ba. Az alkalmazás fejlesztésének fenntarthatóságához hosszú távon elengedhetetlenül fontos lesz a lefedettség növelése, de ez a feladat még a további fejlesztési tervek része.

3.7.1. Eszközök

A szerveroldali kód manuális tesztelése Postman alkalmazással történt, egy összeállított kérés-gyűjteménnyel. A Postman egy részletesen paraméterezhető REST kliens, ami lehetőséget biztosít projektek kezelésére is, és elmentett kérések strukturálására azokban. A kérésgyűjtemény esetei a teszteseteknél részletezettek. A szerveroldali automatizált tesztek JUnit5 és mockito keretrendszerekkel készültek, automatikus futtatásuk minden kódváltozásnál megtörténik a GitHub Actions segítségével.

A kliensoldali kód manuális tesztelése tesztforgatókönyvek segítségével történt. Ezeket a teszteseteknél részletezem. Az automatizált tesztesetek a testing-library jest moduljával íródtak. Futtatásukért a GitHub Actions felel.

3.7.2. Tesztesetek és teszteredmények

3.7.2.1. Szerveroldali tesztesetek

táblázat 4 - A szerveroldali tesztesetek

Eset	Elvárt viselkedés	Eredmény
Védett végpont hívása token nélkül	401-es hibakóddal válaszol a szerver	Sikeres
Védett végpont hívása lejárt tokennel	401-as hibakóddal válaszol a szerver	Sikeres
Ellátóhely hozzáadása	A megfelelő formátumú input bekerül az adatbázisba, 201-es válasszal tér vissza az API	Sikeres
Hiányos ellátóhely hozzáadása	A hiányos input nem kerül az adatbázisba, 400-as hibakóddal válaszol a szerver	Sikeres

Ellátóhelyek listázása üres adatbázissal	Üres adatbázis esetén üres tömbbel tér vissza, 200-as válaszkóddal	Sikeres
Ellátóhelyek listázása adatokkal	Visszaadja az adatbázisban található ellátóhelyeket, 200-as kóddal	Sikeres
Hiányzó ellátóhely lekérdezése	404-es hibakóddal válaszol a szerver	Sikeres
Létező ellátóhely lekérdezése	200-as kóddal és a kért ellátóhely adataival válaszol a szerver	Sikeres
Ellátóhely törlése	204-es kóddal válaszol a szerver	Sikeres
Ellátóhely-adatok verziójának lekérdezése	200-as kóddal és a verziószámmal válaszol a szerver	Sikeres
Cím geokódolása	200-as kóddal és a koordinátákkal válaszol a szerver	Sikeres
Cikk létrehozása	201-es kóddal és id-val ellátott cikkel tér vissza a szerver	Sikeres
Cikkek listázása üres adatbázis esetén	200-as kóddal és üres tömbbel tér vissza a szerver	Sikeres
Cikkek listázása adatokkal	200-as kóddal listázza a rendszer a cikkeket	Sikeres
Cikk lekérdezése azonosító alapján	200-as kóddal és a cikkel tér vissza a szerver	Sikeres
Hiányzó cikk lekérdezése	404-es kóddal tér vissza a szerver	Sikeres
Apidoc generálása	200-as kóddal, openapi v3 szabványú válasszal tér vissza a szerver	Sikeres

3.7.2.2. Kliensoldali tesztesetek

táblázat 5 - A kliensoldali tesztesetek

Eset	Elvárt viselkedés	Eredmény
Ellátóhelyek lekérdezése	Az ellátóhelyek megjelennek a listában és a térképen	Sikeres
Ellátóhelyek részleteinek lekérdezése a térképen	Megjelennek a részletek	Sikeres
Részletek lekérdezése a listában	Megjelennek a részletek	Sikeres
Szűrők használata a találatok szűkítésére	A szűrők szűkítik a találati listát	Sikeres
Ellátóhely hozzáadása	Az ellátóhely átirányítás után megjelenik a listában	Sikeres
Ellátóhely törlése	Az ellátóhely eltűnik a listából	Sikeres
Cikkek lekérdezése	Megjelennek a cikkek	Sikeres
Cikk megnyitása	A cikk megnyílik	Sikeres

4. Összefoglalás

Szakdolgozatomban egy pszichoterápiás ellátótérképet fejlesztettem, ami széleskörűen támogat különböző eszközöket, akkor is, ha azok nem mindig tudnak hálózathoz csatlakozni. Fontos cél volt, hogy intuitív, felhasználóbarát oldalt kapjon a látogató, amiben könnyen megtalálhatja, amit keres. Kezdetben úgy gondoltam, a backend jelentősebb szerepet fog kapni, a keresések, szűrések, összetett geolokáció-alapú kérések számítását szerveroldalon fogom végezni. A projekt előrehaladtával egyre világosabbá vált, hogy a modern internetes szokások, az alkalmazások használatát illető trendek olyan elvárásokat állítanak, amik miatt több figyelmet kell szentelnem a kliensoldali megoldásokra: számolnom kell azzal, hogy nagyon sokan, és egyre többen használnak alkalmazásokat nem asztali böngészőkből. A hordozható eszközök nem hagyatkozhatnak arra, hogy bármikor lekérdezhetik az adatokat a szerverről, hiszen például vonaton utazva, vagy egy rossz lefedettségű helyen csődöt mondanak. Ezzel párhuzamosan robbanásszerűen nőtt a teljesítményük, már igazán nem elképzelhetetlen a szerveroldali számításokat a kliensoldalon elvégezni. Mivel célom az alkalmazás fenntartása, fejlesztése, támogatása a jövőben is, ezért a skálázhatóságot is segíti egy ilyen architektúra. A projekt közben több dolgon változtattam ezek a megfontolások szerint. Számomra igen nehéz nem befejezni, nem tökéletesre – vagy legalábbis számomra teljesen elfogadhatóra – csiszolni egy beadott munkát, de most be kellett látnom, hogy az ambícióim túlmutattak a lehetőségeimen. Mindazonáltal a munkám az „egy program sosem készül el, csak eljön a határidő” szellemiségében íródott, a lehetőségekhez mért teljesség szerint. Az olvasó egy aktuális állapotot, egy pillanatképet lát az alkalmazásból. A dokumentáció gyakorta kitér a következő lépésekre, hiszen azok már jól látszanak ebből a távolságból. A következő fejezetben összefoglalóan írok ezekről.

5. További fejlesztési lehetőségek

Ugyan már több éve szoftverfejlesztőként dolgozom, ez az első ilyen típusú alkalmazásom. Az offline-first filozófia, megközelítés sokat tanított, hogyan gondolkodjak az alkalmazás egységéről, a szerver-kliens kapcsolatról. Ezek a tanulságok még csak részben tudtak bekerülni az alkalmazás jelenlegi változatába. Mivel úgy vélem, a Firebase szolgáltatásai jól reagálnak az alkalmazás igényeire, szeretném a jövőben a teljes backendet a Firebase különböző lehetőségeivel helyettesíteni. Ezáltal nem csak a technológiai stack lesz fókuszáltabb, tömörebb, de az alkalmazás üzemeltetése, skálázhatósága is egyszerűbbé válik majd. Ezek a jól kipróbált eszközök lehetőségeket adnak részletes jogosultságkezelésre, perzisztens adattárolásra, fájlok kiszolgálására, mindezt offline-first megközelítésben.

A tervezett funkciók közül a felhasználói szerepek, jogosultságkezelés csak minimális szinten került be az alkalmazásba. Ez a közösségi szerkesztéshez kötődő elképzeléseim megvalósulásához még elégtelen. (Erről részletesen a Felhasználói jogosultságok alfejezetben írtam.) A jövőben módszeresen bevezetésre kerülnek majd a hiányzó szerepek, és a tapasztalatgyűjtés után majd finomodnak, míg beáll egy használható, az eredeti elképzelésekhez közeli egyensúly a felhasználóból szerkesztővé válás feltételeiben.

A kliensoldali kód Reactre épül, de szeretném emellett Typescriptre átírni. Nagyon jó eszköznek találom hosszútávon, a kód fenntarthatóságának tekintetében. A tesztlefedettség növelésével együtt a Typescript bevezetése lenne, ami a fenntarthatóságot segíti.

Jelenleg a megjelenített intézményi adatok igen szűkösek. Ennek a bővítése mindenképpen szükséges, hogy igazán hasznos lehessen az alkalmazás. Már most több intézménnyel felvettem a kapcsolatot az adatbázis bővítése érdekében. Elsőként az állami pszichiátriai és pszichoterápiás intézményhálózat adatait gyűjtöttem össze a Semmelweis egyetem megkeresésével. Sajnos a koronavírus-helyzet jelentősen befolyásolja az intézmények nyitvatartási adatait. Több osztályt átalakítottak a járványhelyzet támogatására. A stabilizálódás mindenképpen szükséges ahhoz, hogy megbízhatóan össze lehessen gyűjteni az aktuális értékeket. A Nemzeti Szociálpolitikai Intézet volt olyan kedves, hogy elküldte nekem az országosan elérhető család- és gyermekvédelmi központok és szolgálatok címlistáját. Ennek a beolvasása még igényel munkát.

Szándékomban áll még a nevelési tanácsadók megkeresése (sajnos eddig sikertelen), a főleg alapítványi- és egyházi fenntartású addiktológiai ellátás hozzáadása – ami folyamatban van, de egy készebb platformot akarok mutatni az intézmények megkeresésénél. Ha demonstrálható állapotba kerül az alkalmazás, nyitni fogok társszakmák, szakemberek felé, hogy jól fel tudjam térképezni a potenciális intézményeket, akik bekerülhetnek az adatbázisba. Jelenleg a magánrendelések nem részei az adatbázisnak, és ezen nem is szándékozom változtatni, mivel egyrészt ilyen típusú keresők már léteznek, másrészt fontosnak találom a szolgáltatók hozzáférhetőségét, akár a legelesettebb embertársaink számára is. A rendszerben új intézménytípusok létrehozása is szükséges, és a keresőrendszer kapcsolódó módosításai is elkerülhetetlenek újfajta intézmények hozzáadásakor.

Az intézménylista bővítésén túl a cikkek, cikkszerzők toborzása is fontos feladat. Több pszichológiai portál is működik, ahol szakemberek, szakmai szemmel is kiváló, a témában releváns cikkeket produkálnak. A jövőben, ha a portál ehhez szükséges funkcionalitásai már teljesen bevezetésre kerülnek, és az intézménylista is kellően meggyőző, több ilyen online újsággal is felvenném a kapcsolatot a cikkek bővítését illetően. Első sorban konkrét, a Pszi-Infó szempontjából releváns cikkek szerzőmegjelöléssel, attribúcióval való átvétele kapcsán. Ezen túl megnyitok a portálon egy szerzőtoborzó felületet, ahol vendégcikkeket lehet küldeni.

A legtöbb fejlesztést igénylő tervem egy olyan alkalmazás, vagy chatbot létrehozása, ami interaktív módon, kérdések megválaszolásával irányítja el a segítségkérőt a megfelelő típusú intézményrendszerbe, vagy ajánl konkrét ellátóhelyet. Ennek a kifejlesztése a társszakmákkal együttműködve lehetséges, és még beláthatatlan, mikorra készül majd el.

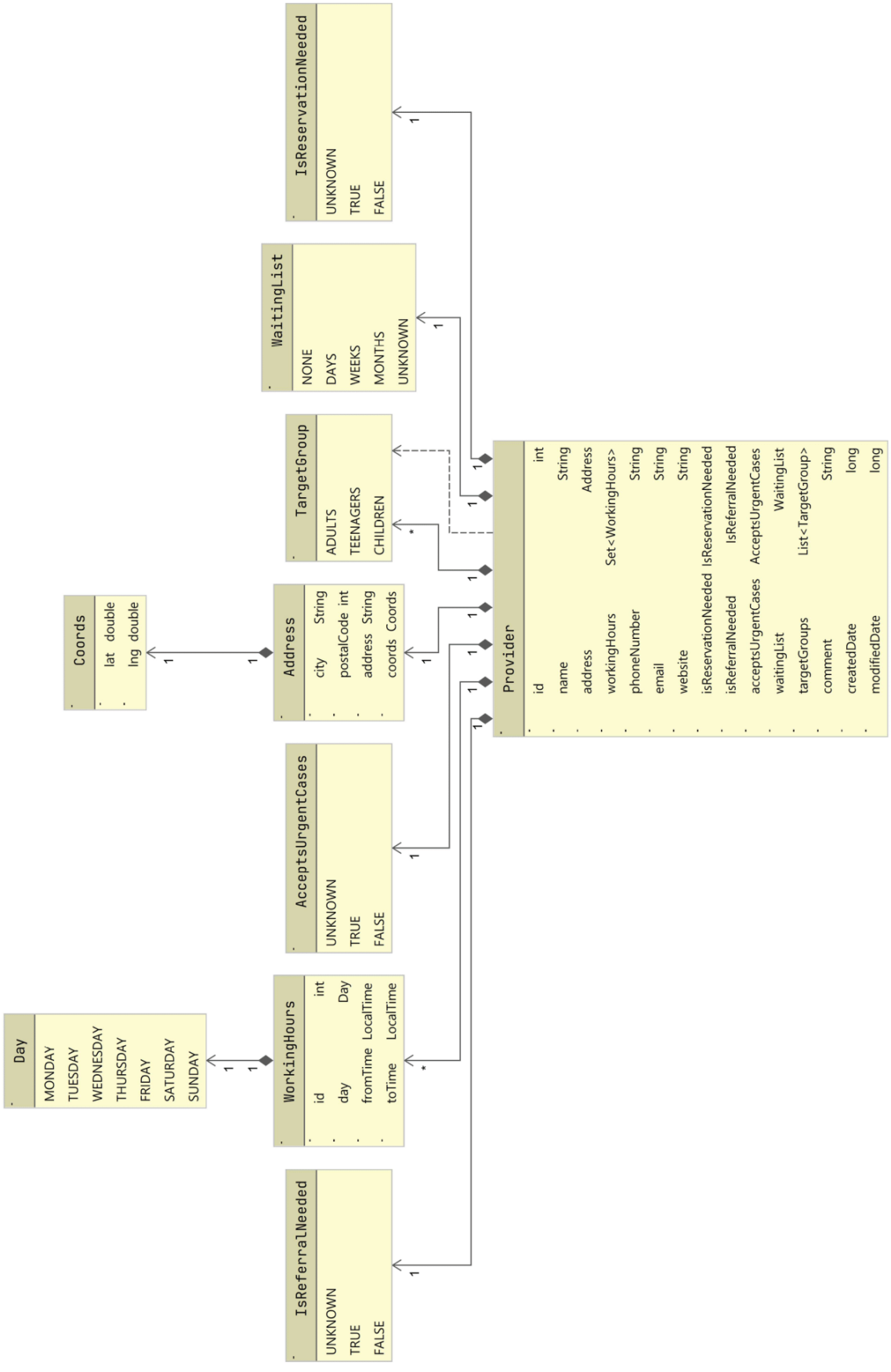
Úgy vélem, ezekkel a fejlesztésekkel teljessé válik az oldal, és maradéktalanul beteljesítheti tervezett, társadalmilag is hasznos feladatát – összeköthet rászoruló, mentálhigiénés szempontból nehéz helyzetbe sodródott embereket a számukra megfelelő intézményrendszerrel.

Csuvik Gábor

Budapest, 2021. 05. 24.

6. Mellékletek

6.1. 1. sz. melléklet: A szolgáltatókhoz kapcsolódó osztályok UML diagramja



6.2. 2. sz. melléklet: A szerveroldali alkalmazás docker-compose fájlja

```
version: '3.8'

services:
  db:
    image: postgres:13-alpine
    restart: unless-stopped
    environment:
      POSTGRES_PASSWORD: "${POSTGRES_PASSWORD}"
      POSTGRES_DB: "${POSTGRES_DATABASE}"
    volumes:
      - dbdata:/var/lib/postgresql/data
  api:
    image: csuvikg/pszi-info-api:latest
    restart: unless-stopped
    environment:
      POSTGRES_HOST: "${POSTGRES_HOST}"
      POSTGRES_PORT: "${POSTGRES_PORT}"
      POSTGRES_DATABASE: "${POSTGRES_DATABASE}"
      POSTGRES_USERNAME: "${POSTGRES_USERNAME}"
      POSTGRES_PASSWORD: "${POSTGRES_PASSWORD}"
      GOOGLE_MAPS_API_KEY: "${GOOGLE_MAPS_API_KEY}"
    depends_on:
      - db
  nginx:
    image: nginx:1.15-alpine
    restart: unless-stopped
    volumes:
      - ./data/nginx:/etc/nginx/conf.d
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    ports:
      - '80:80'
      - '443:443'
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s
reload; done & nginx -g \"daemon off;\"'"
```

```
  certbot:
    image: certbot/certbot
    restart: unless-stopped
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew;
sleep 12h & wait $${!}; done;'"

volumes:
  dbdata:
```