XX. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Kolozsvár, 2016. május 18-21.

Terapeuták munkáját támogató szoftverrendszer: a dWelling projekt

Szerzők:

Ambrus Adrián-Zoltán
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, informatika szak, III. év

Balázsi Róbert-Sándor
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, informatika szak, III. év

Témavezetők:

Dr. Simon Károly
Egyetemi adjunktus, Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar
Projektmenedzser, Codespring

Kandó Norbert
Projektmenedzser, Codespring

Sipos Áron
Szoftverfejlesztő, Codespring

Kivonat

A dolgozat egy szoftverprojektet mutat be, amelynek célja egy főként alternatív gyógyászatban használható foglalási és lelet-nyilvántartási szoftverrendszer kifejlesztése. A rendszer egy központi szerverből és egy webes felületből áll. A rendszer adminisztrátorai a webes felületen keresztül terápiákat vihetnek fel, terapeutákat regisztrálhatnak a rendszerbe és menedzselhetik a páciensek felhasználói fiókjait. A vendég felhasználók megtekinthetik a rendszer által támogatott terápiákat és regisztrálhatnak pácienskénť. A páciensek a beépített keresőmotor segítségével számukra megfelelő kezeléseket kereshetnek, és jelentkezhetnek ezekre a kezeléseket végző terapeutáknál. A terapeuták terápiákat rendelhetnek magukhoz, meghatározhatják munkaprogramjukat és email-en keresztül értesülnek a foglalásokról. Felvihetik a rendszerbe a páciensek kórlapjait, egészségügyi állapotukra vonatkozó információkat, illetve kezelésekre vonatkozó adatokat rögzíthetnek.

Tartalomjegyzék

Bevezető **0**

1. Alkalmazott módszerek és eszközök **1**

2. A dWelling projekt **2**

2.1. Funkcionalitások 2

2.1.1. Vendégek számára elérhető funkcionalitások 2

2.1.2. Páciensek számára elérhető funkcionalitások 2

2.1.3. Adminisztrátor számára elérhető funkcionalitások 2

2.1.4. Terapeuták számára elérhető funkcionalitások 3

2.2. Architektúra 3

3. A szerver megvalósítása **4**

3.1. Felhasznált technológiák 4

3.1.1. Spring Boot 4

3.1.2. Spring Web MVC 5

3.1.3. Spring Data JPA 5

3.1.4. Spring Security 5

3.1.5. Spring Data Elasticsearch 6

3.1.6. Hibernate 6

3.1.7. Mapstruct 7

3.2. Adatmodell 7

3.3. Kommunikáció 8

3.4. Adathozzáférési réteg 10

3.5. Szolgáltatás réteg 11

3.6. Biztonság 12

3.6.1. Spring Security 12

3.6.2. Autentikáció 12

3.6.3. Autorizáció 13

4. Web kliens **13**

4.1. Felhasznált technológiák 13

4.1.1. AngularJS 13

4.1.2. TypeScript 14
4.2. Adatmodell 14
4.3. Kommunikáció 15
4.4. Biztonság 16

5. Az alkalmazás működése 17
5.1. A terapeuták számára elérhető funkciók 17
5.2. Páciensek számára elérhető funkciók 21
5.3. Adminisztrátorok számára elérhető funkciók 25

Következtetések és továbbfejlesztési lehetőségek 26

Hivatkozások 27

Bevezető

A dWelling projekt célja egy olyan szoftverrendszer megvalósítása, amely segítséget nyújt egészségügyi problémákkal küszködő emberek és az őket kezelő terapeuták számára. Lehetőséget ad alternatív gyógymódok, különböző komplementer terápiák megismerésére és kipróbálására.

A szoftver különböző terápiákat gyűjt össze, a felhasználók ebben a gyűjteményben kereshetnek, információkat kaphatnak a kezelési eljárásokról. Az eljárásokat végző terapeutákat is nyilvántartja a rendszer, számos funkcionalitást biztosít számukra a könnyebb munkavégzés érdekében.

A webes felületen keresztül a beteg megkeresheti a számára legmegfelelőbb terápiát, ebben segítséget nyújt egy kereső motor, amely képes kulcsszavak alapján kiszűrni azokat a terápiákat, amelyek megoldást jelenthetnek a látogató panaszaira. A terápiákról részletes leírást ad a rendszer és képek is segítik az érdeklődőt abban, hogy megtalálja a megfelelő kezelést. A rendszer által nyilvántartott terapeuták között is kereshet a páciens, megtekintheti a terapeuta nyilvános adataid, munkabeosztását, szabad időpontjait. Ha sikerült egy megfelelő időpontot találni, akkor lefoglalhatja ezt.

A terapeuta a webes felület segítségével jegyzi a betegeit és a hozzájuk tartozó információkat (kórlapok, életmóddal és egészségi állapottal kapcsolatos információk stb.). A rendszer lehetőséget nyújt számára, hogy minden páciens részére külön-külön vezesse a kezelések kimeneteleit, leírhatja, hogy miként reagált a beteg egy adott kezelésre. Egy grafikus felület teszi lehetővé a fájdalompontok felvitelét és változásaik követését. A terapeuta így könnyen kaphat átfogó képet az adott beteg állapotáról. A terapeuta munkaprogramját is megszabhatja, beállíthatja, hogy mikor dolgozik, milyen időpontoktól kezdődnek az általa végrehajtott kezelések és azok mennyi időt tartanak. Ez a megoldás megkönnyíti az időpontfoglalásokat.

A dWelling szoftverrendszer két részből tevődik össze, egy szerverből és egy webes kliensalkalmazásból. A webes kliensalkalmazás segítségével a felhasználók elérhetik a rendszer szolgáltatásait. A fejlesztés során fontos szerepet kapott a reszponzivitás, hogy az alkalmazásnak optimális megjelenést, könnyű felhasználhatóságot biztosítsunk bármely képernyőméreten. A szerver fejlesztése során a többrétegű architektúra tervezési mintát követtük, a könnyebb karbantartás, továbbfejlesztés érdekében. A projekt esetében három réteget különítettünk el: megjelenítésért felelős réteg, üzleti logikáért felelős réteg, adathozzáférési réteg.

A dolgozat a fentebb leírt dWelling projektet mutatja be. Az első rész a megvalósítás során használt fejlesztési folyamatot, a felhasznált eszközöket ismerteti. A második rész a projekt részletes leírását tartalmazza továbbá részletesen tárgyalja a követelményeket, a használati eseteket és a rendszer architektúráját. A harmadik részben kerül sor a szerver részletes bemutatására, itt szó esik a szerver oldalon felhasznált technológiákról, eszközökről. A negyedik rész a web kliens megvalósítását tárgyalja, a felhasznált technológiákkal együtt. A dolgozat a következtetések és továbbfejlesztési lehetőségek leírásával zárul.

A projekt a Codespring Mentorprogram keretein belül született, a nyári gyakorlat kezdetén a fejlesztőcsapat elsajátította a projekt fejlesztéséhez szükséges technológiák és módszerek alapjait. A fejlesztést a cég kinevezett szakemberei irányították. Az egyetemen a „Csoportos Projekt” tantárgy keretein belül a fejlesztés tovább folytatódott, a csapat ideiglenesen kibővült két taggal, Hankó Lehellel és Bőr Tamással. A projekt létrejöttéért köszönet illeti mentorainkat: Kandó Norbertet, dr. Simon Károlyt és Sipos Áront.

1. Alkalmazott módszerek és eszközök

A dWelling projekt fejlesztése során a fejlesztők több olyan népszerű, széles körben alkalmazott fejlesztői eszközt használtak, amelyek a hatékonyságot javították.

A csapat a fejlesztés során Scrum [1] módszert alkalmazott, ez egy agilis módszer, amely inkrementáló és iteratív fejlesztést támogat. A fejlesztés a Scrumnak megfelelően sprintekben valósult meg (a csapat a két hetes fejlesztési ciklusokat találta a leghatékonyabbnak). A sprintek egy tervezési fázissal kezdődtek, ekkor a csapat kiválasztotta, hogy az elvégzendő feladatok közül melyek kerüljenek megvalósításra, és megbecsülte az implementáláshoz szükséges időt. A napi szintű, rövid találkozók segítették a csapatmunkát. A kéthetes fejlesztési ciklus egy bemutatóval zárult (Demo).

A forráskód változásainak nyomon követesére a fejlesztői csapat a GIT osztott verziókövetőt választotta. A csapat kliensalkalmazásként SourceTree-t, illetve az IntelliJ IDEA beépített verziókövetőjét használta. A távoli tároló karbantartását egy GitLab szerver biztosította. Minden új funkcionalitás fejlesztése külön ágon történt, ha a funkcionalitás elkészült és a mentorok jóváhagyták, akkor bekerült a default ágba.

A szoftverrel szemben támasztott megrendelői elvárásokat, követelményeket felhasználói történetek (User Story-k) fogalmazták meg. Ezeknek a nyilvántartásában és menedzselésében a GitLab rendszer beépített issue tracking modulja segített.

A build folyamat automatizálására és a függőségek menedzselésére a csapat Gradle-t használt. Ez egy Groovy szkript alapú rendszer. A webes projekt build-elését a Gulp végezte, a Gradle indította el. Az npm csomagkezelő biztosította a webes fejlesztésben felhasznált eszközök menedzsmentjét.

A folyamatos integráció (Continuous Integration) egy szoftver fejlesztési módszer, azt a célt szolgálja, hogy a rendszer mindig helyes, felépíthető és futtatható állapotban legyen. A dWelling projekt esetében a GitLab CI szolgáltatása végezte a folyamatos integrációt, nagy előnyt jelentett, hogy nem kellett összekötni más rendszereket a verziókövetővel. A különböző folyamatok leírása egy YAML állományban történt, a folyamatok egy-egy docker image-en belül hajtódtak végre. Ha új kód került a GitLab által karbantartott tárolóba akkor a rendszer automatikusan buildelte a projektet, lefuttatta az automatizált teszteket majd utolsó lépésként kitelepítette az alkalmazást a teszt szerverre.

2. A dWelling projekt

A dWelling rendszer olyan valós problémákra szeretne megoldást nyújtani, amelyekkel terapeuták szembesülnek mindennapi munkavégzésük során. Ilyen probléma lehet például az online foglalási lehetőség hiánya, vagy az adatok megfelelő tárolására alkalmas rendszer hiánya. A dWelling egy könnyen áttekinthető, keresési lehetőséget támogató felületet biztosít ezeknek a problémáknak a kiküszöbölésére. A különböző terápiákat egy helyre gyűjti és a felhasználókat a megfelelő információkkal látja el a terápiákról.

2.1.Funkcionalitások

A rendszer négy felhasználói szerepkört különít el (vendég, páciens, terapeuta és adminisztrátor), a funkcionalitásokat ennek megfelelően kategorizálhatjuk.

2.1.1. Vendégek számára elérhető funkcionalitások

A vendégek számára elérhetőek az alkalmazásban szereplő terápiák, azok leírásai, a terápiákat végző terapeuták neve és értékelése. A leírások mellett egy galéria is megjelenik, amelyen a kezelésekkel kapcsolatos képek vannak feltöltve. A vendég felhasználók számára nem elérhetőek a terapeuták elérhetőségei és nem jelentkezhetnek terápiákra, viszont lehetőségük van regisztrációra, amely által elérhetővé válnak számukra az említett funkciók.

2.1.2. Páciensek számára elérhető funkcionalitások

A páciensek szerkeszthetik profiljukat, ahol megadhatják elérhetőségeiket, beállíthatnak profilképet. Böngészhetik a terápiákat, teljes szöveg alapú kereséssel kereshetnek a terápiák között. Időpontot foglalhatnak terapeutákhoz. Email értesítéseket kapnak, ha a terapeuta elfogadta, vagy elutasította a foglalásukat, és le is mondhatják a foglalásaikat.

2.1.3. Adminisztrátor számára elérhető funkcionalitások

Az adminisztrátor regisztrálhat terapeutákat a rendszerbe. Menedzselheti a felhasználókat, korlátozhatja a hozzáférésüket a rendszerhez. Az adminisztrátor feladata felvinni a különböző típusú terápiákat is. A terápiákkal kapcsolatos információk szerkesztése egy Rich Text Editor segítségével történik, és képeket is fel lehet tölteni a terápiákhoz, ezek a páciensek számára egy galériában jelennek meg.

2.1.4. Terapeuták számára elérhető funkcionalitások

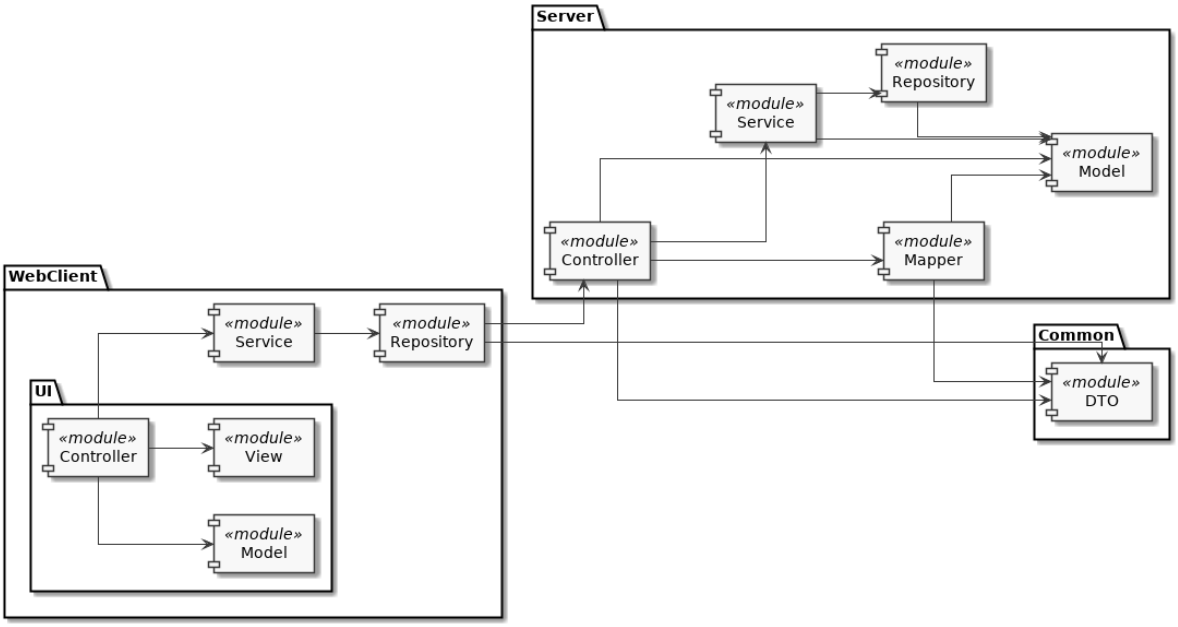
A terapeuták szerkeszthetik a profil oldalukat, ahol különböző elérhetőségeket adhatnak meg, profil képet állíthatnak be és egy rövid leírást is megadhatnak magukról. Egy naptár segítségével meghatározhatják a munkaprogramjukat, akár több hétre előre is. Az órarend szerkesztő flexibilis, ezért különböző hetekre különböző munkaprogramot is meg lehet határozni. Lehetőséget biztosítunk szünetek megadására is, amelyek lehetnek 1 órás, vagy akár egy hetes szünetek is. A szünetek alatt a foglalási lehetőség nem elérhető a páciensek számára. A terapeuták email értesítést kapnak a foglalásaik helyzetéről és menedzselhetik a foglalásaikat az oldal segítségével. Egy foglalás létrejötte után a terapeuták számára elérhetővé válnak a páciensek publikus adatai, az orvosi leletek, amelyeket más terapeuták vittek fel a rendszerbe vagy a páciensre vonatkozó általános egészségügyi információk. Lehetőség van ezeknek az adatoknak a módosítására, ugyanakkor kezelésekre vonatkozó információkat is felvihetnek a terapeuták a rendszerbe. Egy „virtuális emberi testen” a terapeuták felvihetnek fájdalompontokat, a fájdalom kiterjedésének és mértékének megfelelően, ezekhez a fájdalompontokhoz kommentárt is fűzhetnek. A rendszerben szereplő terápiák közül kiválaszthatják azokat, amelyek esetében kompetenseknek számítanak.

2.2.Architektúra

A projekt két fő komponense a Java alapú, Spring technológiákra épülő szerver és az AngularJS web kliens. A két komponens REST alapú kommunikáció által éri el egymást, az adatokat a DTO tervezési minta szerint küldik.

A szerver komponens Controller rétege publikálja a REST erőforrásokat. Ezen a komponensen keresztül kommunikál a kliens a szerverrel. A Controller réteg a DTO objektumokat szerver oldali entitásokká alakítja, a Mapper komponensek segítségével. A Controller réteg hívja meg a Service réteg szolgáltatásait. A Repository réteg felelős az adathozzáférésért és az adatok perzisztenssé tételéért. Mindegyik szerver oldali komponens használja a Model csomagot, amely az entitásokat tartalmazza, melyek POJO-ként (Plain Old Java Object) vannak implementálva.

A kliens oldalon a Repository réteg tartja a kapcsolatot a szerver oldali erőforrás réteggel. A kliens oldali Service réteg tartalmazza a kliens oldali üzleti logikát. A prezentációért felelős Controller réteg vezérli a View-t és fogyasztja a Service réteg szolgáltatásait. A szerver oldali DTO-k tulajdonképpen a kliens oldali modelleknek felelnek meg.



1. ábra A rendszer architektúrája

3. A szerver megvalósítása

A dWelling projekt szerver oldalon Java-alapú technológiákra épül. A Spring keretrendszert használja, amely egy elterjedt alternatívája a Java EE technológiának. MySQL adatbázist használ az adatok tárolására, ElasticSearch keresőmotort az adatok indexelésére és teljes szöveg alapú keresésre.

3.1.Felhasznált technológiák

3.1.1. Spring Boot

A Spring Boot a Spring keretrendszernek egy ’convention-over-configuration’ elvet támogató megoldása, amely a kezdeti fejlesztést felgyorsítja az előbb említett fejlesztési mintának megfelelően. Különálló Spring alkalmazások fejlesztését teszi lehetővé, biztosítva egy beépített webszervert, így szükségtelenné teszi a webszerverre vagy alkalmazásszerverre történő külön kitelepítést. Elkülöníti a konfigurációt az alkalmazástól, lehetővé téve a különböző környezetekben való futtatást. Az XML alapú konfiguráció mellett Java alapú konfiguráció is használható.

3.1.2. Spring Web MVC

A Spring Web MVC [2] (model-view-controller) keretrendszer lehetővé teszi, hogy rugalmas, MVC mintára épülő webes alkalmazásokat fejlesszünk. Az egész keretrendszer a DispacherServlet köré épül. Ez a komponens fogadja a beérkező HTTP kéréseket és a megfelelő kezelő felé továbbítja ezeket. A keretrendszer a @Controller és @RestController, valamint @RequestMapping annotációk bevezetésével lehetőséged ad a fejlesztőknek, hogy általuk implementált kezelőket definiáljanak. A @PathVariable, a @RequestBody és a fentebb megemlített annotációk által támogatja RESTful alkalmazások fejlesztését.

3.1.3. Spring Data JPA

A Spring Data JPA a JPA (Java Persistence API) szabványt implementáló ORM (Object-Relational Mapping) keretrendszerek fölött biztosít egy absztrakciós réteget, lényegesen leegyszerűsíti a CRUD (Create-Read-Update-Delete) műveletek implementációját. Gyakorlatilag csak a Repository interfészeket kell létrehozni, az implementációt a keretrendszer generálja. A lekérdezések is megadhatóak az interfészek metódusainak szintjén alkalmazott elnevezési konvenciók alapján, vagy a bonyolultabb lekérdezések annotációk segítségével, JPQL (Java Persistence Query Language) nyelvet alkalmazva, illetve natív SQL lekérdezések használatára is lehetőséget biztosít a keretrendszer.

3.1.4. Spring Security

A Spring Security egy erős és rendkívül testre szabható keretrendszer, amellyel Spring alkalmazások autentikációját és autorizációját lehet megvalósítani. Alapból védelmet nyújt a legelterjedtebb biztonsági problémákkal szemben, mint a session fixation, click jacking, CSRF (Cross Site Request Forgery). A könnyű kiegészíthetőségének köszönhetően egyszerű integrálni egy JWT (JSON Web Token) alapú autentikációt. Autorizációra a Spring Security annotációkat biztosít, amelyekkel szerepkör alapján nyújthatunk hozzáférést a rendszer különböző szolgáltatásaihoz.

3.1.5. Spring Data Elasticsearch

A Spring Data Elasticsearch [3] egy része a Spring Data keretrendszernek, amelynek célja megkönnyíteni az adathozzáférési réteg implementálását. Egy egységes, a Spring elvein alapuló, egyszerű módot biztosít erre, amely megtartja a különböző adatbázis-specifikus funkciókat, képességeket.

A Spring Data Elasticsearch keretrendszer biztosítja egy Elasticsearch motor integrálását a projektbe, a keretrendszer egy POJO-centrikus modell alapján lép interakcióba ezzel a motorral. Az ElasticsearchRepository interfész (a Spring Data legfontosabb interfészének, a Repository interfésznek az ötödik rendbeli leszármazottja) felhasználásával a programozóknak elegendő csak egy metódust deklarálniuk, ebből a keretrendszer képes felépíteni a lekérdezésnek megfelelő Elasticsearch oldali parancsot. A metódusok neveit a Spring által meghatározott szabvány szerint, a megadott kulcsszavakból kell felépíteni. Ezek segítségével lehetőség van bonyolult lekérdezések végrehajtására is, ha ez nem biztosít elég mozgásteret a programozók számára, akkor az adott metódus felannotálható @Query annotációval, itt megadható egy Elasticsearch lekérdező parancs.

Az Elasticsearch [4] egy könnyen skálázható, nyílt forráskódú NRT (Near Real Time) platform. Lehetőséget ad nagy mennyiségű adat tárolására, teljes szöveg alapú keresés megvalósítására és az adatok elemzésére közel valós időben.

3.1.6. Hibernate

A Hibernate egy Red Hat által fejlesztett ORM keretrendszer, amely implementálja a JPA 2.1-es szabványt. Absztrakciós réteget biztosít a JDBC (Java Database Connectivity) API fölött. Célja az objektumorientált modellen belüli osztályok és az adatbázis táblák közötti megfeleltetés megvalósítása. A keretrendszer lehetőséget biztosít az adatok Java kódból történő kezelésére. A lekérdezések megírására használhatjuk a JPQL lekérdező nyelvet, vagy a Criteria Query API-t, melyek részei a JPA standardnak. A lekérdezések eredményeit a keretrendszer a Reflection API segítségével automatikusan Java objektumokká alakítja.

3.1.7. Mapstruct

A szerver és kliens közti kommunikáció megvalósítása a DTO tervezési minta alpján történt. A DTO-k csak a kliens számára szükséges információkat tartalmazzák, adott esetben több entitás adatait vonják össze egy objektumba. Kizárólag az adatok reprezentációjára és az alrendszerek közötti megosztására vannak felhasználva, semmilyen logikát nem tartalmaznak. A minta elsődleges célja, hogy csökkentsük a kliens oldali hívások számát, a rendszerek között átvitt adatmennyiséget és ezáltal a hálózati forgalmat.

A Mapstruct egy kód generátor, amely a DTO tervezési minta használatát könnyíti meg a fejlesztő számára, annotációs mechanizmus segítségével generálja az átalakításokhoz szükséges osztályokat.

3.2.Adatmodell

A dWelling rendszer központi entitásai Java Bean-ként vannak reprezentálva, olyan POJO-k (Plain Old Java Object), amelyek privát adattagokkal, az adattagokhoz tartozó publikus getter és setter metódusokkal, valamint publikus paraméter nélküli konstruktorral vannak ellátva és szerializálhatóak.

A központi entitások az edu.codespring.dwelling.backend.model csomagban kaptak helyet. Mivel a rendszer adathozzáférési rétegként Java Persistence API (JPA) specifikációt implementáló ORM keretrendszert használ, ezért az entitások JPA annotációkkal vannak felannotálva.

A BaseEntity absztrakt osztály áll a modellhierarchia legfelsőbb fokán, implementálja a java.io.Serializable interfészt, a rendszerben felhasznált összes entitás ősosztálya. Ellátja az entitásokat egy egyedi azonosítóval (id) és két dátum mezőt vezet be: az egyik az entitás létrejöttének, a másik az entitás utolsó módosításának megfelelő időpecsétet tárol.

A rendszer működésében a legfontosabb szerepet betöltő entitások közé tartozik a Therapist entitás. Az osztály példánya egy terapeutát reprezentál, a hozzá tartozó specifikus információkkal. A TreatmentRecord osztály példánya azokat az adatokat fogja össze, amelyek a páciens kezeléséhez kapcsolódnak. Ezek az információk csakis az adott terapeuta és páciens által ismertek, ha a pácienst mást terapeuta is kezeli, akkor annak a kezelési folyamatnak szintén van egy külön TreatmentRecord példánya. A Patient entitás reprezentál egy pácienst a rendszeren belül, tartalmazza a beteghez tartozó információkat, melyek minden terapeuta számára elérhetőek (pl. a beteg kórlapja). Egy másik fontos entitás a Therapy, ez reprezentál egy terápiát, tartalmazza a hozzá tartozó leírást, illetve a kapcsolódó képeket. A munkabeosztást és a foglalásokat összefogó entitás szintén egy fontos elem a rendszerben. A TherapistCalendar példányok segítségével valósul meg a terapeuta munkaprogramjának, valamint a terapeutához érkező időpont foglalásoknak a tárolása.

3.3.Kommunikáció

A dWelling szerverének szolgáltatásai RESTful webszolgáltatásokon keresztül érhetőek el a külvilág számára. A REST (Representational State Transfer) egy szoftverarchitektúra típus, melyet Roy Fielding vezetett be doktori értekezésében. A REST HTTP analógián alapszik és működése általában szintén HTTP protokollra épül. A szerver által elérhetővé tett erőforrásokat a kliensalkalmazások adott konvencióknak megfelelő URI-k alapján azonosítják és a HTTP analógia alapján meghatározott műveletek (GET, POST, PUT, DELETE) alapján érhetik el vagy módosíthatják azokat. Az adatok küldése és fogadása a kliens és szerver között JSON objektumok formájában történik.

A dWelling szerver esetében a különböző kéréseket kezelő osztályok az edu.codepsring.dwelling.backed.rest csomagban találhatóak. Ezek az osztályok a Spring Web MVC által definiált @RestController és @RequestMapping annotációkkal vannak ellátva. Az annotációk szerepe, hogy egy adott URI-t az illető osztályhoz társítsanak. Pl:

```
@RestController
@RequestMapping(“/api/therapies”)
public class TherapyController {
        …
}
```

A példában felhasznált @RestController annotáció tulajdonképpen a @Controller és @ResponseBody annotációkat vonja egybe. A szerverhez beérkező, /api/therapies URI-re küldött kéréseket a TherapyController hivatott kiszolgálni, ezek a kérések a rendszer által támogatott terápiákra vonatkoznak. Az osztály szintjén a @RequestMapping annotációval bejelentett URI tovább bővíthető, és minden metódus esetében külön definiálható, hogy az adott metódus milyen típusú HTTP kérések esetében hívódjon meg. A dWelling fejlesztőcsapata betartotta a HTTP szabvány által megfogalmazott metódusok jelentéseit, a GET kérésekre a szerver válaszként a kért információt küldi vissza, ha PUT kérés érkezik, akkor az üzenetben megtalálható módosítások végrehajtódnak a szerver oldali modell objektumon, a POST kérés esetén egy új entitás kerül be a rendszerbe, a DELETE kérésnél törli a rendszer a megadott entitást. A TherapyController osztály searchTherapy metódusa szemlélteti a fent leírtakat:

```
@RequestMapping(value = “search/{query}”, method = RequestMethod.GET)
public ResponseEntity<List<TherapyDTO>> searchTherapy(
                                                @PathVariable(“query”) String query) {
        …
}
```

A metódus által nyújtott szolgáltatást az api/therapies/search/{query} URI-en keresztül érhetjük el. Látható, hogy az osztály szintjén megadott URI kibővült a /search/{query} utótaggal. Az URI-ban paramétereként megadott {query} a @PathVariable annotáció felhasználásával beépül a metódus paraméter listájába. A metódus feladata, hogy a kérésben megjelenő karaktersorozat alapján kikeresse a megfelelő terápiákat, az eredményből összeállítson egy listát és elküldje a válaszra váró félnek. A metódus a hozzá társított műveletet a szolgáltatási réteg megfelelő metódusainak meghívása által végzi el.

A szerver és a kliens közötti kommunikáció során DTO-kat küld a rendszer, így garantálva azt is, hogy a szerver által használt entitások bizonyos információi (pl. jelszavak) nem hagyhatják el a szervert. Minden modell osztálynak van egy neki megfelelő DTO osztálya, amik nem tartalmazzák a „kritikus” információkat (pl. Therapy -> TherapyDTO). Az entitások átalakítása a MapStruct segítségével valósul meg. A kontroller osztályok a szolgáltatási rétegtől kapott adatokat átalakítják DTO-ká, majd ezeket küldik el a kliensnek, a klienstől kapott DTO-kat pedig átalakítják modell objektumokká, ezeket továbbítják a szolgáltatási réteg irányába.

A válasz küldésekor fontos a HTTP szabványban meghatározott állapotkódok (status codes) helyes beállítása. A szerver minden válaszadás esetében egy ResponseEntity objektumot épít fel. Az objektumban helyet kap a válaszüzenet és az állapotkód. Minden sikeresen végrehajtott kérés esetében a 200-as (OK) állapotkód kerül beállításra. A Spring MVC a ResponseEntity objektumból fogja felépíteni a HTTP szabványnak megfelelő válaszüzenetet. A szerver támogatja több állapotkód megfelelő kezelését, pl.: 401 (Unauthorized), 403 (Forbidden), 404 (Not Foud), 423 (Locked) stb. A Spring Web MVC gondoskodik arról, hogy szerverhiba esetén az 500 (Internal Server Error) állapotkóddal ellátott üzenetet küldje vissza válaszként a várakozó félnek. Továbbá gondoskodik a keretrendszer arról is, hogy ha egy olyan kérés érkezik a szerverhez, amelyhez nincs társítva megfelelő kezelő osztály és metódus, akkor 404-es állapotkód legyen beállítva a válaszüzenetben.

3.4.Adathozzáférési réteg

Az adathozzáférési réteg feladata, hogy kapcsolatot létesítsen különböző típusú adatbázisokkal, elvégezze az adatbázisok és a projekt közötti adatátvitelt. A dWelling projekt az adatok tartós tárolása érdekében egy MySQL adatbázist használ, továbbá az adatok egy részhalmazát az Elasticsearch keresőmotor is tárolja.

A Spring keretrendszer a fejlesztőcsapatnak segítséget nyújtott a perzisztencia réteg megvalósításában. A Spring Data keretrendszer lehetőséget ad több fajta adattároló támogatására, úgy, hogy a projekt szintjén egy egységes kezelési módot biztosít a fejlesztőknek. A dWelling projekt esetében a Spring Data JPA és a Spring Data Elasticsearch modulok voltak felhasználva a keretrendszerből. A Spring Data JPA hivatott biztosítani a Hibernate integrálását a projektbe, továbbá egy plusz absztrakciós réteget képez az ORM keretrendszer felett, amely megkönnyíti az adathozzáféréssel kapcsolatos műveletek megvalósítását. A projekt esetében az adathozzáférési réteget alkotó interfészek az edu.codespring.dwelling.backend.repository csomagon belüli jpa és elasticsearch csomagokban kaptak helyet.

A jpa csomagban található komponensek a JpaRepository interfészből származnak. Ez az interfész a Spring Data legfontosabb interfészének, a Repository interface-nek a negyedik rendbeli leszármazottja.

```
public interface TreatmentRecordRepository extends JpaRepository<TreatmentRecord, Long>{
    TreatmentRecord findByPatient_IdAndTherapist_Id(Long patientId, Long therapistId);
}
```

A példából jól látszik, hogy a lekérdezések deklarálása tömören és egyszerűen megvalósítható. Az olyan esetekben, amikor nem előnyös ezzel az elnevezési konvenciókon alapuló módszerrel deklarálni a megfelelő lekérdezést, akkor a @Query annotációval megadhatjuk a megfelelő JPQL (Java Persistence Query Language) parancsot.

Az elasticsearch csomagban található komponensek az ElasticsearchRepository interfészből származnak, amely szintén a Repository interface leszármazottja.

3.5. Szolgáltatás réteg

A dWelling szolgáltatási rétege tartalmazza az üzleti logikával kapcsolatos műveletek megvalósításait, továbbá általa érhetőek el az adathozzáférési réteg szolgáltatásai. Az edu.codespring.dwelling.backend.service csomag tartalmazza a különböző entitásokhoz tartozó szolgáltatások megvalósításait, azaz a szerver oldali üzleti logikáért felelős komponenseket. A szolgáltatásokat a szerverhez érkező kérések fogadásáért és kiszolgálásáért felelől komponensek használják. A szerviz osztályok metódusai ellenőrzik a beérkező paraméterek helyességét, ezt követően a kérésnek megfelelő feldolgozást végeznek rajtuk, majd a választ továbbítják a hívó félnek. Az ellenőrzés és feldolgozás során felhasználják az adathozzáférési réteg szolgáltatásait.

A szolgáltatási réteg osztályai @Service annotációval ellátott Spring bean-ek. A @Service annotáció egy specifikusabb változata a @Component annotációnak, melynek szerepe, hogy jelezze a component-scanning mechanizmus során, hogy a keretrendszernek menedzselnie kell az illető szerviz osztályt. A kivételek átlátható kezelése érdekében a réteg egy saját kivételtípust határoz meg, az edu.codespring.dwelling.backend.service.exception csomagban található a ServiceException osztály, amely a RuntimeException osztályt terjeszti ki. Egy kivétel felléptekor a rendszer naplózza a kivételt a pontos típusának megfelelően, majd a felsőbb réteg számára már ezt az általánosabb, réteg-specifikus kivételt dobja tovább. A naplózást a rendszer SLF4J által biztosított absztrakciós szinten keresztül LOG4J naplózási keretrendszerrel valósítja meg.

A szerviz osztályok közül megemlíthető például a TherapyService, amely searchTherapy metódusával kulcsszavak alapján képes különböző terápiákat ajánlani a rendszer felhasználóinak. Ennek megvalósítására teljes szöveg alapú keresést használ. Továbbá kiemelhető az EventService is, melynek szolgáltatásai lehetővé teszik új foglalások létrehozását és a már meglévő foglalások menedzselését.

3.6. Biztonság

A rendszerben négy felhasználói szerepkör létezik. A legkevesebb jogosultsággal az anonim vendégfelhasználók rendelkeznek, ezen kívül létezik a páciens és terapeuta szerepkör, illetve az adminisztrátor felhasználók, akik az oldal karbantartásáért felelősek.

Fontos volt, hogy a rendszerben szereplő terapeuták csak azokat a pácienseket láthassák, akiket már kezeltek, illetve az, hogy a páciensek ne férhessenek hozzá más páciensek adataihoz. Külön kellett kezelni a páciensek terapeuták számára publikus adatait, amelyek bármely arra jogosult terapeuta által megtekinthetőek és szerkeszthetőek. Az autorizációs mechanizmus implementációja a Spring Security keretrendszeren alapszik.

3.6.1. Spring Security

A Spring Security keretrendszer előre megírt, könnyen beépíthető biztonsági megoldásokat tartalmaz. Ha egy testreszabott megoldást implementálunk, a keretrendszer alkotóelemei könnyen kiegészíthetőek, lecserélhetőek. Spring Bean-ként implementálhatjuk a megfelelő interfészt, majd bekonfigurálhatjuk az alapértelmezett megoldás helyett.

A dWelling rendszer JWT token alapú autentikációs mechanizmust implementál, a szerveren semmilyen állapotinformáció nincs tárolva, így a kommunikációs mechanizmus teljesen megfelel a REST modellben meghatározott alapelveknek.

3.6.2. Autentikáció

Az autentikáció során a felhasználó elküldi a bejelentkezési adatait, melyeket a szerver validál, majd létrehoz egy JWT tokent. A tokent Secure, Http-Only cookie-ként állítja be a válaszban. A JWT token-t a válasz testében is vissza lehetne küldeni, majd kliens oldalon perzisztálni, azonban ez a megoldás biztonsági kockázatot jelent. A Http-Only cookie-k Javascript-ből nem elérhetőek, így a megoldás jóval biztonságosabb.

A Spring által adott User osztály és a projektben található User entitás közötti megfeleltetést egy saját UserDetailsService interfészt implementáló komponens valósítja meg. Ez a komponens az adatbázisból kinyeri a felhasználónévnek megfelelő dWelling User entitást. Az entitást átalakítja egy Spring UserDetails-t implementáló objektummá.

A JWT token minden kérés során validálva lesz, ezt a feladatot egy saját JWTFilter Spring komponens végzi. A JWTFilter komponens kiterjeszti a GenericFilterBean absztrakt osztályt, amelynek feladata az, hogy elkapja a Servlet kérést és továbbítsa a kérést a következő filter osztály felé. A JWTFilter osztály validálja a kérést, majd a tokenben található információ alapján beállítja a SecurityContextHolder context adattagját. A SecurityContextHolder-ből a kérés ideje alatt bármikor lekérdezhetőek az aktív felhasználóra vonatkozó információk.

3.6.3. Autorizáció

Az autorizáció Spring annotációk és az SpEL (Spring Expression Language)-el van megvalósítva. A @PreAuthorize annotáció segítségével felhasználói szerepkör alapján adhatunk hozzáférést. A PreAuthorize az annotáció értékeként SpEL kifejezéseket kaphat. Ilyen kifejezések a hasAuthority, hasAnyAuthority stb. Ezeket a kifejezéseket a Spring Security értékeli ki a SecurityContextHolder Authentication objektuma alapján, és 403-as (hozzáférés megtagadva) hibakódot térít vissza, ha a kiértékelés hamis eredmény ad.

A rendszer fejlesztése során bizonyossá vált, hogy a szerepkör alapú autorizáció nem lesz elégséges. Például, mikor egy terapeuta próbál elérni egy pácienst, szükséges annak a vizsgálata, hogy a terapeuta "ismeri"-e a pácienst vagy sem. Ezeket az ellenőrzéseket saját osztályok végzik, és a metódushívás után vannak végrehajtva. Ha a felhasználónak nincs joga az erőforrást megtekinteni/módosítani, akkor egy kivételt váltunk ki, amely 403-as hibakóddal tér vissza a kliens számára.

4. Web kliens

A dWelling projekt kliens oldali alkalmazása egy AngularJs keretrendszerre épülő dinamikus webes alkalmazás. A kliensalkalmazás RESTful webszolgáltatásokon keresztül kommunikál a szerver oldallal.

4.1.Felhasznált technológiák

4.1.1. AngularJS

Az AngularJS [5] egy nyílt forráskódú, a Google által fejlesztett, JavaScript alapú keretrendszer, amely lehetőséget nyújt dinamikus webes alkalmazások fejlesztésére. A HTML leíró nyelv eszköztárát kibővíti az általa létrehozott alkotóelemekkel. Ezeket az újonnan létrehozott alkotóelemeket direktíváknak nevezik. A keretrendszert SPA (single-page application) típusú alkalmazások fejlesztésére tervezték.

Az AngularJS által fejlesztett applikációk az MVC (Model View Controller) szoftvertervezési mintát követik, vagy ennek valamilyen speciálisabb változatát. A keretrendszer kötelez minket a minták követésére, ezáltal a projektünk átláthatóbbá válik, elkülönül egymástól a megjelenítés, az adatmodell és a vezérlés.

A keretrendszer biztosítja az adatkötést (data-binding), összeköti az adatmodellt a nézettel. A modell az egyetlen olyan egység, amely meghatározza az alkalmazás állapotát. A nézet a modell megjelenítése. Az adatkötésnek köszönhetően bármilyen változás a nézetben azonnal tükröződik a modellben is, ha a modellben történt változás, akkor az is látható a nézeten. Továbbá a keretrendszer biztosítja az alkalmazáskomponensek közötti függőségek menedzsmentjét, a Dependency Injection minta alapján.

4.1.2. TypeScript

A TypeScript [6] a Microsoft által fejlesztett objektumorientált programozási nyelv. A fejlesztés célja, hogy könnyebbé váljon az összetett, nagy méretű alkalmazások fejlesztése. A TypeScript-ben megírt kódot egy fordító JavaScript-re fordítja.

A TypeScript lehetőséget ad arra, hogy osztályokkal dolgozzunk, támogatja az öröklődést is. Bevezeti a típus fogalmát, ezáltal az adatmodellünk átláthatóbbá, biztonságosabbá válik, elkerüljük a típus inkompatibilitások előfordulását.

A nyelv támogatja, hogy már meglévő AngularJS komponenseket is felhasználjunk a fejlesztésben, a kompatibilitás biztosításáért egy definíciós fájl felel. A definíciós fájlok biztosításáért, karbantartásáért a „The TypeScript Definition Manager” a felelős.

4.2.Adatmodell

A web kliens adatmodellje TypeScript osztályok segítségével van felépítve. A szerver komponenshez hasonlóan itt is megjelennek a központi entitások, amelyek a rendszer működéséért felelősek, pl.: Therapy, Therapist, Patient. Az entitások néhány specifikus mező kivételével megfelelnek a szerver oldali adatmodellben definiált párjuknak.

4.3.Kommunikáció

A szerverrel való kommunikáció implementációjának megkönnyítésére a csapat a Restangular modult hívta segítségül. A Restangular [7] egy AngularJS szolgáltatás, melynek szerepe, hogy egyszerűsítse a gyakran használt GET, POST, PUT, és DELETE típusú aszinkron kérések megvalósítását, egy minimális kliens oldali kód implementálásával. A web kliens esetében a különböző URI-ra irányuló kérések külön osztályokban vannak implementálva. Például a TherapyRepository osztály felelős minden olyan művelet megvalósításáért, amelyek a terápiákhoz kapcsolódnak. A kommunikációt megvalósító osztályok metódusai a hívó fél számára a visszajelzést IPromise-okon keresztül végzik, callback függvények segítségével. A kliens alkalmazás esetében a kontroller és a kommunikációért felelős réteg közé beépül egy szerviz réteg, ez egy átmenetet képez az előbb említett két réteg között továbbá a beérkező adatok manipulálását végzi. Tekintsük a következő példát, mely a TherapyRepository osztály getAllTherapies metódusát mutatja be:

```
public getAllTherapies():IPromise<Therapy[]> {
    var deferred = this.$q.defer();
    this.Restagula
        .all('therapies')
        .all('all')
        .getList()
        .then(response => {
            deferred.resolve(response);
        })
.       .catch(error => {
            deferred.reject(error);
        })
    return deferred.promise;
}
```

A fenti kódrészlet szerepe, hogy a szervertől lekérje az általa ismert terápiákat. Az /api/therapies/all URI-re küld egy GET kérést. A szerver válaszként egy terápiákból álló listát terít vissza. A JSON formátumban megkapott válaszból a Restangular Therapy objektumokat épít fel, melyeket egy listába gyűjt.

4.4.Biztonság

A web kliens három különböző szerepkörrel rendelkező felhasználó típust támogat (PATIENT, THERAPITS, ADMIN), továbbá egy általános felületet biztosít a vendég felhasználók számára. A szerepköröknek megfelelően a felhasználók különböző felületeken keresztül érhetik el azokat a szolgáltatásokat, melyekre jogosultak. A felhasználók nézetei különböző állapotokhoz (state) vannak kötve, az állapotok közti navigációt az AngularUI Router [8] hivatott elvégezni. Az AngularUI Router AngularJS felhasználásával készült SPA alkalmazásokon belüli navigálást vezérlő keretrendszer. A böngésző URL-jét frissíti, annak függvényében, hogy miként navigálunk a webes alkalmazáson belül.

A dWelling projekt webes alkalmazásának esetében minden állapothoz hozzárendelődik egy lista, amelyben azok a szerepkörök szerepelnek, amelyeknek hozzáférése van az adott állapothoz. Ha ez a lista üres, az azt jelenti, hogy az adott oldalt bárki megtekintheti. A következő példa szemlélteti egy állapot definiálását:

```
export const PatientState:IState = {
        url: '/patient/{patientId}',
        template :'<patient patient-id="sctrl.$stateParams.patientId"
                        patient="sctrl.$stateParams.patient"><patient>',
        controller: PStateController,
        controllerAs: 'sctrl',
        data: {
                role: ["THERAPIST"]
        }
};
```

A példában látható állapot csak a THERAPIST szerepkörrel rendelkező felhasználók számára érhető el, a template mezőben egy nézet van hozzárendelve az állapothoz, ez fog megjelenni a felhasználó számára, amikor a /patient/{patientId} URL-re navigál. A kliens oldali Principal entitás felelős az aktuális felhasználó adatainak a tárolásáért és karbantartásáért. Fontos kiemelni az entitás authentication metódusát. Ez a metódus felelős azért, hogy ne engedje olyan állapotok elérését az adott felhasználónak, amelyhez nincs jogosultsága. A metódus minden állapotváltás esetében meghívásra kerül, a meghívását az Angular $stateChangeStart eseményére feliratkozott metódus végzi. Ha olyan oldalra akar navigálni a felhasználó, amelyhez nincs jogosultsága, akkor a metódus átirányítja a szerepkörének megfelelő alapértelmezett oldalra.

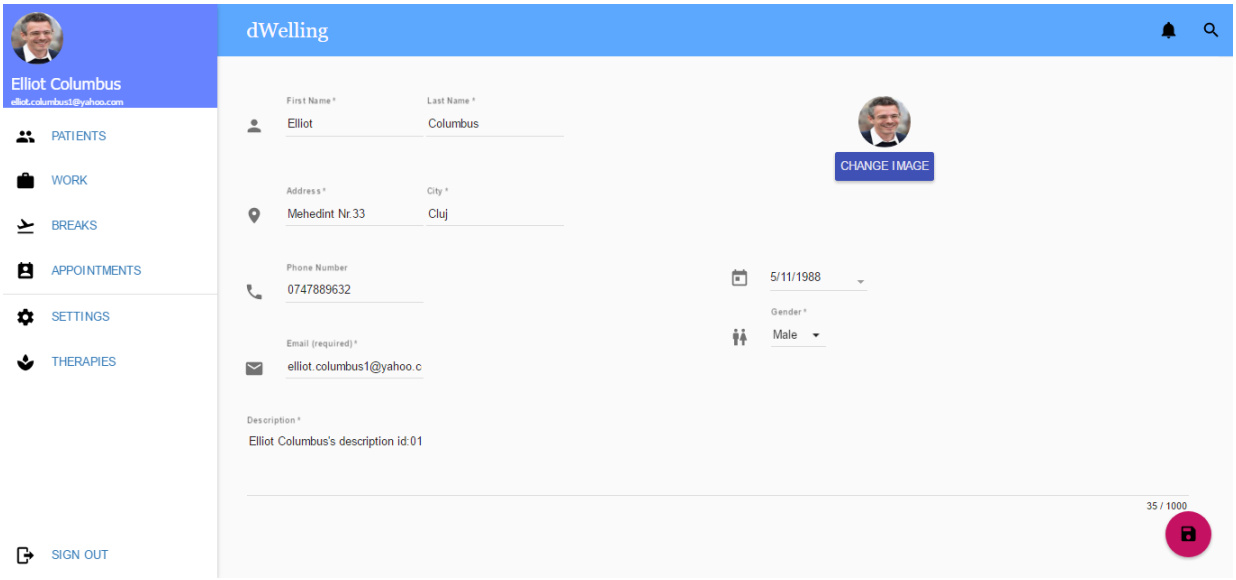
Azokban az esetekben, amikor a felhasználó olyan tartalmat szeretne elérni, amelyhez nincs jogosultsága a szerver válaszként egy 403-as állapotkóddal ellátott HTTP válaszüzenetet térít vissza. Az ilyen eseteknek a kezelését egy HTTP interceptor hivatott elvégezni. Ez az interceptor minden olyan HTTP válaszüzenetet elkap, amelyiknek a hibakódja 403-as, a felhasználót átirányítja a szerepkörének megfelelő alapértelmezett állapotra, és egy hibaüzenetet jelenít meg a számára.

5. Az alkalmazás működése

Az alkalmazás mindhárom szerepkör számára ugyanazt a felületet biztosítja különböző funkciókkal. A fejezet képernyőképekkel illusztrálja ennek a felületnek a működését.

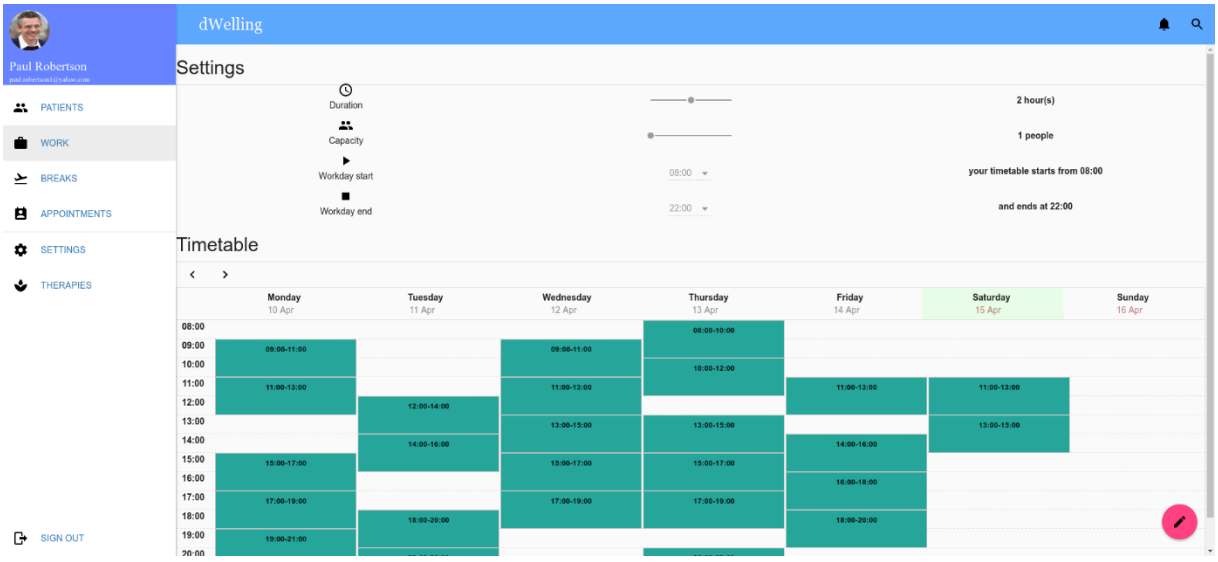
5.1. A terapeuták számára elérhető funkciók

A terapeuták az első belépést követően kitölthetik profil adataikat, profil képet tölthetnek fel, egy rövid leírást és különböző elérhetőségeiket adhatják meg.



2. ábra Terapeuta profil szerkesztő

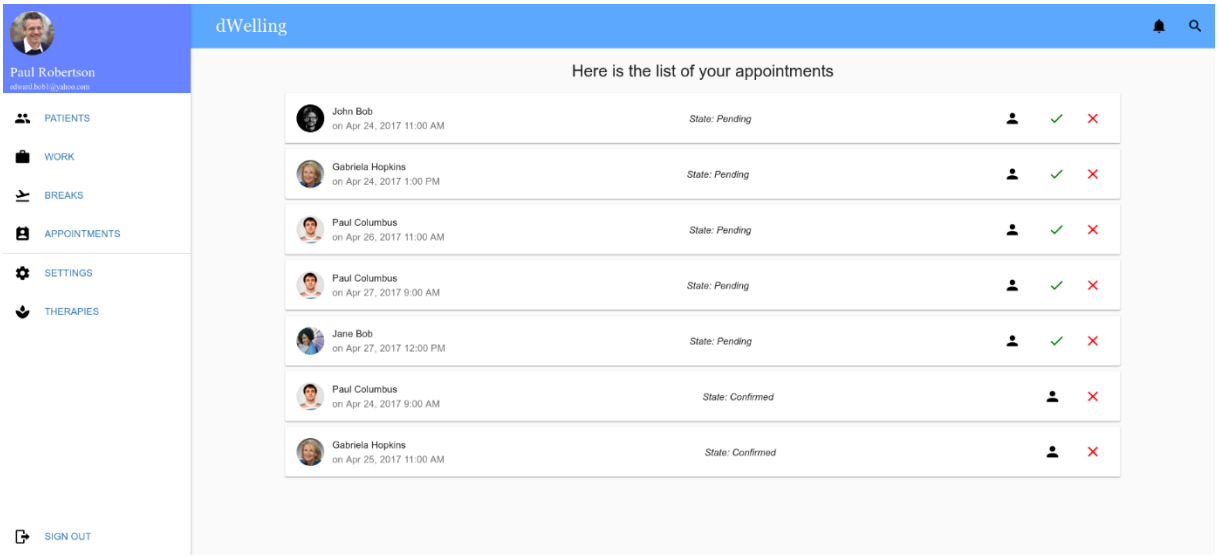
A következő lépés a munkaprogram meghatározása. Itt meghatározható a kezelések hossza, illetve, hogy egyszerre hány pácienst fogad egy kezelésen. Megadható, hogy az órarend hánytól hányig legyen érvényes. Az előbbi információk alapján egy naptár segítségével megadható a munkaprogram egy hétre.



3. ábra Terapeuta munkaprogram szerkesztő

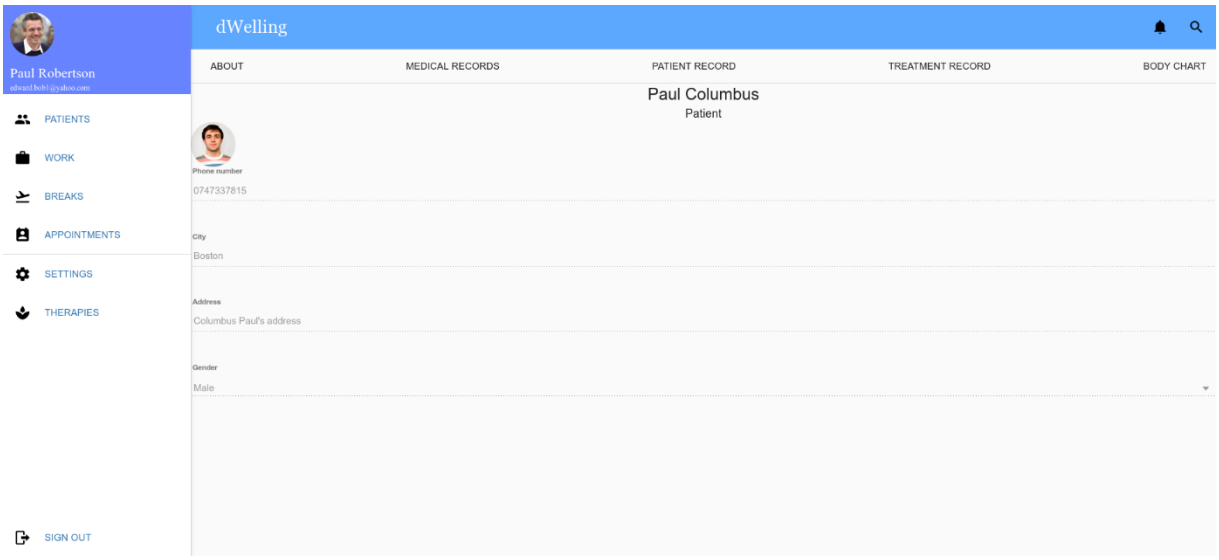
A terapeutáknak szüneteket is megadhatnak, amikor a pácienseik számára nem lesz elérhető a foglalás lehetősége. Ez az opció a kiválasztott időszakra felülírja a munkaprogram nézetben található órarendet.

A terapeuták megtekinthetik a foglalásokat, elfogadhatják vagy elutasíthatják ezeket.

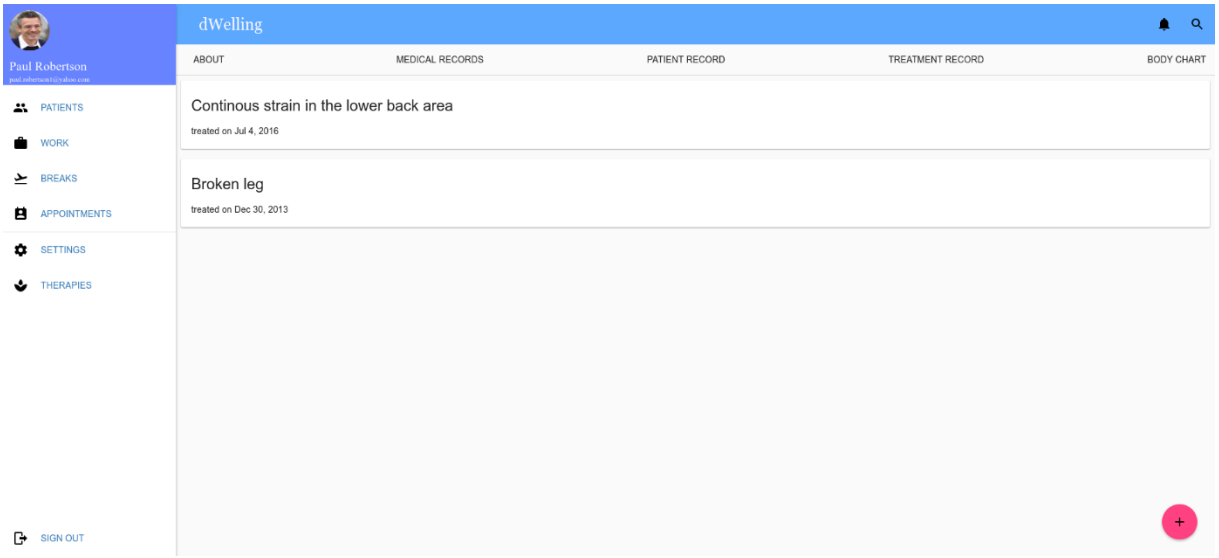


4. ábra Terapeutához érkezett foglalások listája

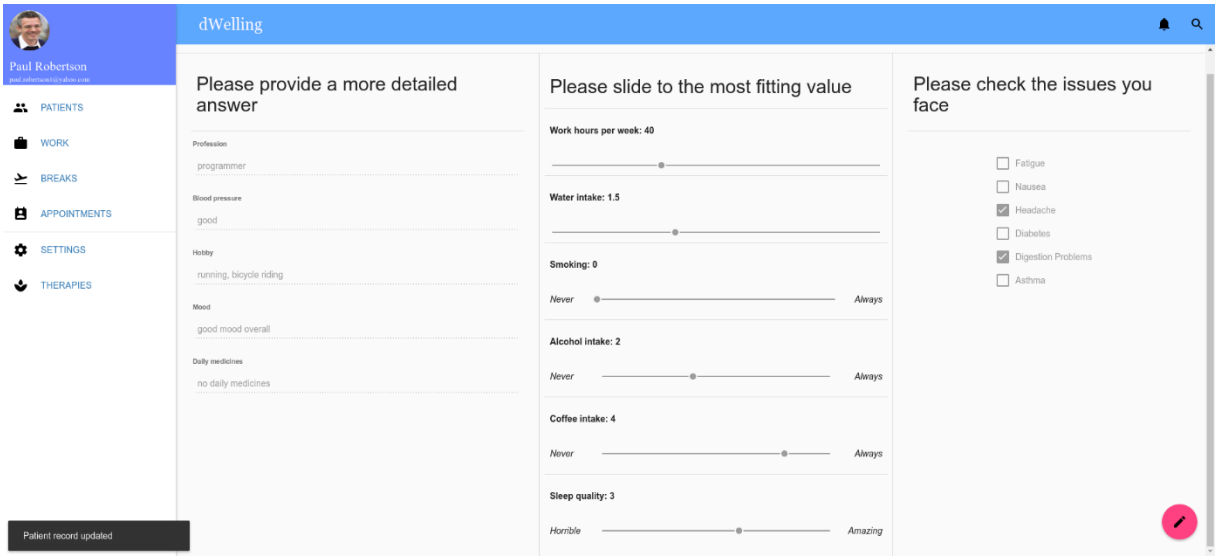
Megtekinthetőek azok a páciensek, akiknek már volt foglalásuk az adott terapeutához. Itt elérhetővé válnak a páciens publikus adatai (5. ábra), elérhetőek lesznek a páciens orvosi leletei (6. ábra), és a rá vonatkozó általános egészségügyi információk (7. ábra).



5. ábra Páciens adatai

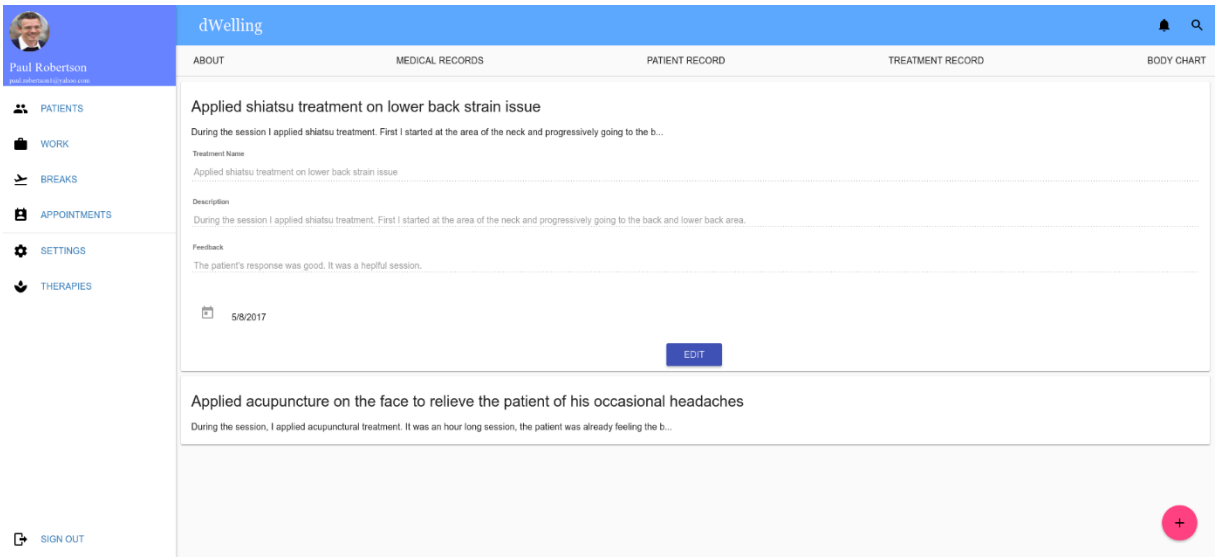


6. ábra Orvosi leletek

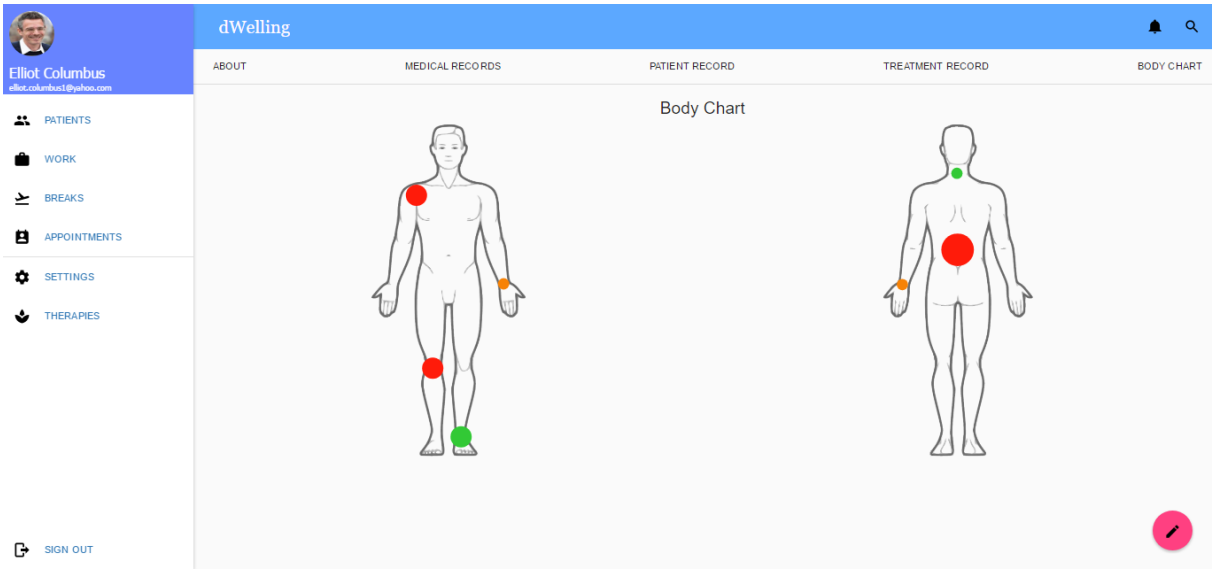


7. ábra Páciens általános egészségügyi információi

A terapeuta felvihet a kezelésekre vonatkozó információkat (8. ábra) és egy virtuális testen fájdalompontokat jelölhet meg, amelyekhez kommentárt is fűzhet (9. ábra).



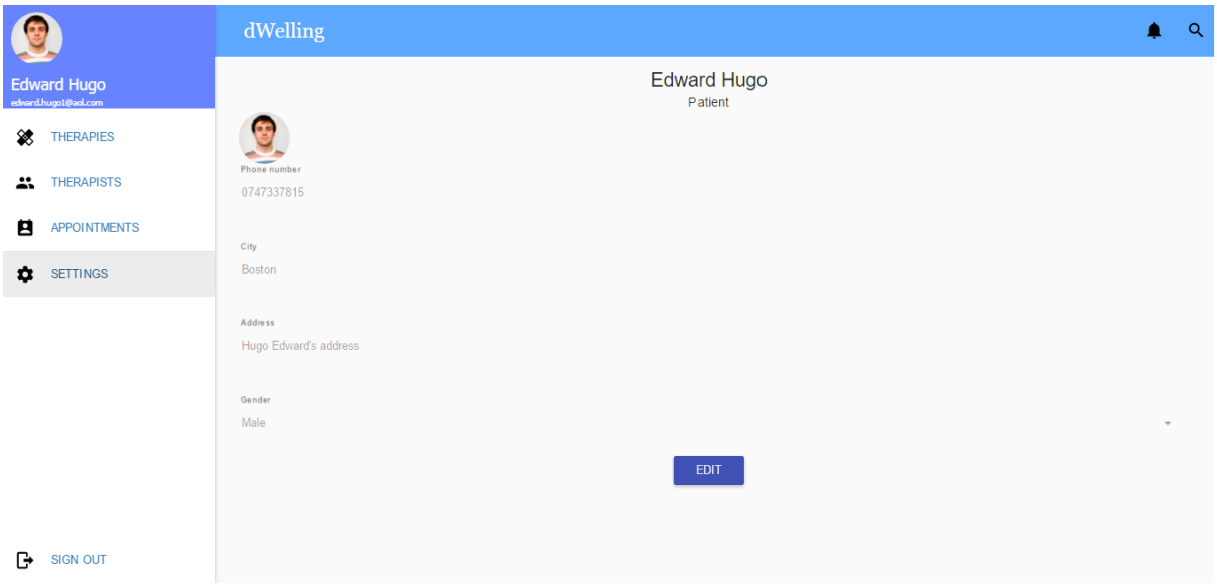
8. ábra Kezelésekre vonatkozó információk



9. ábra Virtuális testre felvitt fájdalompontok

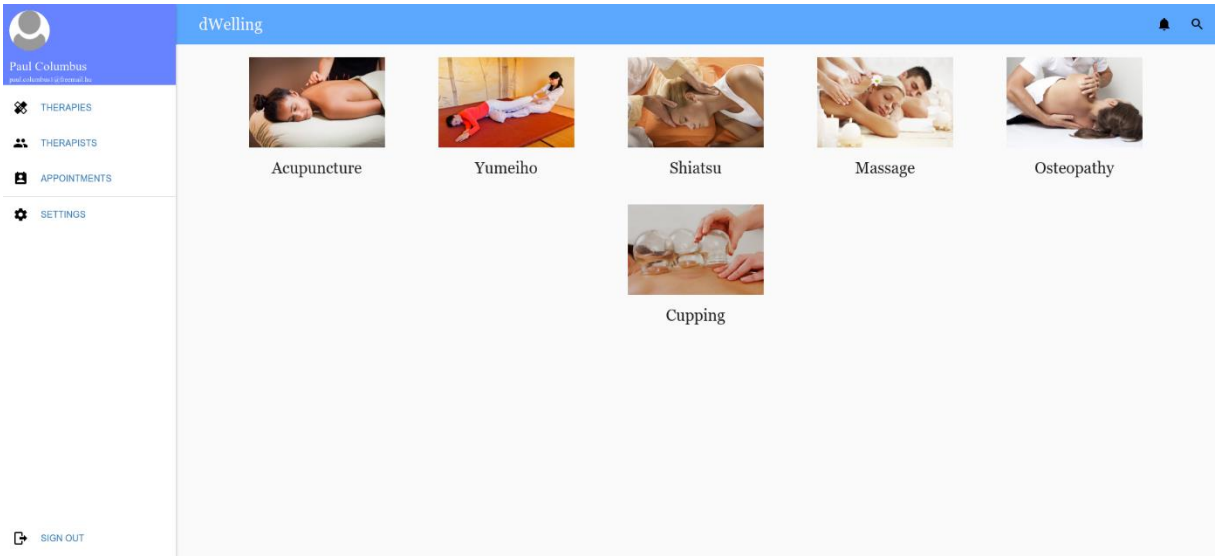
5.2. Páciensek számára elérhető funkciók

A páciensek szerkeszthetik profiljukat, ahol profilképet és elérhetőségeket adhatnak meg (10. ábra).

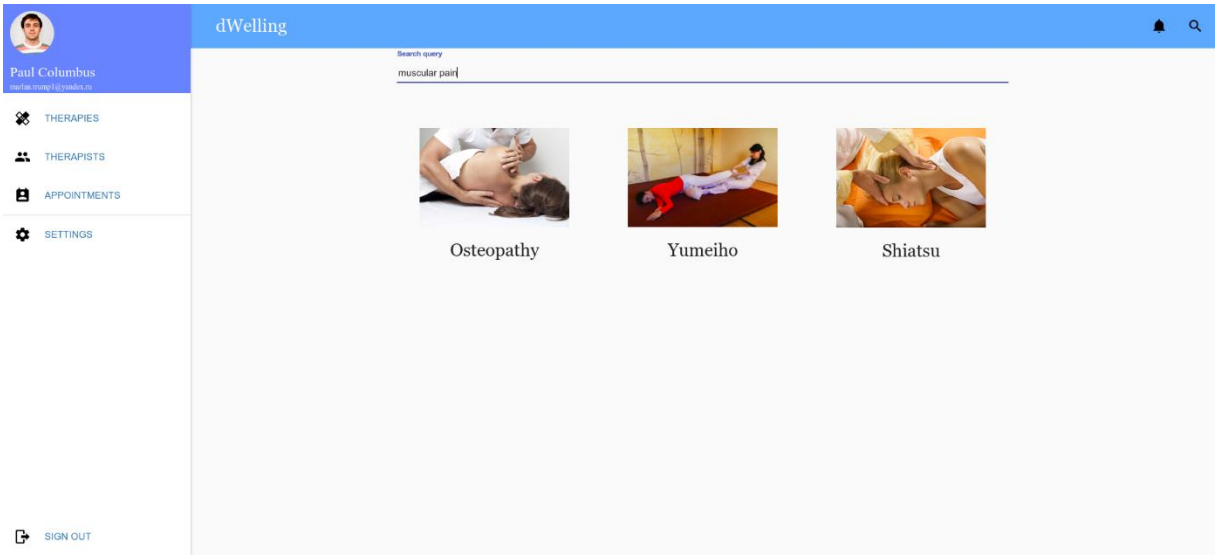


10. ábra Páciens profil adatok

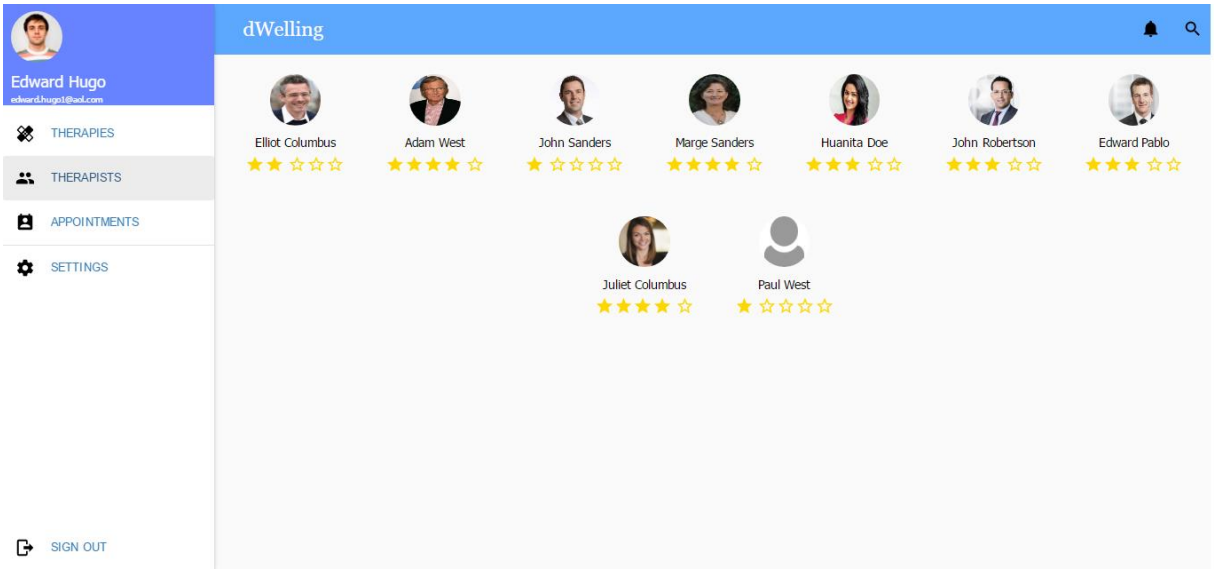
A páciensek megtekinthetik a rendszerben található terápiák listáját (11. ábra), kereshetnek a terápiák között teljes szöveg alapú kereséssel (12. ábra), megtekinthetik a terapeuták listáját (13. ábra).



4. ábra Rendszerben található terápiák

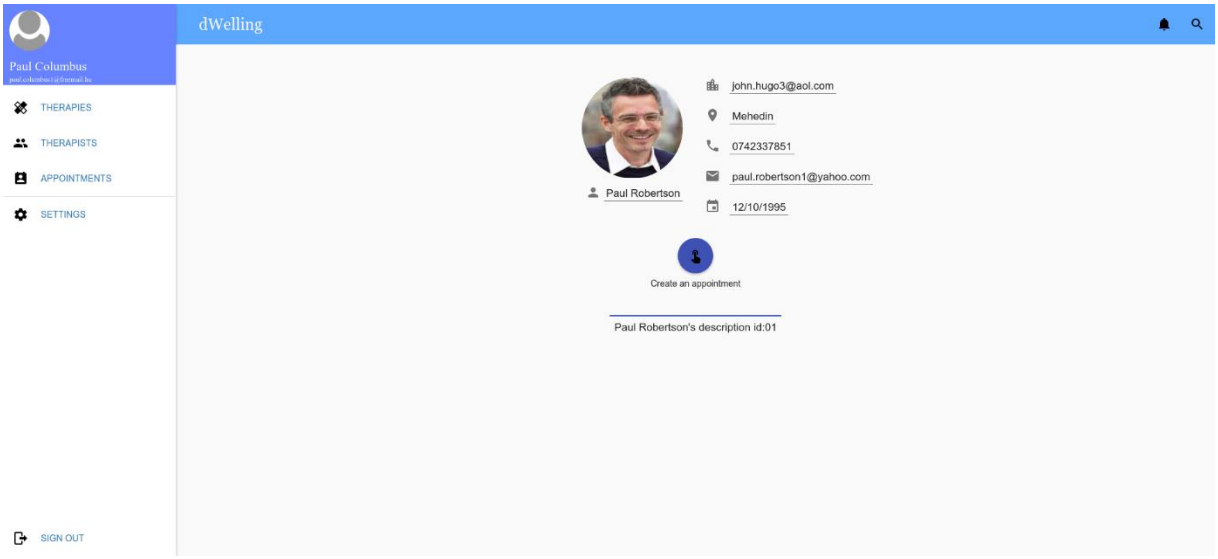


12. ábra Teljes szöveg alapú keresés terápiák esetében

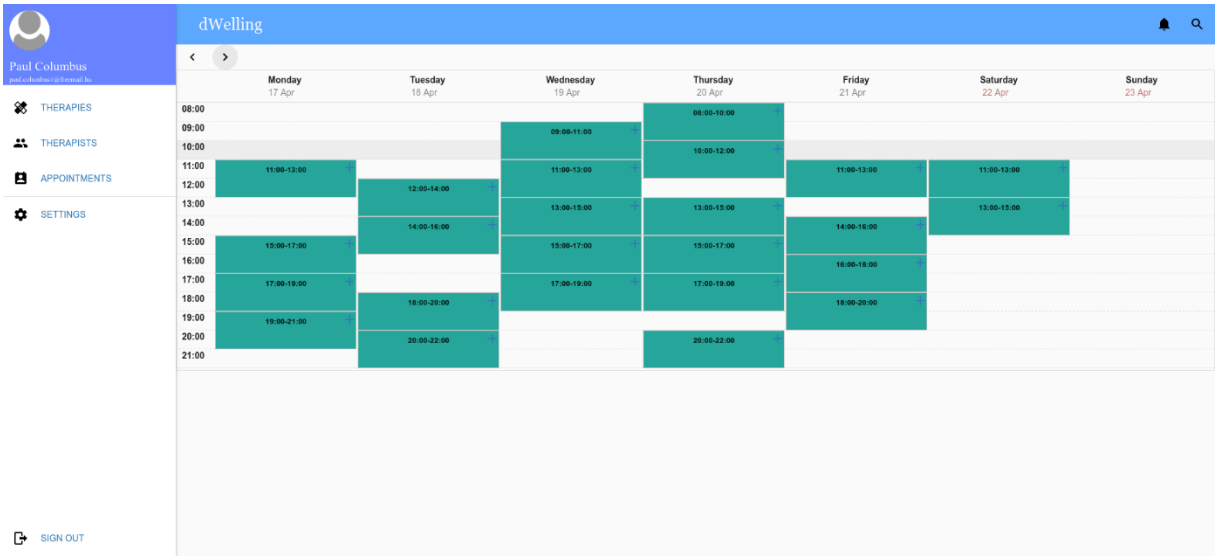


13. ábra A rendszer terapeutáinak listázása

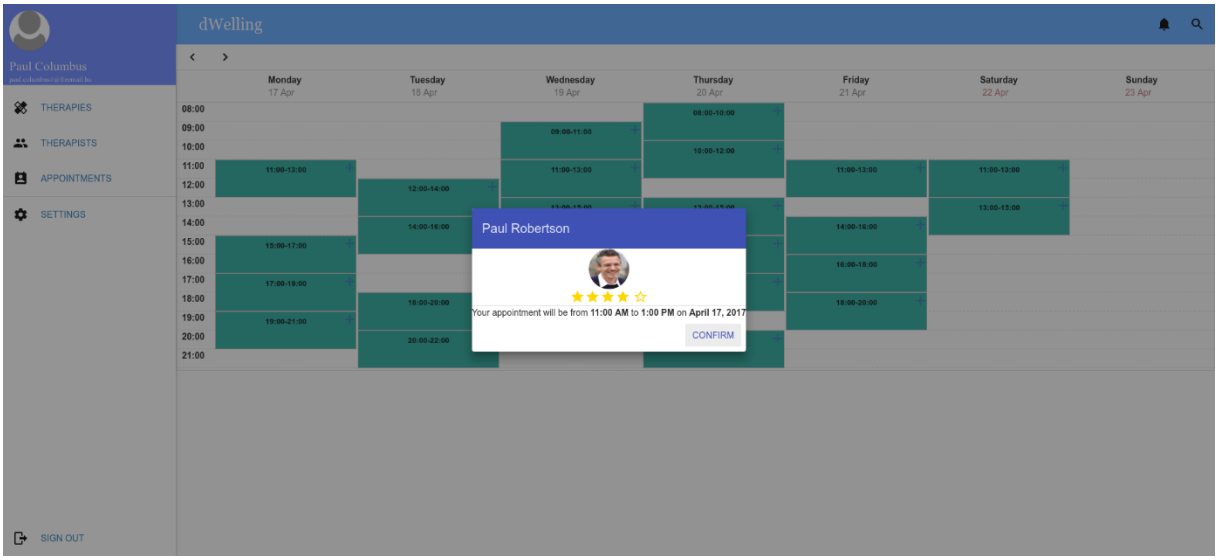
Miután a felhasználó megtalálta a számára megfelelő terapeutát a terapeuta profiljáról (14. ábra) a foglalási nézetre mehet (15. ábra). Miután kiválasztotta a megfelelő időpontot, véglegesítheti a foglalását (16. ábra).



14. ábra Terapeuta profilja a páciens számára megjelenítve



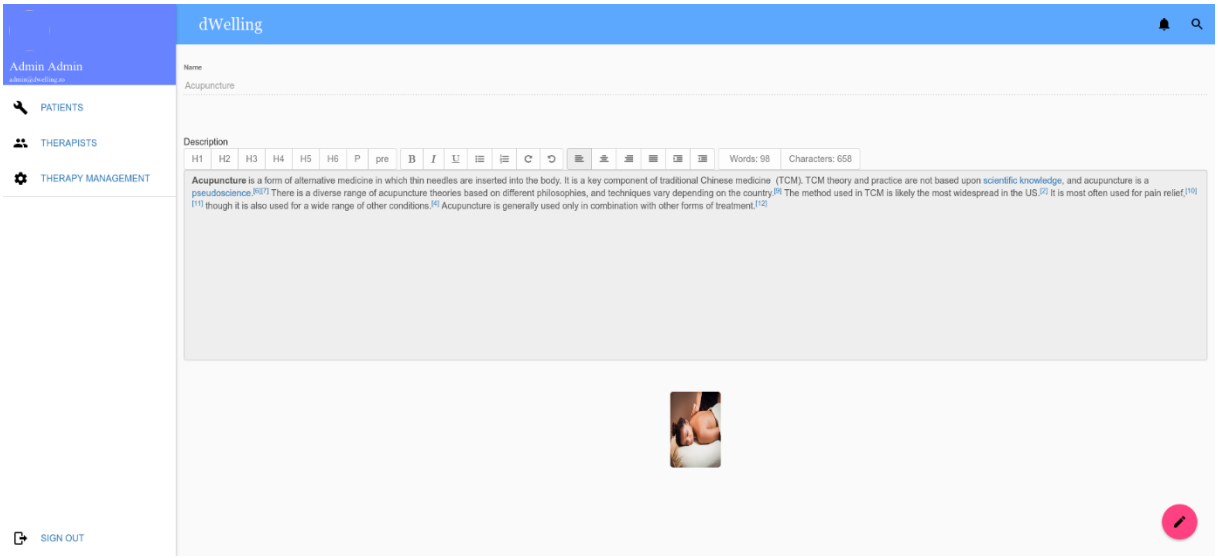
15. ábra Páciens számára elérhető szabad időpontok



16. ábra Kiválasztott időpont véglegesítése

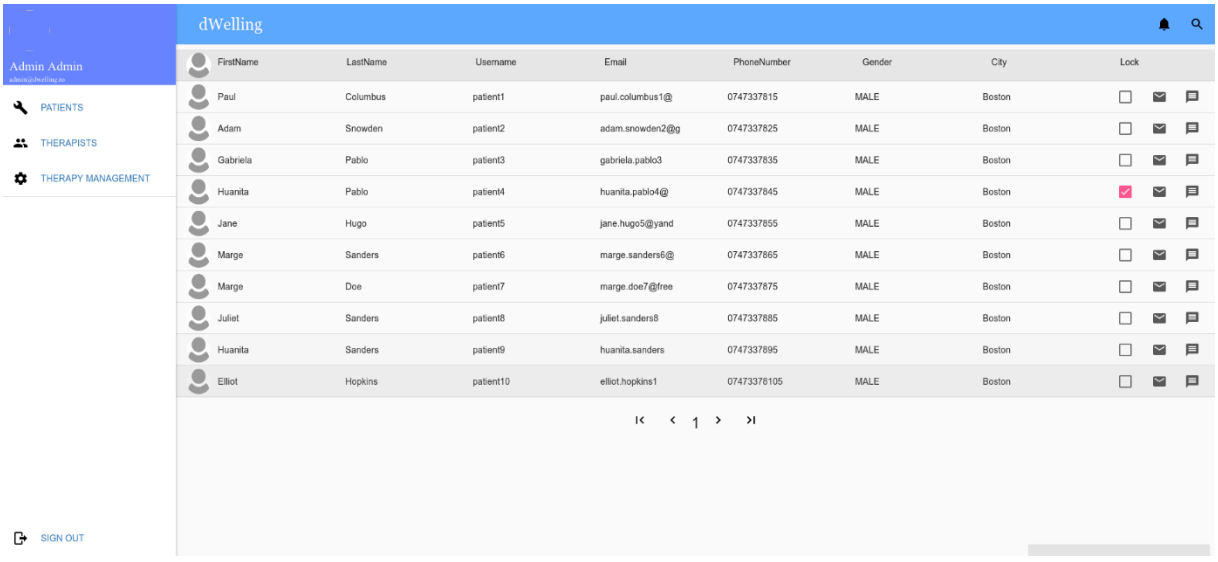
5.3. Adminisztrátorok számára elérhető funkciók

Az adminisztrátorok felelőssége az oldalon található terápiák menedzselése, új terápiák felvitele. A terápiák leírását egy Rich Text Editorral szerkeszthetik az oldalon belül és a galériába képeket vihetnek fel (17. ábra).



5. ábra Terápia szerkesztő

Az adminisztrátorok letilthatnak bizonyos felhasználókat, ha azok megszegik az oldal használatának szabályzatát (18. ábra).



6. ábra Felhasználók listája az adminisztrátor számára megjelenítve, letiltási lehetőséggel

Az adminisztrátorokon keresztül történik az oldalra a terapeuták regisztrációja is.

Következtetések és továbbfejlesztési lehetőségek

A dWelling projekt keretein belül sikerült egy olyan szoftverrendszert létrehozni, amely segítségül szolgálhat egészségügyi problémákkal küszködő emberek és terapeuták számára. Egy webes alkalmazás segítségével a vendég felhasználók megtekinthetik a rendszer által támogatott terápiákat, regisztrálhatnak a rendszerbe. A pácienskénti regisztrált felhasználóknak lehetőségük van jelentkezni a kezelésekre. A terapeuták a rendszer segítségével meghatározhatják munkaprogramjukat, szabályozhatják mindennapi munkabeosztásukat. Felvezethetik a rendszerbe a páciensek korlapjait, kezelésekre vonatkozó adatokat rögzíthetnek. Az időpont kérésről, annak módosulásáról mind a terapeuta, mind a páciens email-ben értesül. Az adminisztrátornak lehetősége van a felhasználók menedzsmentjére, továbbá az ő szerepkörébe tartozik az is, hogy új terapeutát és új terápiát vigyen fel a rendszerbe.

A szoftverrendszer kiegészíthető további funkcionalitásokkal, például:

- Push notification szolgáltatás, melynek segítségével értesíteni lehetne a felhasználókat a különböző eseményekről (foglalás, foglalás elfogadása/elutasítása, hasznos hírek a felhasználó számára)
- Egy belső üzenetküldő rendszer a gyors és személyes kommunikációért.
- A népszerű közösségi oldalak szolgáltatásainak integrálása a projektbe (a közösségi hálók segítségével történő bejelentkezés, naptár szinkronizációja).
- Egy zárt „e-learning” rendszer bevezetése, amely azt a célt szolgálja, hogy a terapeuták különböző leírásokat, képeket, videókat tudjanak megosztani betegeikkel.
- Lehetőséget biztosítani a pácienseknek arra, hogy tudják értékelni a már befejezett terápiákat, illetve a terapeutákat.

Hivatkozások

[1] K. Schwaber and J. Sutherland, *The Scrum Guides*. [Online] http://www.scrumguides.org/ [Utolsó megtekintés dátuma: 2017.04.23]

[2] *Spring Doc Web MVC*, Spring. [Online] https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html [Utolsó megtekintés dátuma: 2017.04.14]

[3] BioMed Team, *Spring Data Elasticsearch.* [Online] http://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html/ [Utolsó megtekintés dátuma: 2017.03.25]

[4] *Elasticsearch Reference*, Elastic. [Online] https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html [Utolsó megtekintés dátuma: 2017.03.25]

[5] AngularJs hivatalos weboldala. [Online] https://angularjs.org/ [Utolsó megtekintés dátuma: 01 03 2017]

[6] S. A. Basarat, *TypeScript Deep Dive*, 2015

[7] F. Boström, *Restangular.* [Online] https://github.com/mgonto/restangular [Utolsó megtekintés dátuma: 16 04 2017]

[8] Austin, *AngularUI Router.* [Online] https://github.com/angular-ui/ui-router [Utolsó megtekintés dátuma: 16 04 2017]