XXI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2018. május 24–27.

E-migrated

Közösségi háló elszármazott szakemberek számára

Szerzők:

Borsay Zsuzsanna
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
Tüzes-Bölöni Kincső
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

dr. Simon Károly, egyetemi adjunktus
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar
Sulyok Csaba, doktorandusz,
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Kivonat

Elterjedt jelenség manapság, hogy Székelyföldről származó szakemberek külföldön vállalnak munkát, és átmenetileg vagy akár hosszabb időre ott telepednek le. A fizikai országhatárok megnyílása óta azonban nem könnyű ezeket a migrációs adatokat naprakészen tartani. Az E-migrated webes alkalmazás többek közt erre a problémára nyújt megoldást. Feltérképezi a nagyvilágban levő, székelyföldi régióból származó szakembereket és a diaszpórát dinamikusan jeleníti meg. Lehetőséget teremt egy közösség létrehozására, ösztönözve a rendszerhez csatlakozókat a kapcsolatépítésre, szakmai jellegű együttműködésre. Virtuális kapocsként szolgál az alkalmazás felhasználóinak, fizikailag a világ bármely pontján legyenek is.

Jelen dolgozat bemutatja az E-migrated alkalmazás architektúráját, beszámol a megvalósítás részleteiről, a felhasznált technológiákról, eszközökről és módszerekről. Ismerteti az alkalmazás funkcionalitásait és néhány esettanulmányon keresztül szemlélteti a rendszer működését.

Tartalomjegyzék

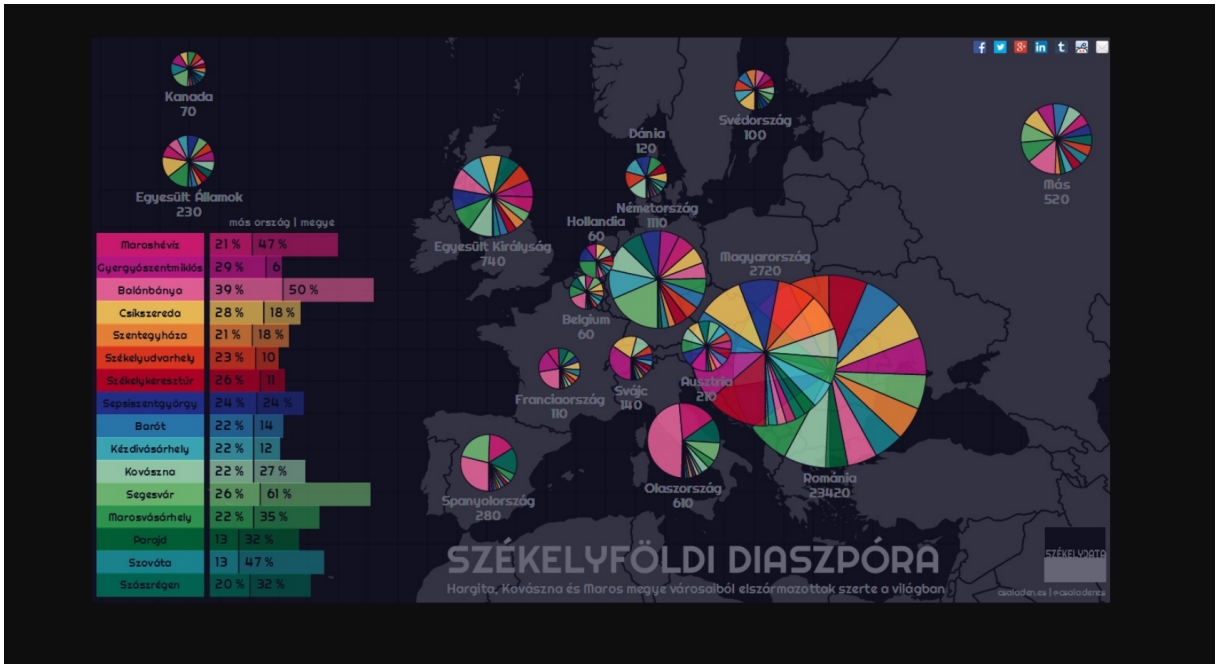
Bevezető

Napjainkban gyakori jelenség a székelyföldi régióból származó szakemberek körében, hogy ideiglenesen, vagy akár hosszú távon külföldön telepednek le, tanulnak, dolgoznak. De vajon hányan vannak, melyik országokban és milyen arányban oszlanak el ezek a szakemberek a nagyvilágban? Ezekre a kérdésekre keresi a választ Csala Dénes a SZÉKELYDATA [23] nevű projektjében, melynek célja, hogy feltérképezze a székelyföldi diaszpórát. Ebben az esetben a feltérképezésnek a módszere *social media mining* (közösségi háló alapján történő adatbányászat), konkrétan az adatok begyűjtése Facebook-bányászattal (1. ábra [1]).

Csala Dénes projektje által inspirálva született az E-migrated alkalmazás ötlete. Az E-migrated a székelyföldi IT Plus Cluster [32] által kezdeményezett Digitális Székelyföld projekt része. Tulajdonképpen a SZÉKELYDATA egy interaktív változatának és továbbgondolásának tekinthető, hiszen egy olyan webes szoftverrendszer, amely a csatlakozó szakembereket megjeleníti klaszterezett formában egy világtérképen. Lehetővé teszi a szakemberek szűrését dinamikus eszközök segítségével, foglalkozások, régiók, lakhelyek szerint. Ha valakinek megváltozik a lakhelye, módosíthatja a rendszeren belül, így mindig naprakész statisztikák állnak a felhasználók rendelkezésére. A székelyföldi diaszpóra vizuális megjelenítésén túl kollaborációs platfromot is biztosít. Létrehoz egy közösséget és funkcionalitások sokaságával ösztönöz a csatlakozásra, csatlakozás után pedig mások meghívására.

A projekt általános célja a székelyföldi és Székelyföldről elszármazott szakemberek feltérképezése és összekapcsolása, és egy olyan átfogó szakemberkataszter létrehozása, amely le-

[1] Kép forrása: http://csaladenes.egologo.ro/wp-content/uploads/2015/08/szekely_diaszpora12.jpg, utolsó megtekintés dátuma: 2018-03-07



1. ábra. A székelyföldi városok migrációs arányaihoz szükséges adatok begyűjtése Facebook-bányászattal történt.

hetővé teszi a csoport tagjai számára a szakmai tudásátadást, tapasztalatcserét. Hozzájárul egy olyan hálózat kialakulásához, melynek tagjai saját szakmájukban kiemelkedő eredményeket értek el, és szívesen hozzájárulnának a régió gazdasági és technológiai fejlődésének elősegítéséhez. Hosszabb távon akár hozzájárulhatna a székelyföldi digitális polgárság kialakulásának megalapozásához.

A dolgozat további részében bemutatásra kerül a projekt felépítése, funkcionalitásai és működése, valamint a fejlesztés során használt technológiák, eszközök és módszerek. Az 1. fejezet összefoglalja az E-migrated projekt elindulásának körülményeit, követelményspecifikációját, funkcionalitásait, néhány komponens- illetve osztálydiagramon keresztül szemlélteti a rendszer felépítését, architektúráját, végül kitér az alkalmazás adatmodelljének részleteire. A 2. és 3. fejezetek bemutatják a szerver- illetve a kliensoldal működését, valamint ezeknek a megvalósításához felhasznált technológiákat és eszközöket. A 4. fejezet részletezi a fejlesztés során használt munkamódszert, a verziókövetéshez, folyamatos integrációhoz és a kitelepítéshez használt eszközöket, valamint bemutatja a használt fejlesztői környezeteket, build eszközöket és statikus kódelemzőket. A 5. fejezet ábrázolja a különböző funkcionalitásokat felhasználói szemszögből, az utolsó fejezet pedig levonja a következtetéseket és feltár néhány továbbfejlesztési lehetőséget.

Az E-migrated a Codespring mentorprogram[2] keretei között készült. A mentorprogram egyik lényeges szakaszában, a nyári gyakorlat során látta meg a napvilágot, 2017 nyarán. Ekkor születtek meg az alkalmazás alapkövei és a funckionalitások egy része. Ezt követően a fejlesztés a Csoportos projekt [3] nevű tantárgy keretein belül folytatódott, ahol a dolgozat szerzőinek kétszemélyes csapata ideiglenesen további három taggal bővült. Egy egyetemi félév erejére a fejlesztésbe, az alkalmazás funkcionalitásainak bővítésébe Balázs Brigitta, Molnár Krisztina és Mezei Boldizsár kapcsolódtak be. A fejlesztési folyamat koordinálásáért köszönet jár a Codespring-nek, valamint dr. Simon Károly egyetemi adjunktusnak, Sulyok Csaba doktorandusznak és Szász István szoftverfejlesztőnek, akik vállalták a mentorok szerepét az alkalmazás létrehozása során, és köszönet illeti Szilágyi Zoltán szoftverfejlesztőt, aki átnézte és értékelte a frontend kódbázisát.

Az alkalmazás bemutatásra került az V. Székelyföldi IT&C és Innovációs Konferencián[4]. Köszönet illeti a konferencia szervezőit és résztvevőit, akiknek segítőkész észrevételei pozitív módon járultak hozzá az alkalmazás fejlődéséhez.

[2]Forrás: https://edu.codespring.ro/, utolsó megtekintés dátuma: 2018-02-27

[3]Forrás: https://www.cs.ubbcluj.ro/files/curricula/2016/syllabus/IM_sem5_MLM5012_hu_csatol_2016_1984.pdf, utolsó megtekintés dátuma: 2018-03-16

[4]Forrás: http://www.itpluscluster.ro/hu/node/399, utolsó megtekintés dátuma: 2018-02-27

1. Az E-migrated projektről

Az E-migrated egy olyan webes szoftverrendszer, amely székelyföldi és Székelyföldről elszármazott szakemberek számára egy szakmai kollaborációs platformot biztosít, kialakítva egy aktív, egymást szakmailag segítő közösséget. A projekt alapötletét Csala Dénes SZÉKELYDATA nevű projektje ihlette, amelynek célja, hogy felmérje a különböző székelyföldi régiókból kivándorlók számát, eloszlását országok szerint és az eredményeket megjelenítse vizuálisan, különböző térképeken és diagramokon. Ezt továbbgondolva, a székelyföldi IT Plus Cluster által kezdeményezett Digitális Székelyföld projekt részeként született az E-migrated alkalmazás.

Az E-migrated túlszárnyalja azt a célt, hogy feltérképezze a rendszerben jelen levő elszármazottakat a nagyvilágban. Az elszármazottak megjelenítését interaktívvá teszi, lehetővé téve a térképen való szűrést dinamikusan, különböző szempontok szerint. Ezen kívül kialakít egy szakmai közösséget, és lehetővé teszi számukra a kölcsönös segítségnyújtást, közös munkálkodást, egymás támogatását.

A rendszer felhasználói olyan szakemberek, akik figyelemre méltó eredményeket értek el saját szakterületükön belül. Különböző területeken tevékenykedhetnek, legyen az akár az IT szektor, vagy a gazdasági szféra, de lehetnek kutatók, cégvezetők, oktatók, vagy kreatív szakmák képviselői is, stb.

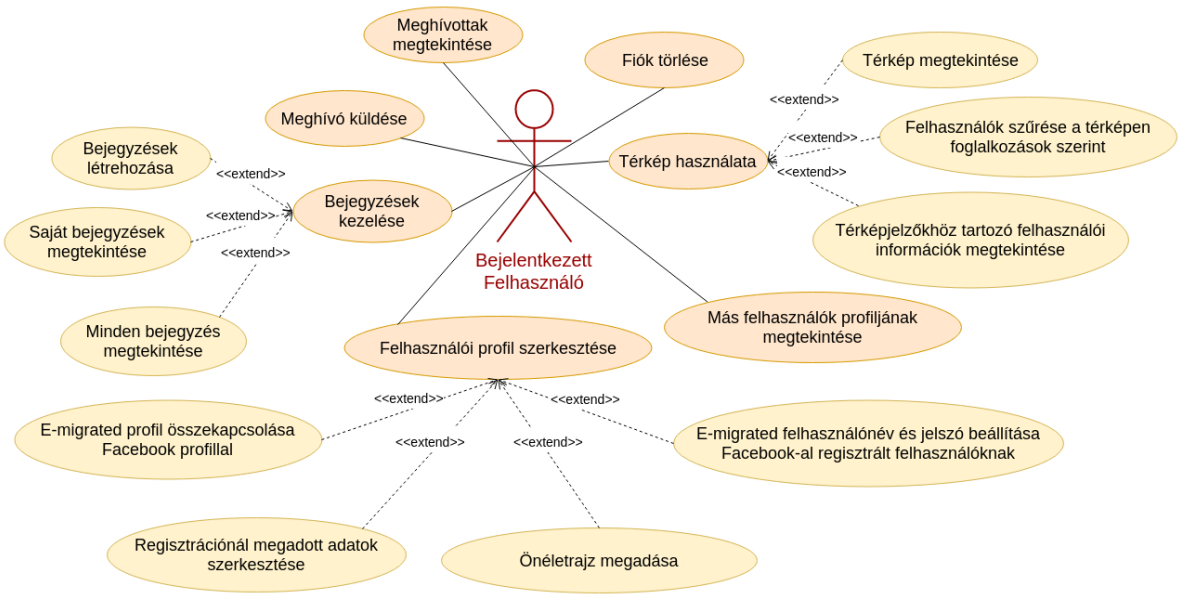
1.1. Funkcionalitások

A felhasználók csatlakozása a rendszerhez meghíváson alapszik. Egy már regisztrált felhasználó további személyeket hívhat meg, viszont a meghívók száma korlátozott, amely arra ösztönzi a felhasználókat, hogy gondolják át, hogy kik azok, akiknek szerintük valóban helye lenne ebben a közösségben.

Csatlakozáskor minden felhasználónak el kell olvasnia az E-migrated alkalmazás adatvédelemre vonatkozó szabályzatát. Ez az európai GDPR (General Data Protection Regulation) [15] elveknek megfelelően született és leírja, hogy csatlakozáskor, illetve a későbbiekben a profiladatok bármilyen módosításakor hogyan lesznek kezelve, és hol lesznek eltárolva az alkalmazáson belül megadott személyes adatok. Adott felhasználó a regisztrálás, vagy bármilyen profilmódosítást jóváhagyó gombra kattintva elismeri, hogy olvasta az alkalmazás adatvédelmi szabályzatát, és elfogadta azt. A szabályzat bármikor elérhető minden regisztrált felhasználó számára.

A felhasználói profilok a regisztrációnál megadott adatokon túl (név, foglalkozás, lakhely stb.) tartalmazzák a felhasználó rövid bemutatkozó leírását is. Ezen kívül minden felhasználó mások számára is publikus bejegyzéseket tehet közzé.

Egy bejelentkezett felhasználó kérheti a fiókja törlését, a jelszava újbóli megerősítése által. A törlés során kiválaszthatja, hogy szeretné-e törölni a közzétett bejegyzéseit is. Amennyiben szeretné törölni, minden hozzá tartozó adat törlődik az adatbázisból. Ha azonban azt az opciót



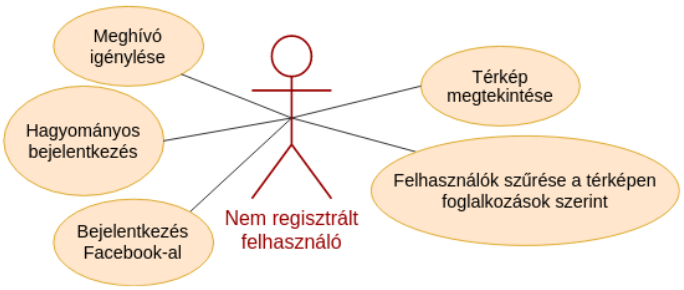
2. ábra. Felhasználó funkcionalitásai: profil szerkesztése, bejegyzések létrehozása és megtekintése, térkép megjelenítése, keresés/szűrés a térképen, más profilok megtekintése, meghívó küldése, meghívottak megtekintése, profil törlése.

választja, hogy megtartja a bejegyzéseket, hogy azok továbbra is böngészhetőek és visszakereshetőek legyenek mások számára, törlődik minden hozzá tartozó adat, kivéve a közzétett bejegyzéseit. Ezek a törlés pillanatában automatikusan egy "anonymous felhasználóhoz" rendelődnek hozzá.

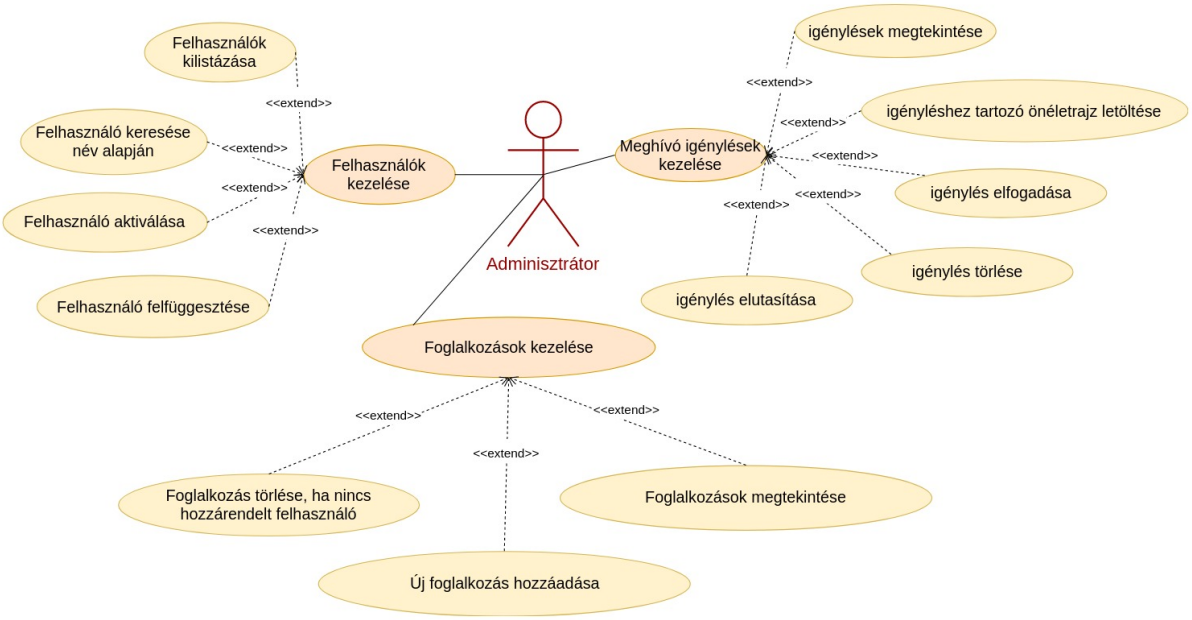
Az alkalmazás főoldalán meg van jelenítve egy világtérkép, melyen láthatóak a rendszer felhasználói lakhelyek szerint csoportosítva, és lehetőség van a felhasználók szűrésére foglalkozások szerint. A regisztrált felhasználók megtekinthetik a térképjelzőkhöz tartozó felhasználói profilokat és felvehetik egymással a kapcsolatot, a profilon megadott elérhetőségek (e-mail cím, Facebook oldal, lakcím) segítségével (2. ábra). A vendégfelhasználók (nem regisztrált felhasználók) a térképen csak a felhasználókat reprezentáló térképjelzőket láthatják, viszont nem láthatnak róluk semmilyen információt.

Ha valaki nem kapott meghívót, de csatlakozni szeretne a rendszerhez, igényelheti a felvételt. Ez esetben el kell küldenie egy önéletrajzot és egy rövid motivációs levelet, mely összefoglalja, hogy miért szeretne csatlakozni.

Meghívást követően a csatlakozás kétféleképpen történhet. Az egyik mód a hagyományos regisztráció. Ez esetben egy űrlapot kell kitölteni, megadva a standard profilhoz szükséges adatokat, mint például név, foglalkozás, lakhely, stb. A másik lehetőség a valamilyen közösségi hálón keresztül történő regisztráció. Ennek az az előnye, hogy nincs szükség űrlapok kitöltésére, gyors és egyszerű, mert a felhasználó profiljához szükséges adatokat a rendszer átveszi az adott szociális hálótól és a későbbiekben a bejelentkezés is történhet az adott szociális hálón keresztül. Az E-migrated esetében jelenleg a használt szociális háló a Facebook, de tervben van a bővítés a Google+ és LinkedIn közösségi hálókkal. (3. ábra)



3. ábra. A vendégfelhasználó funkcionalitásai: térkép megtekintése felhasználói információk nélkül, meghívó kérése, bejelentkezés.



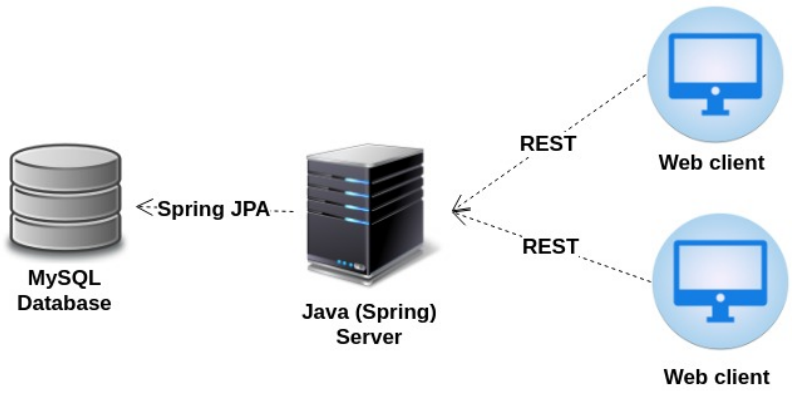
4. ábra. Adminisztrátor funkcionalitásai: meghívó igénylések elbírálása, foglalkozások szerkesztése, felhasználói profilok kezelése.

Az adminisztrátorok egyidőben felhasználói is a rendszernek, emellett azonban ha átlépnek admin módba, bővül a funkcionalitásaik listája. Ők bírálják el a beérkező meghívóigényléseket, szerkeszthetik az alkalmazásban megjelenő foglalkozások listáját. Láthatják a rendszer összes felhasználói fiókját, indokolt esetben pedig felfüggeszthetik vagy éppen újraaktiválhatják azokat. (4. ábra)

1.2. Architektúra

Az alkalmazás két fő komponensből áll: szerver és webkliens (5. ábra). A köztük történő kommunikáció a REST (Representational State Transfer) [49] konvencióknak megfelelő HTTP hívásokon keresztül valósul meg.

Az alkalmazás perzisztens adatainak tárolása MySQL adatbázisban történik, és a szerver-adatbázis kommunikáció a Spring keretrendszer által biztosított Java Persistence API (JPA) implementációval valósul meg (részletesebb információk a 2. fejezetben).

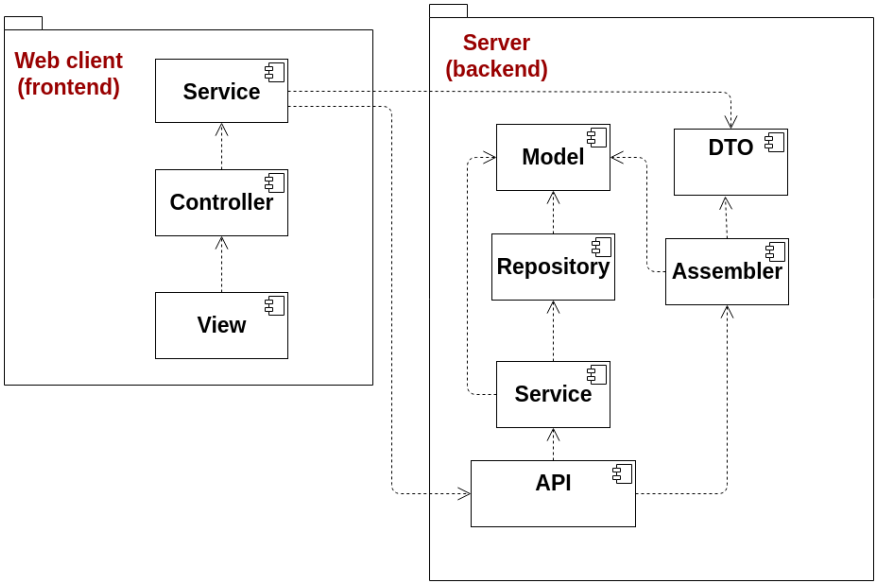


5. ábra. Az E-migrated alkalmazás kliens-szerver modellre épül. A perzisztens adatok tárolása MySQL adatbázisban történik. A szerver-adatbázis kommunikáció Spring JPA által, a szerver-webkliens kommunikáció pedig REST princípiumoknak megfelelő HTTP kérések segítségével valósul meg.

A szerveroldal modularitása [37] a komponens többrétegű architektúrájának [28] köszönhető. Előnye, hogy a különböző modulok újrahasználhatóak, könnyedén lecserélhetőek a többi réteg módosítása nélkül, a kód pedig átláthatóbbá, érthetőbbé válik és a hibakezelés is hatékonyabb. (6. ábra)

Az *adathozzáférési réteg* (data access layer) valósítja meg az alkalmazás adatainak adatbázisszintű kezelését. Ehhez a réteghez tartoznak a Model és Repository komponensek. A *Model* tartalmazza az alkalmazás alapvető entitásait. Ezek egyszerű POJO-k (Plain Old Java Object) [40], privát adattagokkal és publikus getter, illetve setter metódusokkal. Magát az adatbázissal való kommunikációt a *Repository* komponens valósítja meg, a DAO (Data Access Object) [21] tervezési minta segítségével. Olyan interfészeket biztosít, amelyek lehetővé teszik az adatbázisban található adatokhoz való hozzáférést a felsőbb rétegek számára. Így elválasztja az alkalmazás logikáját az alacsonyszintű adatbázishozzáféréstől. Ennek előnye, hogy ez a réteg egyszerűen lecserélhető, anélkül, hogy a felsőbb rétegeket módosítani kellene. A Model komponens osztályait használják az Assembler, a Repository és a Service komponensek.

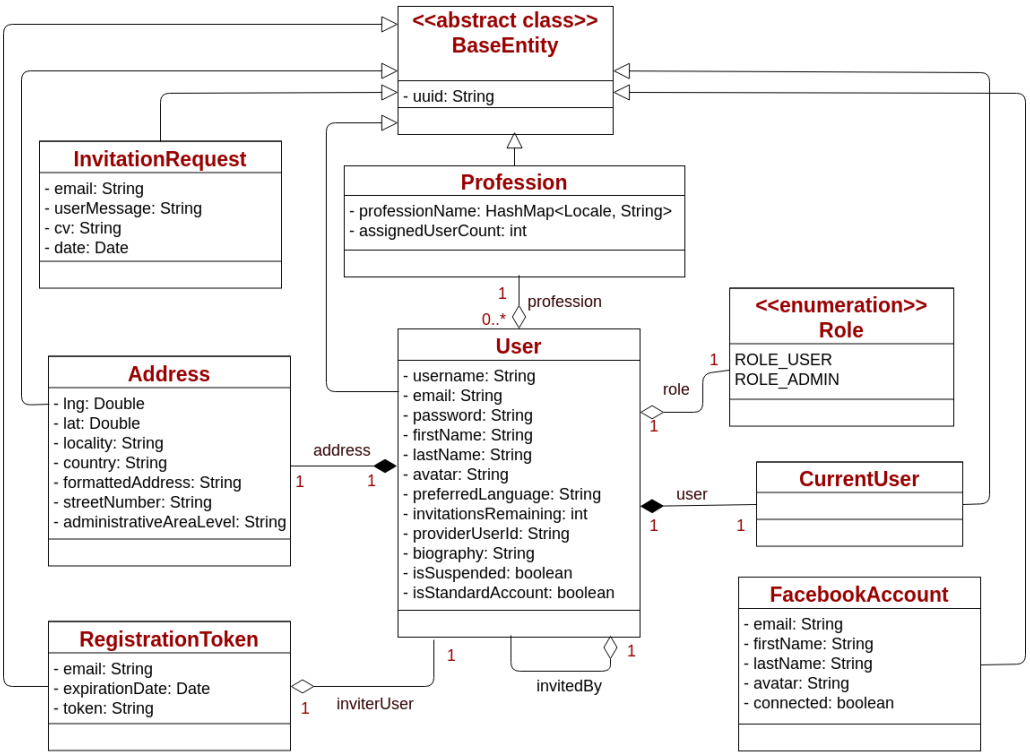
Az *üzleti logika réteg* (business logic layer) határozza meg, hogy az alkalmazáson belül hogyan lehet interakcióba lépni az adatokat jelképező objektumokkal. Köztes réteget képez az adathozzáférési réteg és a megjelenítési réteg közt. Így a megjelenítési réteg nem fér közvetlenül hozzá az alkalmazás alapvető entitásaihoz, és nem kell tudnia az adathozzáférés implementációs részleteiről sem. Egyedüli feladata, hogy szolgáltatásokat biztosítson a megjelenítési réteg számára. Ezt a réteget az E-migrated esetén a Service, az API, a DTO és az Assembler komponensek képviselik. A *Service* a "facade" tervezési mintára [55] épül és különböző interfészeken keresztül biztosítja az API számára elérhető szolgáltatásokat. Az interfészek megfelelő implementációi felhasználják a Repository komponens által biztosított metódusokat. Az *API* komponens REST erőforrásokat biztosít a megjelenítési réteg számára, tehát a REST princípiumok alapján felépített HTTP kéréseket (request) kap és válaszokat (response) küld vissza.



6. ábra. Az E-migrated alkalmazás architektúrája a web kliens és szerver komponensekből áll. A szervert az adathozzáférési réteg (Model, Repository) és az üzleti logika réteg (Service, API, Assembler, DTO) alkotja. A frontend pedig a megjelenítési réteget képviseli a View, Controller és Service komponensek által.

A megjelenítési rétegnek azonban az üzleti logika objektumok különböző tulajdonságaira vagy akár ezek kombinációjára van szüksége. Annak érdekében, hogy csökkenteni lehessen a HTTP hívásokat, a szerver-web kliens közti HTTP üzenetek szerializációja DTO-k (Data Transfer Object) [26] segítségével van megvalósítva. Egy DTO tartalmazza az összes olyan attribútumot amire szükség van egy adott nézet megjelenítése esetén, az *Assembler* komponens pedig elvégzi a szükséges átalakításokat a Model és a DTO elemek közt. [18] Például amikor egy felhasználó létrehoz egy új bejegyzést, a megjelenítési réteg egy PostDTO típusú objektumot hoz létre, amely tartalmazza a bejegyzés címét, szövegét, a szerző felhasználónevét, illetve a létrehozás dátumát. Amikor viszont a meglévő bejegyzéseket szeretné kilistázni, már szükség van egy olyan DTO-ra, amely ezek mellett tartalmazza a bejegyzés szerzőjének teljes nevét, és profilképét is, hiszen ezek az információk is megjelennek a bejegyzések fejlécében megtekintéskor.

A *megjelenítési réteg* (presentation layer) felelős a felhasználói felület viselkedéséért. Ezen belül a *Web Service* a Web Controller kéréseit szolgálja ki, a szerver API rétegével kommunikálva és felhasználva a szerver DTO elemeit. A *Controller* felel a View irányításáért, használva a Service réteget. A *View* réteg pedig az alkalmazás felhasználói felületének kinézetét biztosítja. A megjelenítési réteg kliens oldali MVC-t (Model-View-Controller) [51] alkalmaz, vagyis elkülöníti a nézetet a vezérléstől és az adatoktól. A hagyományos MVC átkerül kliensoldalra, így a szerver nem egy felépített nézetet fog visszaküldeni, amit a böngészőnek csak meg kell jelenítenie. A kliens oldali MVC esetében a szerver csak statikus erőforrásként szolgál, valamint REST erőforrásokat biztosít és elküldi a megfelelő adatokat a sablonok számára, a nézet kirajzolását viszont a böngésző végzi a kapott adatok segítségével.



7. ábra. Az E-migrated alapvető entitásai egyszerű Java Bean-ek, melyek a BaseEntity ősosztályból származnak. Az alkalmazás központi egysége a User osztály, mely a rendszer felhasználóit ábrázolja. Hozzá kötődő entitások az Address, Profession, CurrentUser, RegistrationToken és FacebookAccount osztályok, valamint a Role enum. A meghívó igényléseket pedig az InvitationRequest osztály ábrázolja.

1.3. Adatmodell

Az E-migrated alkalmazáson belüli entitások Java Beanek, amelyek egyszerű getter és setter metódusokat, valamint egy paraméter nélküli konstruktort tartalmaznak. Minden bean a `BaseEntity` ősosztály leszármazottja. A `BaseEntity` egyetlen attribútummal rendelkezik, a UUID-val, amely egy egyedi, rendszer szintű azonosító és a relációs adatbázisban az entitásoknak megfelelő táblák elsődleges kulcsa. (7. ábra)

Az E-migrated alkalmazás központi entitása a `User` osztály, melynek attribútumai közé tartozik a felhasználó neve, jelszava, email címe, profilképe, önéletrajza, az általa kiválasztott nyelv és egyébb, a felhasználóhoz tartozó információk.

Minden felhasználó rendelkezik egy lakcímmel, amely az adatbázisban egy külön táblában van eltárolva és a User oszályhoz egy az egyhez kapcsolatforma segítségével van hozzárendelve, annak ellenére, hogy lakhatnak többen is ugyanazon a helyen, viszont azért, hogy ha valaki módosítani szeretné a lakcímét, ne módosuljon más személy lakhelye is, a csapat ezt a megközelítést választotta. Az `Address` osztály fontos szerepet játszik a felhasználók clusterezett megjelenítésében a főoldalon található térképen. A `lat` és `lng` adattagjai tárolják egy felhasználó címének földrajzi hosszúság és szélesség koordinátáit, amely alapján a térképen a felhasználók csoportosítva vannak.

A felhasználók foglalkozása szintén egy külön tábla az adatbázisban, és a két entitás egy a többhöz kapcsolat segítségével van összekötve, hiszen egy szakma tartozhat több személyhez is. A `Profession` model osztály tartalmazza a hozzá tartozó felhasználók számát, illetve egy kulcs-érték párost, amelyben tárolva van a foglalkozás neve tetszőleges számú nyelven. A `Post` osztály egy felhasználó által készített bejegyzést jelképez. Tartalmazza a címét, szövegét, feltöltési dátumát illetve a felhasználót, akihez tartozik az adott bejegyzés.

Az `InvitationRequest` entitás teszi lehetővé a meghívók igénylését a közösséghez való csatlakozás érdekében. Tartalmazza a meghívót kérő személy email címét, rövid motivációs levelét, önéletrajzát illetve az igénylés dátumát.

A meghíváson alapuló regisztráció megvalósítását a `RegistrationToken` osztály biztosítja, mely tartalmazza magát a tokent, a token lejárati idejét, az email címet amelyre küldték a meghívót, illetve tárolja a felhasználót aki küldte a meghívót.

A szerepkör alapú jogosultságkezelés a `Role` enum típus segítségével valósul meg, melynek értéke lehet `ROLE_USER` és `ROLE_ADMIN`. Minden felhasználó rendelkezik egy szerepkörrel, amely regisztrációkor automatikusan a `ROLE_USER` értéket kapja.

2. Szerver oldali technológiák

Az E-migrated alkalmazás szerver oldali komponensének alapja a Java programozási nyelv és a Spring keretrendszer. Ez a fejezet leírja a szerver oldalon használt technológiákat, a különböző architektúrális rétegeket, a komponensek működését és egymással való kommunikációját.

2.1. Spring keretrendszer

A Spring egy összefüggő infrastruktúrát biztosító Java keretrendszer. Alappillérei az Inversion of Control (IoC) és a Dependency Injection (DI) technikák [27], amelyek kezelik és konfigurálják a komponenseket, menedzselik a példányokat és az ezek közötti függőségeket. Ezáltal a Spring leveszi a példányosítás és komplex konstruktormegírás terhét a fejlesztő válláról. Segítségével egy átlátható, moduláris alkalmazás hozható létre, mely klasszikus POJO-kra alapozva biztosítja az enterprise Java szolgáltatásait egy kompakt csomagolásban [46].

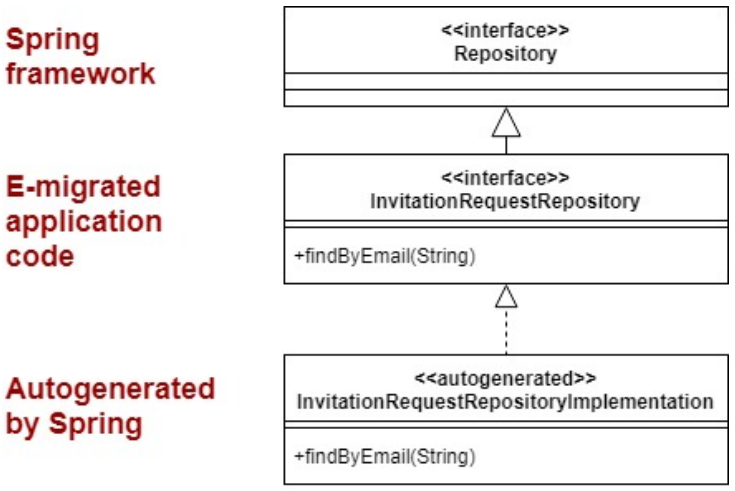
A Spring keretrendszer moduljainak fő csoportja a Core Container [44], amely többek között tartalmazza a spring-core és spring-beans modulokat, amelyek biztosítják az IoC és DI funkciókat.

Az E-migrated egy Spring-Boot alapú alkalmazás, amely egy egyszerű módszer egy kezdetleges, futtatható Spring alkalmazás felépítésére. Erősségét és népszerűségét annak köszönheti, hogy a "convention over configuration" [20] (konvenció konfigurálás fölött) elvet követve, a kezdeti beállításokat, konfigurációkat nem bízza a fejlesztőre, hanem alapértelmezett értéket ad ezeknek, ezáltal elősegítve az alkalmazás egy kezdeti, működő verziójának felépítését. Ez megkönnyíti a külső fejlesztőkkel való kollaborációt is, mert könnyen olvashatóvá teszi a kódot, és a konvenciók számukra is ismerősek lesznek. Természetesen ezek a beállítások a későbbiekben egyszerűen módosíthatóak. [43]

2.2. Adathozzáférés és adattárolás

Az E-migrated alkalmazás szervere MySQL relációs adatbázist használ az adatok tárolására, amellyel a Spring Data JPA segítségével kommunikál. A Spring keretrendszer az adathozzáférési réteg implementálására egyszerű és kényelmes megoldásokat nyújt. Olyan Data Access Object (DAO) támogatást biztosít, amely által a különböző perziszenciát biztosító technológiák egységesen és könnyen kezelhetőek. Biztosít egy absztrakciós szintet a JDBC (Java Database Connectivity) API-hoz, így megkíméli a fejlesztőket a bonyolult, adatbázis specifikus lekérdezések írásától, az alacsonyszíntű részletek implementálásától. Ezen felül biztosít egy egységes hibakezelési módszert, így nem kell külön ismerni a különböző adatbázis specifikus hibákat.

A `@Repository` annotáció gondoskodik arról, hogy a komponens szkennelő megtalálja és konfigurálja az adott adathozzáférést biztosító osztályt, így nem szükséges XML állomány a konfigurációk megadására. Az E-migrated alkalmazás esetében is az annotációs megoldás használatos.



8. ábra. A Spring Data keretrendszer leveszi a fejlesztő válláról a DAO osztályok implementációját, elég kiterjeszteni a keretrendszer által biztosított interfészeket és a konvenciókat betartva deklarálni a metódusokat, a Spring a metódus nevéből automatikusan ki tudja generálni az implementációt.

A Spring keretrendszer támogatja a Java Persistence API (JPA) integrációját, amely egy absztrakciós szintet képez a JDBC felett és segítségével szabványosítható az Object Relational Mapping (ORM). Az ORM célja, hogy megkönnyítse az objektum orientált programozási nyelvek és relációs adatbázisok közötti leképezést [22].

Minden Data Access Object-nek vagy adathozzáférést implementáló osztálynak szüksége van egy erőforrás hozzáférést biztosító objektumra, amely a `@PersistenceContext` annotáció segítségével injektálható be, és amely a META-INF mappában található *persistence.xml* állományban levő konfigurációkkal szabható testre.

A Spring Data repository absztrakció fő interfésze a `Repository`. A `CrudRepository` ennek leszármazottja, és lehetővé teszi a CRUD (Create, Read, Update, Delete) műveletek egyszerű megadását. Nem szükséges implementálni a DAO osztályokat, elegendő csupán kiterjeszteni a Spring Data által biztosított interfészeket, és deklarálni a számunkra szükséges metódusokat, a keretrendszer dinamikusan fel tudja építeni a megfelelő adaterélési parancsot [45]. Például az `InvitationRequestRepository` interfészben az `findByEmail` metódus, mely visszatéríti a paraméterként megadott e-mail címhez tartozó meghívó kérést, csak ki van jelentve, nincs implementálva, az implementációt a keretrendszer generálja, a betartott konvenciók alapján (8. ábra). Lehetőség nyílik bonyolultabb lekérdezések egyszerű megadására is `NamedQuery`-k segítségével. A lekérdezéshez szükséges model osztály definíciója előtt a `@NamedQuery` annotáció segítségével felépíthető a lekérdezés, akár parametrizálva is. A lekérdezés törzsszövegében használt nyelvezet a Java Persistence Query Language (JPQL), amelynek segítségével az entity modelre épített adatbázis lekérdezések írhatóak. A JPQL szintaxisa hasonlít az SQL nyelvezetére, de a Java környezetben levő entity osztályneveket használja a lekérdezések felépítésére, nem pedig az adatbázis oldalon levő tábla elnevezéseket.

Példa erre többek között a `UserRepository` interfészben, az 1. kódrészletben található

`findUsersLikeSearch()` metódus, amely a `search` paraméter által megadott szöveg alapján visszatéríti azokat a felhasználókat, akiknek a nevében megtalálható az adott szövegrész. Ez a metódus a többihez hasonlatosan nincsen implementálva, az adatbázis szintű lekérdezést a keretrendszer építi fel, a `User` model osztálydefiníciója előtt található `@NamedQuery` annotáció segítségével, amely a 2. kódrészletben látható. A `@Param` annotáció köti össze a `@NamedQuery`-ben szereplő `:search` változót a metódus paraméterlistájában szereplő `search` paraméter értékével. Ennek segítségével bármilyen bonyolult lekérdezés egyszerűen megírható.

A Spring Data `PagingAndSortingRepository` interfésze olyan esetekben használható amikor nem szükséges, illetve nem kívánatos egyszerre az összes adatbázisban levő objektumot visszatéríteni, csupán azoknak egy részét, például az 10. objektumtól a 15. objektumig. A lapozás megvalósításához elegendő a `findAll()` metódus paraméterének átadni egy `PageRequest` objektumot, amelynek be van állítva, hogy hányadik oldalt térítse vissza, illetve, hogy egy oldalon hány objektum szerepeljen. Az almalmazáson belül a bejegyzések lapozása van megoldva a `PagingAndSortingRepository` segítségével.

Az E-migrated adathozzáférési rétegében minden az 1.3. fejezetben említett model osztály számára létezik egy neki megfelelő `Repository` interfész. Ezeken kívül még megtalálható a `UserConnectionRepository`, ami nem leszármazottja a `Repository` ősinterfésznek, natív implementációt tartalmaz, amire azért volt szükség, hogy a Facebook integráció esetében is biztosítani lehessen a meghíváson alapuló regisztrációt, mivel a Spring Social (lásd 2.6. fejezet) akkor is létrehoz egy bemenetet a `UserConnection` táblába, ha valaki illegálisan próbált bejelentkezni. Ezt a rekordot pedig törölni kell, hogy ne kapjon hibát az a személy, aki legálisan, meghívó által próbál regisztrálni az adott Facebook profillal, vagy épp hozzá akarja kötni ezt a Facebook profilt az E-migrated fiókjához.

2.3. Üzleti logika réteg

Az E-migrated alkalmazáson belül minden `Repository` osztályhoz tartozik egy `@Service` annotációval ellátott Service Spring Bean, amely injektálja a neki megfelelő `Repository` interfészt, ezáltal hivatkozhat annak metódusaira. Az alkalmazás szolgáltatási rétege a cserélhetőség érdekében egy interfészt publikál az API réteg számára, és ezeket az interfészeket implementálják a `ServiceImpl` osztályok. Ezáltal az API réteg egyszerűen injektálhatja ezeket az inter-

```
@Repository
public interface UserRepository extends CrudRepository<User, Long> {
        Iterable<User> findUsersLikeSearch(@Param("search") String search)
                    throws RepositoryException;
        ...
}
```

1. kódrészlet. Bonyolult lekérdezés megvalósítása a Spring Data keretrendszer és NamedQuery-k segítségével.

```
@Entity
@Table(name = "user")
@NamedQueries({
        @NamedQuery(name = "User.findUsersLikeSearch",
        query = "SELECT u FROM User u WHERE CONCAT(first_name, ' ', last_name)
                LIKE :search ") })
public class User extends BaseEntity {
        ...
}
```

2. kódrészlet. NamedQuery megadása a User bean osztálydefiníciója előtt, JPQL segítségével.

fészeket, nem kell tudnia a háttérben levő implementációkról.

A szolgáltatási réteg naplózza, burkolja és továbbítja a kivételeket a felsőbb rétegeknek. Minden szolgáltatási rétegbeli metódus hiba esetén dob egy `ServiceException`-t amit az API réteg kap el és kezel le.

A `UserService` osztály kezeli a felhasználókkal kapcsolatos műveleteket, mint például a regisztráció, bejelentkezés, jelszó és felhasználónév módosítás, Facebook által regisztrált felhasználók E-migrated fiókjának létrehozása, valamint visszatéríti a felhasználókat a bejövő paraméterek alapján.

A `FacebookConnectionService` osztály hozzájárul a Facebook-kal való regisztráláshoz, illetve egy hagyományosan regisztrált felhasználó E-migrated fiókjának a Facebook profiljával való összekötéséhez (további információ a 2.6 fejezetben).

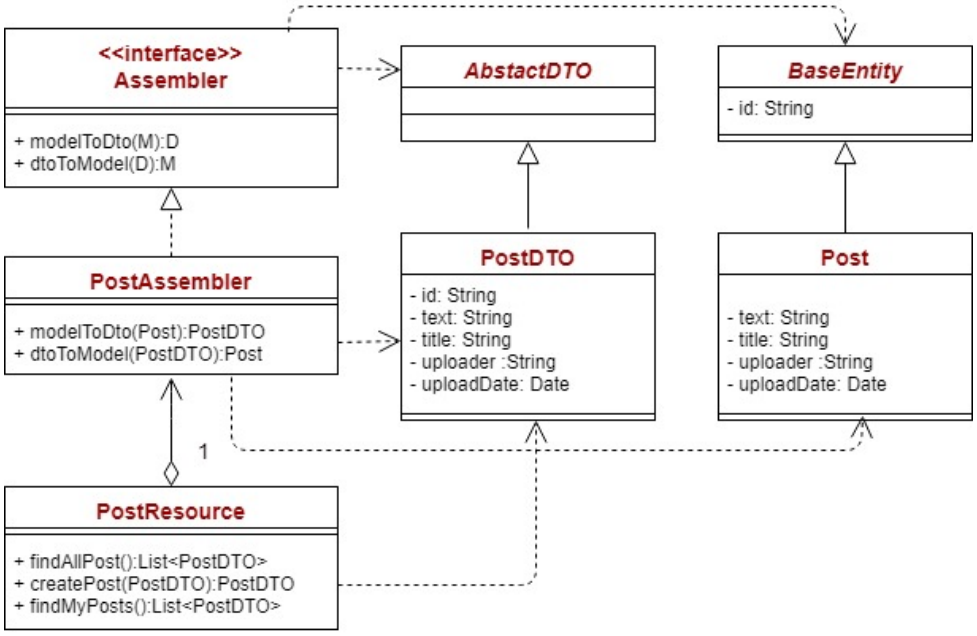
2.4. API réteg

Mivel az API réteg felelős a klienssel való kommunikációért és ezt egy RESTful alkalmazás segítségével teszi meg, kézenfekvő volt használni a Spring által biztosított spring-web csomagot, amely segítségével egyszerűen implementálható egy RESTful alkalmazás.

2.4.1. Data Transfer Object

A kliens és a szerver közötti adatmegosztás Data Transfer Objectek segítségével valósul meg. A DTO-k egyszerű POJO-k, melyek az attribútumaikat beállító és lekérdező getter és setter metódusokat tartalmaznak. A DTO tervezési mintának [26] megfelelően minden Transfer Objectnek van egy megfelelő `Assembler` interfész, amelyek egy közös Assembler interfészből származnak, tehát kötelezően tartalmaznak legalább két metódust (lásd 9. ábra). A metódusok deklarációjában levő M és D típusnevek egy-egy generikus típust jelölnek, melyet a specifikus `Assembler` interfészek határoznak meg attól függően, hogy melyik DTO-hoz tartoznak. Az említett metódusok közül az első egy adott `Model` objektumot alakít át egy neki megfelelő `DTO`-ra, a második pedig fordítva.

Azokban az esetekben, ahol szükség volt a klienstől érkező adatok alapján egy `Model` objektum felépítésére és különböző alapértelmezett értékek beállítására az `Assembler` osztályok



9. ábra. Az E-migrated alkalmazás a kliens és a szerver közötti adatmegosztásra DTO-kat használ. A DTO tervezési mintát alkalmazva minden Assembler egy ős Assemblerből származik, és minden Model, illetve DTO rendelkezik egy neki megfelelő Assemblerrel, mely az adott Modelt alakítja át DTO-vá illetve fordítva. A Resource osztályok ezeket az Assemblereket használva dolgozzák fel a beérkező adatatokat.

tartalmaznak egy `M create(D dto)` metódust is, amelyek visszatérítenek egy adott `Model` objektumot. Ilyen például a `RegistrationUserAssembler` is, ahol egy újonnan regisztrált felhasználónak be kell állítani, hogy kezdetben hány darab meghívóval rendelkezik, valamint az alapértelmezett nyelvet (magyar). Ezeket az alapértelmezett értékeket az `Assembler` osztály properties állományokból tölti be, hogy egyszerűen, a kód megértése nélkül lehessen változtatni az értéküket. A `@PropertySource("classpath:application.properties")` annotáció segítségével adható meg, hogy az osztály melyik állományból olvassa az értékeket, illetve az attribútumok deklarálásánál a `@Value("$user.defaultNumberOfInvitations")` annotáció határozza meg, hogy az adott állományból mely mező értékét vegye fel az annotált attribútum.

2.5. Biztonsági megoldások

A Spring Security összefogó biztonsági megoldásokat kínál vállalati Java alkalmazások számára. A két fő terület amit a Spring megcéloz a hitelesítés (authentication) és jogosultságellenőrzés (authorization) [47].

A Spring Security testreszabása az alkalmazáson belül a `SecurityConfig` osztályban található. Itt történik a felhasználók jelszavainak sózásához és hash-eléséhez szükséges kódolási algoritmus beállítása, amely ebben az esetben a Spring Security által biztosított, napjaink technológiáival visszafejthetetlennek vélt algoritmus, a `BCryptPasswordEncoder`. A Spring Security megkapja a felhasználó által beírt hitelesítési adatokat, ezeket kódolja, majd összehasonlítja

az adatbázisban található hash-elt jelszóval, amennyiben a két jelszó megegyezik a hitelesítés sikeresen megtörtént, különben kivételt dob.

A rendszeren belül használt `User` model osztály össze van kötve a Spring Security `User` osztályával, amely rendelkezik a felhasználónéven és jelszón kívül más attribútumokkal is, melyek további biztonsági ellenőrzéseket tesznek lehetővé, ilyen például az `accountNonLocked` adattagja. Az E-migrated alkalmazáson belül, ennek segítségével van megoldva, hogy a bejelentkezés sikertelen legyen, amennyiben a felhasználó fiókja fel van függesztve.

A klienstől érkező kérések jogosultságának elbírálását is a `SecurityConfig` osztályban levő beállítások határozzák meg. Itt van konfigurálva, hogy mi történjen, ha valaki nem megfelelő jogosultsággal próbál elérni egy erőforrást, illetve beállítható, hogy a különböző útvonalak, milyen szerepkörrel rendelkező felhasználók számára legyenek elérhetőek. Ebben a konfigurációs bean-ben vannak beállítva a sikeres, illetve sikertelen bejelentkezést kezelő útvonalak is.

Az E-migrated esetében az összes `/guest/**` formájú URL-re érkező kérés mindenkinek elérhető, viszont a `/user/**` és `/admin/**` URL-ek alatt levő erőforrásokat csak a megfelelő szerepkörrel rendelkező felhasználók érthetik el. Ugyanez a konvenció vonatkozik az `/api`-val kezdődő útvonalakra is.

2.6. Szociális hálók integrálása

Az E-migrated alkalmazásba való regisztráció lehetséges Facebook segítségével, de akár a későbbiekben is hozzákötheti egy felhasználó a Facebook profilját a már meglévő fiókjához. Ennek előnye, hogy a regisztráció, illetve a bejelentkezés egyszerűbbé válik.

A Spring Social [48] célja megkönnyíteni a fejlesztők számára a szociális hálók integrálását a Spring keretrendszerre épülő alkalmazásokba. A Spring Social egy adott felhasználó személyében küld kéréseket a szolgáltató API-jához. A bejelentkezés során a felhasználó a szolgáltató bejelentkezési oldalára lesz átirányítva ahonnan kap egy access tokent. Ez az access token regisztrációkor elmentődik az adatbázisba, tehát az E-migrated szervere ellenőrizni tudja, hogy regisztrált felhasználóról van-e szó. Amennyiben igen, visszatérít neki egy általa generált tokent (az E-migrated szerver esetében a felhasználó nevéből és Facebook ID-jából álló stringet) és ezt követően a felhasználó használhatja az E-migrated szolgáltatásait.

Azért, hogy az alkalmazásba való regisztráció Facebook-os regisztráció esetében is meghívóhoz legyen kötve, szükséges volt kihasználni a Spring Social rugalmasságát és konfigurálhatóságát. Ennek érdekében a regisztráció során a `/signin/szolgáltató` útvonalra küldött POST kérés különböző rejtett mezőket tartalmaz. Ezekben a rejtett mezőkben kapja meg a szerver többek között a regisztrációs tokent, az email címet amelyre a meghívó volt küldve, illetve annak a személynek az azonosítóját, aki küldte a meghívót. Ezek a rejtett mezők a továbbiakban szesszió hatókörű kérés paraméterekként lesznek továbbítva egészen addig, amíg a szolgáltatótól kapott válasz meg nem érkezik. A válasz érkezése után, a rendszer a kapott válasz és a paraméterek alapján eldönti, hogy sikeres a belépés vagy a regisztráció, vagy nem, illetve ellen-

```
private void buildAndSendMail(String email, String writer)
                throws MessagingException {
        MimeMessage message = sender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(message, true);
        helper.setTo(new InternetAddress(email));
        helper.setSubject("emigrated App invitation");
        helper.setText(writer, true);
        sender.send(message);
}
```

3. kódrészlet. E-mail küldése a JavaMailSender interfész segítségével.

őrzi, hogy fel van-e függesztve a felhasználó fiókja.

A Spring Social konfigurálása a `SocialConfig` konfigurációs bean-ben történik a csapat által írt adapterek és interceptorok beállításával.

2.7. E-mail küldő mechanizmus

Az E-migrated alkalmazás használata során több alkalommal is szükség van e-mail küldésére: meghívó küldése, értesítők kiküldése a profil felfüggesztése illettve újraaktiválása után.

Az e-mail küldést az E-migrated a Mailgun e-mailküldő szerver segítségével oldja meg. A Mailgun teljeskörű szolgáltatást nyújt e-mailek küldésére és fogadására egy HTTP API segítségével, ami a háttérben SMTP, POP3 és IMAP protokollokkal dolgozik [10].

A `MailSender` a Spring keretrendszer által biztosított fő interfész e-mailek kezelésére. Nagyon sok e-mailküldő rendszer esetében használható egyszerűsége miatt [36].

A `JavaMailSender` interfész a `MailSender` leszármazottja és ez már lehetőséget nyújt MIME típusú üzenetek küldésére is, általában a `MimeMessageHelper`-rel együtt használják, amely támogatja különböző csatolmányok (képek, dokumentumok, HTML oldalak) hozzáfűzését a levélhez [31]. A referencia-implementációja a `JavaMailSenderImpl` osztály.

Az üzenet felépítése és küldése (ld. 3. kódrészlet) során egy `JavaMailSender` objektum `createMimeMessage()` metódusa által visszatérített `MimeMessage` megkapja az e-mail elküldéséhez szükséges adatokat, majd a `send()` metódus elküldi az üzenetet.

Az E-migrated alkalmazás esetében a különböző e-mailek felépítése és elküldése, a `util` modulon belül levő `MailSenderBean` osztályban történik. Az e-mail küldéshez szükséges konfigurációs beállítások pedig a `conf` modulon belül a `MailSender` konfigurációs osztályban találhatóak meg. Ezeket az értékeket a rendszer egy `mail.properties` állományból olvassa be a 2.4.1 fejezetben említett módon. Ebben az állományban található meg többek között az e-mailszerver elérési útvonala és azon belül az alkalmazásnak lefoglalt felhasználónév és jelszó.

A dinamikusan felépített e-mailek renderelése Freemarker templatek alapján történik a Spring Web MVC (Model View Controller) segítségével.

3. Kliens oldali technológiák

Az E-migrated alkalmazás webes komponense egy single-page application (egyoldalas alkalmazás) [42], melyben az `index.html` oldal dinamikus felépítése és a különöző nézetek kezelése az Angular.js nyelv segítségével valósul meg. Az oldalak megjelenítésében és a responsive design megvalósításában a Bootstrap keretrendszer [1] játszik jelentős szerepet. A kommunikáció a webes komponens és a szerver közt a REST API segítségével történik. A fejezet során bemutatásra kerülnek a kliens oldal megvalósításához használt technológiák, a web kliens struktúrája, komponensei és ezek működése.

A webes komponens struktúra szempontjából a következő modulokra bontható: *controllers*, *services*, *directives* és *config*. A kontrollerek szinkronizálják az alkalmazás kinézetét a megfelelő adatokkal, a service-ek kezelik a szerver fele történő aszinkron API hívásokat, a direktívák lehetővé teszik a DOM (Domain Model Object) elemek viselkedésének a módosítását, illetve kiterjesztését, míg a config modul a különböző konfigurációs beállítások megadásának a helye.

3.1. Angular.js

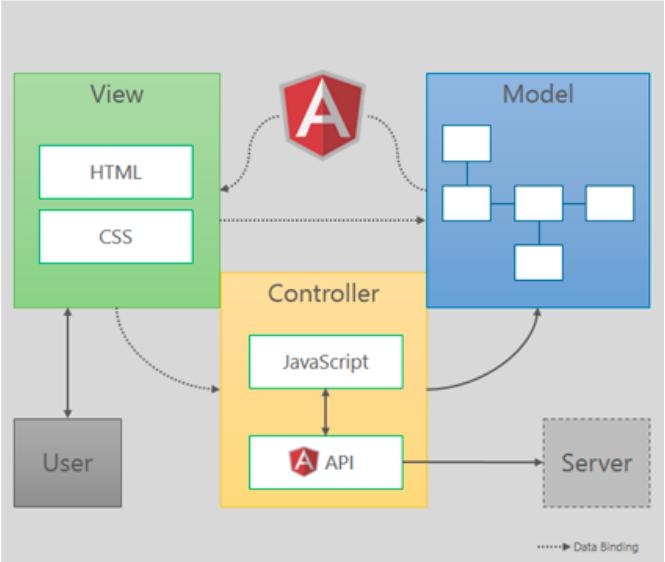
Az Angular.js egy JavaScript alapú keretrendszer webalkalmazások készítésére, amely lehetővé teszi a HTML "szótárának" kiterjesztését speciális jelölésekkel, úgynevezett direktívákkal [33]. Jelentősebb előnyei, hogy támogatja a kliens oldali MVC-t, a kétirányú adatkötést (two way data-binding), valamint a dependency injection-t.

Az MVC elv lehetővé teszi az adatok kezelésének, a nézetek megjelenítésének és az alkalmazás logikájának elkülönítését, így megkönnyítve az alkalmazás bővítését, karbantartását és tesztelését. A nézet a modelltől kapja az adatokat és megjeleníti a felhasználó számára. A modell elemei egyszerű JavaScript objektumok. Ha a felhasználó interakcióba lép a felülettel a kontroller felelős azért, hogy megfelelően módosítsa a modellt. Ezt egy `$scope`-nak nevezett osztott objektumon keresztül teheti meg. Végül pedig ha a modell megváltozik, értesíti a nézetet, amely frissíti a felhasználói felületet. (10. ábra)

Az Angular.js alkalmazások a kliens oldali MVC-t támogatják, vagyis a szerver szerepe csupán, hogy statikus erőforrásként szolgáljon, valamint REST erőforrásokat biztosítson a sablonok (template) számára, és elküldje a megfelelő adatokat, amelyekre a sablonoknak szükségük van. A sablonok és a megfelelő adatok alapján pedig a kliens oldal, jelen esetben a böngésző fogja renderelni a nézetet.

A kétirányú adatkötés biztosítja a felhasználói felület DOM elemeinek és a modell elemeinek az automatikus szinkronizációját, megkímélve a programozókat a nehézkes, manuális frissítgetéstől, és jelentősen lerövidítve az erre vonatkozó kód mennyiségét. Így csak arra van szükség, hogy a programozók deklarálják, hogy a felhasználói felület mely része milyen JavaScript tulajdonságokhoz van térképezve.

A dependency injection megengedi a külső függőségekre való hivatkozást Angular kom-



10. ábra. [5]Az MVC elv lehetővé teszi az adatok kezelésének, a nézetek megjelenítésének és az alkalmazás logikájának elkülönítését. Amikor a felhasználó hat a nézetre, a háttérben megváltoznak a modell elemek, és meghívódnak a megfelelő kontrollerek (*adatkötés*). Amikor viszont a kontrollerek kommunikálnak a szerverrel, és megváltoztatják a modell elemeket, az Angular.js érzékeli ezeket a változásokat és értesíti a nézetet, hogy az megfelelően frissítse a felhasználói felületet (*kétirányú adatkötés*).

ponensekből (pl. controller, service), anélkül hogy szükség volna az adott függőségek implementációs részleteinek ismeretére. A függőség felépítése sem hárul a programozóra, mert automatikusan injektálódik. Ezáltal a minta megkönnyíti a fejlesztést és átláthatóbbá, könnyebben tesztelhetővé teszi a kódot. Ugyanakkor lehetőség van új függőségek megírására és elérhetővé tételére is.

3.1.1. Fontosabb direktívák

Direktívák segítségével új HTML tag-ek adhatóak meg, melyek új funkcionalitásokat valósítanak meg, és különböző módon manipulálják a DOM elemeit. Az Angular.js számos beépített direktívával rendelkezik, melyek segítségével megadható az alkalmazás nézetének a sablonja. Ilyenek például az `ng-app`, `ng-controller`, `ng-model`, stb. Az `ng-app` direktíva befolyásolja, hogy az oldal mely részeit szeretnénk kezelni. Az `ng-controller` megadja, hogy az oldal adott részét melyik kontroller vezérli. Az `ng-model` pedig az adatkötés megvalósulását teszi lehetővé. [30]

3.1.2. Fontosabb külső függvénykönyvtárak

Ezeken kívül léteznek olyan third-party függvénykönyvtárak, amelyek szintén előre megírt hasznos funkcionalitásokat hoznak magukkal, és integrálásuk az Angular.js-ben megírt webalkalmazásba egyszerű. Az E-migrated alkalmazásban a használt third-party függőségek közül

[5]Kép forrása: `https://devjev.files.wordpress.com/2014/07/ajs.png`, utolsó megtekintés dátuma: 2018-04-17

jelentős szerepe van az Angular Google Maps, az Angular Translate, a Sweetalert2 és az Angular UI Router könyvtáraknak.

Az Angular Google Maps teszi lehetővé az alkalmazás főoldalát betöltő lényeges komponens, a szakembereket csoportosító világtérkép megjelenítését. Ez a könyvtár olyan direktívák összessége, amelyek lehetővé teszik a Google Maps beintegrálását Angular.js alkalmazásokba, anélkül, hogy ismerni kellene a Google Maps API részleteit, és olyan jól ismert Google Maps objektumokat biztosítanak, mint például markerek, ablakok, vonalak a térképen stb. [12]

Az Angular Translate könyvtár lehetővé teszi az alkalmazás kliens oldalon történő nemzetköziesítését (i18n). Lazy loading mechanizmusának köszönhetően csak akkor kéri le a nyelv specifikus adatokat a szervertől amikor szükség van rájuk. Használata egyszerű a beépített direktíváknak és filtereknek köszönhetően. Csak arra van szükség, hogy az alkalmazás fő moduljában a megfelelő könyvtár injektálva legyen és minden nyelv esetén létre kell hozni egy JSON formátumú fájlt.[39]

A Sweetalert2 függvénykönyvtár a felugró ablakok, üzenetek szép, responsive megjelenítésében játszik szerepet. Használata egyszerű, és lehetővé teszi a megjelenő tartalom személyre szabását.

3.1.3. Angular UI-Router

Az Angular-UI Router egy kliens oldali keretrendszer, amelyet Angular.js-ben írt single-page alkalmazások számára fejlesztettek ki [13]. A UI-Router rendelkezik az Angular.js beépített `ng-router`-ének minden funkcionalitásával, viszont van két nagy előnye: a többszörös nézet, illetve a beépített nézet.

A legtöbb webes alkalmazás rendelkezik fejéccel, fő tartalommal, lábjegyzettel, illetve bizonyos esetekben egy oldalmenüvel. Az Angular.js alapértelmezett router-jével szemben, a UI-Router megengedi több nézet elhelyezését egy HTML oldalon belül, melyek közül mindegyik külön névvel, illetve külön kontrollerrel rendelkezhet. Ennek előnye, hogy ezek a nézetek újra felhasználhatóak, az alkalmazás átláthatóbb és könnyebben bővíthető.

Beépített nézet használható olyankor, amikor egymáshoz szorosan kapcsolodó nézeteket szeretnénk ugyanazon oldalon megjeleníteni. Például bal oldalon megjelenik egy felsorolás és ezen felsorolás elemeire kattintva, jobb oldalon megjelenik egy űrlap, ahol szerkeszthetjük, módosíthatjuk az elemeket (17a. ábra).

Az E-migrated alkalmázás kliens oldalán az `index.html` tartalmaz két `ui-view` direktívát, az első a fejlécben található menü, a második pedig maga a fő tartalom. A fő tartalom `ui-view`-jának neve `mainContainer` és ez lesz majd helyettesítve dinamikusan, a UI-Router által, a különböző állapotokkal.

Az *állapot* és *nézet* közti megfeleltetés az `app.conf.js` állományban történik, ahol a `$stateProvider state()` függvényének segítségével, minden állapothoz hozzárendelődik egy template URL, egy effektív URL és bizonyos esetekben egy kontroller is, valamint itt

```
$urlRouterProvider.otherwise('/home');

$stateProvider.state('home', {
    url: '/home',
    templateUrl: 'google-map.html',
    controller: 'GoogleMapController as googleMapController',
    data: {
        permissions: {
            only: ["ROLE_USER", "ROLE_ADMIN", "unauthorized"]
        }
    }
});
```

4. kódrészlet. A fő tartalmat megjelenítő ui-view-ba a külöböző állapotok dinamikus behelyettesítése a UI-Router segítségével történik. Az állapot-nézet megfeleltetések az `app.conf.js` állományban a `$stateProvider.state()` függvény segítségével valósulnak meg. Például a `'home'` állapothoz hozzárendeli a `'/home'` URL-t, és a `'google-map.html'` sablont, hozzárendeli a `GoogleMapController`-t és beállítja a megfelelő jogosultságokat. Amennyiben egy felhasználó nem rendelkezik a megfelelő jogosultsággal a `$urlRouterProvider.otherwise('/home')` függvény aktiválódik és vissza lesz térítve a főoldalra.

```
<ul ng-controller="LanguageSwitchController">
    <li><a ng-click="changeLanguage('en')">
            <span translate="BUTTON_LANG_EN"></span>
    </a></li>
    <li><a ng-click="changeLanguage('hu')">
            <span translate="BUTTON_LANG_HU"></span>
    </a></li>
</ul>
```

5. kódrészlet. Ha a felhasználó meg szeretné változtatni az oldal nyelvét, és rákattint valamelyik nyelvre, akkor a `menu.html`-ben meghívódik az `ng-click` tag által megadott, `LanguageSwitchController` kontrollerhez tartozó `changeLanguage()` függvény.

van beállítva, hogy melyik nézetet milyen jogosultsággal rendelkező felhasználó jeleníthet meg (4. kódrészlet). Amennyiben egy felhasználó nem rendelkezik a megfelelő jogosultsággal az `$urlRouterProvider .otherwise('/home')` függvény aktiválódik és vissza lesz térítve a főoldalra.

Az alkalmazásban beépített nézeteket alkalmaz a profil szerkeztése nézet (17a. ábra), valamint az adminisztrátor oldala is (18a. ábra). A felhasználói profil szerkesztése esetében van egy fő view, az `edit-profile.html` amelyhez az `editProfile` státusz van hozzárendelve és amely tartalmaz egy baloldali menüsort, illetve a menü elemeinek szerkesztésére alkalmas nézetet. Ebben a `ui-view` direktívában lesznek megjelenítve az `editProfile` alstátuszai pl. az `editProfile.changeGeneralData` állapot. A leszármazott állapotok rendelkezhetnek külön kontrollerrel, de öröklik az ősállapot kontrollerét, így használhatják annak függvényeit is.

```
function getProfessions() {
    ...
    var req = {
        method: 'GET',
        url: '/api/professions',
        contentType: "application/json"
    };
    $http(req).then(...);
}
```

6. kódrészlet. Az editProfessionService getProfessions() metódusa aszinkron kérést küld a szervernek. Felépít egy GET kérést, és elküldi azt a $http objektumon keresztül a szerver /api/professions endpoint-jára, majd a szervertől érkező válasz függvényében, az aszinkron hívás befejeződése után megfelelő promise üzenetet küld a kontrollernek.

3.1.4. Web Controller

A Web Controller az alkalmazás frontend-jének egyik lényeges komponense, amely a nézetek vezérléséért felelős az alkalmazás felhasználói felületén, és a kliens oldal üzleti logika rétegét képezi. Minden nézethez tartozik egy vagy több megfelelő kontroller osztály, amelyeket a HTML oldalakon az `ng-controller` tag segítségével, vagy az app.config konfigurációs állományban adhatunk meg.

Angular.js-ben minden kontrollernek két fő feladata van: a modell inicializálása és viselkedések megadása. Amikor létrejön egy kontroller és hozzárendelődik a DOM-hoz, akkor létrejön számára egy gyerek `$scope` objektum. Ezen az objektumon keresztül a kontroller inicializálni, majd módosítani tudja a modell tulajdonságait. Szintén ez teszi lehetővé azt is, hogy a kontroller különböző viselkedéseket definiálhasson, azáltal, hogy metódusokat ad hozzá a `$scope` objektumhoz, melyekre majd a különböző események kezelésénél hivatkozhatunk[54]. Például az oldal nyelvének a változtatása esetén ha a felhasználó valamelyik nyelvre kattint, akkor a *menu.html*-ben meghívódik az `ng-click` tag által megadott, *LanguageSwitchController* kontrollerhez tartozó *changeLanguage()* függvény. Ez látható az 5. kódrészletben is.

A kontroller komponensek tehát részt vesznek a kétirányú adatkötés mechanizmusban, különböző eseményekhez rendelhető kezelő függvényeket biztosítanak és kommunikálnak a Web Service réteggel. Mivel az Angular.js-ben minden kérés aszinkron kérés, így a Web Service és Web Controller közti kommunikáció is aszinkron kérések által valósul meg.

3.1.5. Web Service

Az E-migrated alkalmazás kliens oldali komponensében a szervertől lekérdezett adatokat a Web Service rétegben tároljuk. Ez a réteg szolgálja ki a Web Controller kéréseit, továbbítva azokat a megfelelő formában a szerver Api rétegének.

A service komponensek tehát olyan metódusokat tartalmaznak, amelyek a különböző kontrollerek számára elérhetőek, és amelyek aszinkron kéréseket küldenek a szervernek. Lehetnek

getter, setter metódusaik, mint például a *permissions* vagy az *editProfessionService* esetén. Az aszinkron kérések elküldése a szervernek a `$http` objektumon keresztül történik, és a szervertől érkező válasz függvényében különböző promise üzeneteket küldenek a kontrollernek, miután befejeződött az aszinkron hívás. Egy ilyen kérésre látható példa a 6. kódrészletben.

3.2. Karma és Jasmine

Az E-migrated frontend-jének tesztelése Jasmine keretrendszer és Karma test runner segítségével történt. A Jasmine egy behavior-driven (viselkedés által vezérelt) keretrendszer, amely támogatja a felhasználói felület tesztelését, míg a Karma teszi lehetővé a Jasmine tesztek futtatását különféle böngészőkben, illetve különböző eszközökön és összegezve megjeleníti az teszteredményeket. [38]

4. Alkalmazott módszerek és felhasznált eszközök

4.1. Iteratív munkamódszer

A projekt fejlesztési módszere betartja a Scrum keretrendszer [41] által diktált szabványokat, szerepköröket és ceremóniákat. A Scrum egy agilis szoftverfejlesztési eszköz, mely iteratív és inkrementáló fejlesztést tesz lehetővé. Előnyei közé tartozik, hogy a fejlesztést kisebb iterációkra, sprintekre bontja, és minden sprint végére az alkalmazás egy működő verziója kell elkészüljön. Ez a szemléletmód biztosítja a munka során az állandó visszajelzést az alkalmazás újabb és újabb funckionalitásairól (11. ábra). Az E-migrated fejlesztése 2 hetes sprintekben zajlott.

4.2. Verziókövetés és folyamatos integráció

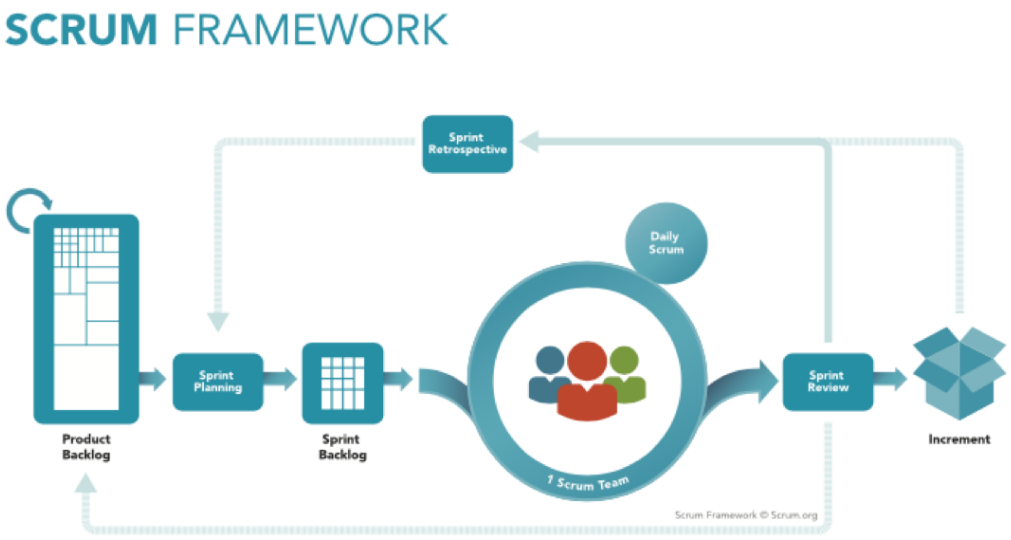
A projekt fejlesztése során a verziókövetés Git segítségével valósul meg, az E-migrated projektmenedzsment, kódbázis és CI/CD (Continuous Integration [52] / Continuous deployment [19]) rendszere pedig a GitLab.

A GitLab előnye, hogy a szoftver teljes életciklusára, a tervezés fázisától a megvalósításon, konfiguráláson át a szoftver monitorizálásáig kínál fejleszői eszközöket. Az E-migrated fejlesztése során a csapat által használt GitLab funkcionalitások közül említésre méltóak a következők:

- user story-k létrehozása és menedzselése a személyre szabható, beépített Kanban board segítségével;
- mérföldkövek létrehozása a különböző sprintek számára;
- repository menedzsment rendszer;
- folyamatos integrációs rendszer.

A projekt verziókövető módszere a Git Flow [24] megközelítést (branching model-t) alkalmazza. A különböző user story-knak megfelelő funkcionalitások fejlesztése új ágakon (branch) valósul meg, melyek miután átmentek egy Merge Request nevű ellenőrző fázison, bekerülnek az alkalmazás stabil változatát tartalmazó `master` nevű ágba.

A folyamatos integráció biztosítja, hogy az új funkcionalitások integrálása ne rontsa el az alkalmazás eddig megvalósított működőképes verzióját, és hogy az új funkcionalitások is működjenek az integráció után. Minden push esetén elindul egy CI pipeline, és végrehajtódik minden fázis (stage), amely a pipeline-t konfiguráló `gitlab-ci.yml` fájlban meg van adva (12. ábra). Az E-migrated esetén a CI pipeline három vagy négy fázisból áll, attól függően, hogy melyik ágra volt push-olva. Ha a push nem a `production` nevű ágon történik, a három fázis a következő: a kód build-elése, a tesztek lefuttatása és a statikus kódelemzés. Abban az esetben viszont ha a push a `production` nevű ágon következik be, az első fázis kiegészül egy Docker image létrehozásával és a pipeline egy negyedik fázissal, amely az alkalmazás kitelepítését valósítja meg az E-migrated teszt szerverre. A sorrend miatt, ha bármelyik előző lépés elbukik,



11. ábra. [6]A Scrum keretrendszer a fejlesztést kisebb iterációkra, sprintekre bontja. Egy iteráció a Sprint Planning ceremóniával kezdődik, amely során a csapat által kiválasztott feladatok átkerülnek a Product Backlog-ból a Sprint Backlog-ba. Minden nap sor kerül egy rövid Daily Scrum-ra, és az iteráció a Sprint Review és Sprint Retrospective ceremóniákkal zárul. A sprintek végére mindig az alkalmazás egy újabb működő verziójának kell elkészülnie.

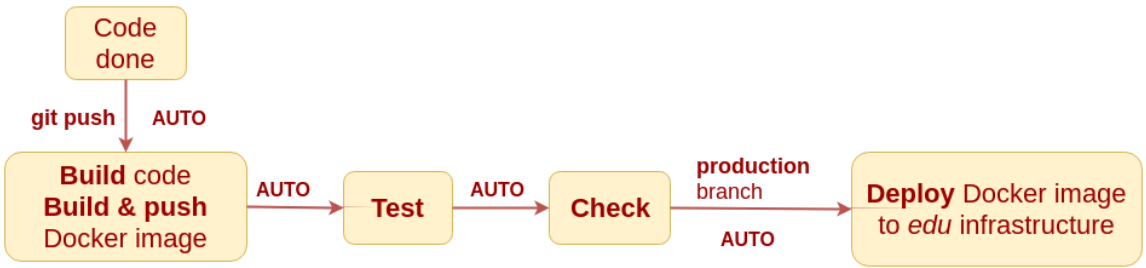
a kitelepítés nem fog megtörténni, így garantált az, hogy az alkalmazásnak csak működőképes verziója kerül kitelepítésre.

4.3. Virtualizáció és kitelepítés

A Docker [4] egy olyan virtualizációs eszköz, amely lehetővé teszi az alkalmazások kitelepítését és futtatását bármilyen környezetben Docker *konténerek (Docker Container)* segítségével. Egy konténer csak az adott alkalmazást tartalmazza, illetve azokat a könyvtárakat, és függőségeket, amelyek a működéséhez szükségesek [53].

Egy konténer létrehozásához először el kell készíteni egy *Docker Image*-et, amely egy virtuális gépet reprezentál. Ennek az image-nek a futó példányát hívják konténernek. Számos előre

[6]Kép forrása: `https://scrumorg-website-prod.s3.amazonaws.com/drupal/inline-images/2017-05/ScrumFrameworkTest.png`, utolsó megtekintés dátuma: 2018-04-16



12. ábra. Deployment és Continuous Deployment

megírt image létezik, melyek elérhetőek a különböző online nyilvántartásokban (registry), mint például a Docker Hub [5]. Saját image létrehozásakor ki lehet indulni ezekből a létező image-ekből, kibővítve azokat, az alkalmazáshoz szükséges egyéb beállításokkal és függőségekkel (13. ábra).

Az E-migrated kitelepítése Docker segítségével történik. A szervert tartalmazó konténer a `Dockerfile` állomány által definiált image egy példánya. Ez egy egyszerű, Java-t tartalmazó image-ből indul ki, és magába foglalja a szerver teljes függőségi gráfját. Felkerül a GitLab Registry-be [29], így bármilyen lokális gépen (vagy tesztszerveren) könnyen futtatható az alkalmazás. Mivel az alkalmazás az adatok tárolására MySQL adatbázist használ, szükség van a szerver mellett egy MySQL konténerre is. A két konténer együttes használatának érdekében a csapat docker-compose-t [3] használ, amely lehetővé teszi több különálló konténerből álló alkalmazások konfigurációját és menedzsmentjét. A kitelepített alkalmazás a `https://emigrated.test.edu.codespring.ro` címen érhető el.

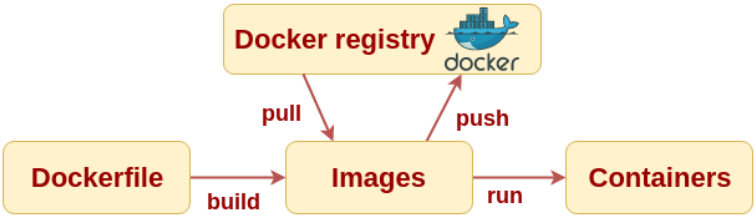
4.4. Fejlesztői környezetek

A szerver fejlesztése Eclipse [14] fejlesztői környezet segítségével történt, kiegészítve néhány hasznos pluginnal, pl. FindBugs Eclipse Plugin [25], Buildship Gradle Integration plugin [17]. A frontend kódjának fejlesztésére pedig a csapat a Visual Studio Code [11] fejlesztői környezetet választotta, mivel könnyen bővíthető olyan hasznos kiterjesztésekkel, amelyek lehetővé teszik a több különböző programozási nyelvben (HTML, CSS, JavaScript) írt frontend kód formázását és statikus kódelemzését. Ilyen használt kiterjesztések pl. JS-CSS-HTML Formatter [34] és jshint [35] .

4.5. Build és függőségkezelő eszközök

A kliens oldal függőségeit az npm (Node Package Manager) [16] kezeli, és a Gulp [8] injektálja automatikusan a forráskódba a `gulpfile.js` segítségével. A `gulpfile.js` különböző task-okból áll, melyek futtathatóak külön-külön, de akár együtt is.

A Gulp különböző automatizált műveletei segítségével megkönnyíti a fejlesztést, megkíméli a fejlesztőt a webes kód manuális kezelésétől. Beállítható egy figyelő, amely automatikusan



13. ábra. Saját *Docker Image* létrehozásához egy `Dockerfile` elnevezésű állományt kell írni és build-elni. Ehhez ki lehet indulni egy valamilyen *Docker Registry*-ben már létező image-ből, és az új image-et is fel lehet tölteni ide. Az image futtatásával *konténer* példányok jönnek létre.

frissíti a módosított állományokat (pl. js, css) az alkalmazás build mappájában is.

Mivel az E-migrated alkalmazás webes kliense egy single-page alkalmazás, célszerű volt a JavaScript forráskódokat egy közös állományba, a `main.js`-be, összefésülni és a build-elt `index.html`-ben ezekre egységesen hivatkozni. Az állományok összefűzését szintén a Gulp egyik task-ja, a `'gulp-concat'` valósítja meg.

Szerver oldali build- és függőségkezelő eszközként a csapat a Gradle mellett döntött, mely a Maven alternatívája és annak a hiányosságait igyekszik pótolni [50, 7].

Gradle scriptek írására Groovy alapú DSL (Domain Specific Languaget) vagy Kotlin script használható. A Gradle támogatja a függőségek automatikus letöltését és konfigurálását, több modulos projektek létrehozását, Eclipse projekteket generál, valamint teszteket és kódelemzőket futtat. Az `'application'` plugin segítségével becsomagolja az alkalmazást teljes függőségi gráffal és indító szkriptekkel, így az egyszerűen elmozgatható bárhova. Külső pluginokkal kiterjesztve lehetővé teszi a build-elési folyamat teljes automatizációját.

A Gradle a teljes projektet build-eli, hiszen a `'gulp'` plugin segítségével meghívja a Gulp megfelelő utasításait, ezáltal build-elve a frontend oldali kódot is. Az eredményt pedig a Gradle build mappájába másolja, és beállítja az eclipse classpath-jében, hogy az hol keresse a projekt erőforrásait és hová build-eljen.

4.6. Statikus kódelemzés

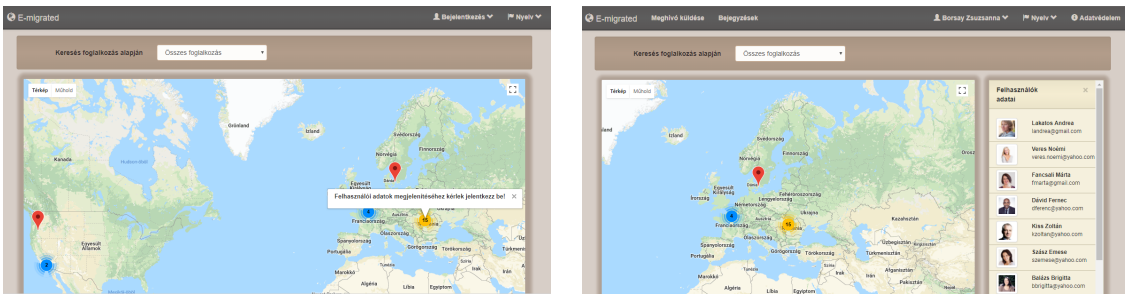
Szerver oldali statikus kódelemző a Checkstyle és a FindBugs, amelyeknek célja ellenőrizni hogy a Java forráskód megfelel-e bizonyos konvencióknak és szabványoknak [2, 6]. Segítségükkel növelhető a forráskód minősége, átláthatósága, újrafelhasználhatósága. Kliens oldalon a csapat a JSHintet választotta statikus kódelemzőnek, amely ellenőrzni a JavaScript kód helyességét, figyelmeztet, ha potenciális hibákat, problémákat, gyanús kódot talál a forráskódban [9].

A Gradle `'check'` taskja és a gulp`'jshint'`-je segítségével a projekt build-elésekor, a CI alatt, a kliens- és szerveroldali statikus kódelemzők lefutnak és megakadályozzák a hibás build-ek megjelenését a tesztszerveren.

5. Az alkalmazás működése

Ebben a fejezetben bemutatásra kerül az alkalmazás működése, valamint az, hogy a felhasználók miként érhetik el a 1.1. fejezetben taglalt funkcionalitásokat.

A webes felület főoldalán látható egy Google Maps térkép, amelyen gyűjtőpontok jelzik, hogy mely régióból hány regisztrált felhasználója van az alkalmazásnak. Közelítve a térképet ezek a gyűjtőpontok felbomlanak további térképjelzőkre, részletesebben mutatva a régiókat. Egy nem bejelentkezett felhasználó nem lát információt a regisztrált tagokról, csupán szűrheti őket foglalkozás szerint, az ablak tetején levő legördülő listából kiválasztva egy foglalkozást (14a. ábra).



(a) (b)

14. ábra. a) Az E-migrated alkalmazás főoldala, melyet nem bejelentkezett felhasználók is láthatnak, felhasználói információk nélkül. b) Az E-migrated főoldala, bejelentkezett felhasználók számára láthatóak a felhasználói információk.

Ahogyan a 14b. ábrán is látható, egy bejelentekezett felhasználó viszont már megjelenítheti az adott gyűjtőpont alatt lakó felhasználók adatait: nevüket, e-mail címüket, profilképüket, illetve nevükre kattintva át is navigálhat az adott felhasználó publikus profiljára (17b. ábra). Ha egy gyűjtőpont alatt többen laknak mint három, akkor a gyűjtőpontra kattintva az ablak jobb oldalán megjelenik egy keskeny sáv a felhasználók adataival.

Az E-migrated alkalmazás fő menüje az ablak felső részén kapott helyet. Itt tudják a felhasználók kiválasztani a kívánt nyelvet, illetve itt tudnak bejelentkezni. A *Bejelentkezés* menüpont alatt választhatnak a hagyományos bejelentkezés vagy a Facebook-kal való bejelentkezés között, illetve kérhetnek meghívót, amennyiben még nem regisztráltak. Meghívót, ahogyan a 15a. ábra is mutatja, egy e-mail cím és egy rövid motivációs levél megadásával, illetve egy önéletrajz feltöltésével kaphatnak a vendégek.

A bejelentkezett felhasználók számára a menü további menüpontokkal bővül. Megjelennek a *Meghívó küldése*, illetve *Bejegyzések* menüpontok, valamint a *Bejelentkezés* menüpont helyett most a felhasználó neve lesz feltüntetve.

A *Meghívó küldése* menüpontra kattintva, ahogy azt a neve is sugallja meghívót küldhetnek eddig még nem regisztrált ismerőseiknek és a meghívóhoz személyes üzenetet is csatolhatnak (15b. ábra). Az az ismerős aki a meghívót kapta, a meghívó küldésekor kapott egy regisztrációs tokent is, amelynek lejárati dátuma fel van tüntetve az emailben, amit kapott a személy.

(a) Meghívó igénylése (b) Meghívó küldése

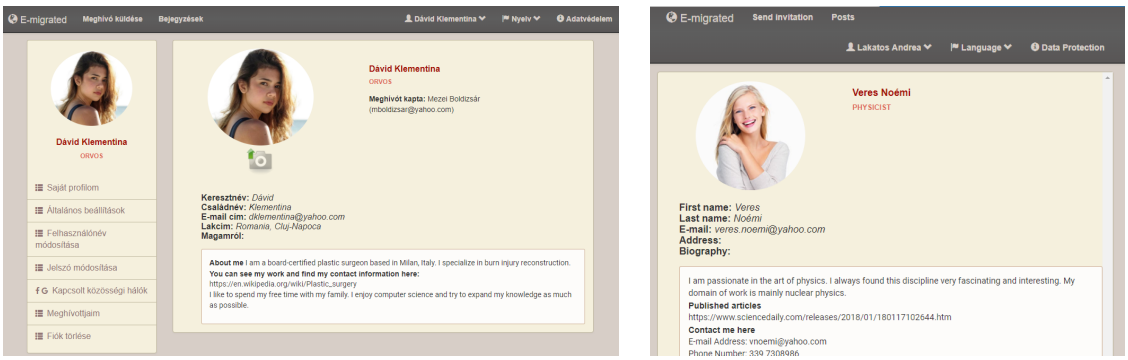
15. ábra

Lehetőség nyílik kétféle regisztrációra, a felhasználó választhat, hogy hagyományosan szeretne regisztrálni, vagy Facebook által. Ha a token lejárt, vagy nem érvényes, akkor hibaüzenet jelenik meg.

A *Bejegyzések* menüpont alatt a felhasználók böngészhetnek a bejegyzések között, melyek egymás után, a görgetés során, folyamatosan töltődnek be. Egy baloldali menüből kiválasztva a megfelelő menüpontot megtekinthetik a saját bejegyzéseiket is, illetve új bejegyzéseket hozhatnak létre (16. ábra).

16. ábra. Bejegyzések böngészése

A *Saját Profilom* menüpont alatt, melyet a saját nevükre kattintva jeleníthetnek meg a 17a. képen látható nézet jelenik meg, baloldalon egy menü, melynek fejlécén a profilképük látható. Itt szerkeszthetik a profiljuk különböző adatait, jelszót, felhasználó nevet cserélhetnek, hozzárendelhetik az E-migrated fiókjukhoz a Facebook profiljukat, megnézhetik kinek küldtek meghívót és hogy azok elfogadták-e a lehetőséget, illetve törölhetik a fiókjukat. Ha valaki hozzárendelte a Facebook profilját az eredeti, E-migrated profiljához, attól a pillanattól kezdve bejelentkezhet Facebook segítségével is. A Facebook-kal regisztrált felhasználók itt állíthatnak be egy E-migrated account-ot, ha ezentúl szeretnék, hogy legyen a rendszerhez egy felhasználónevük és egy jelszavuk.

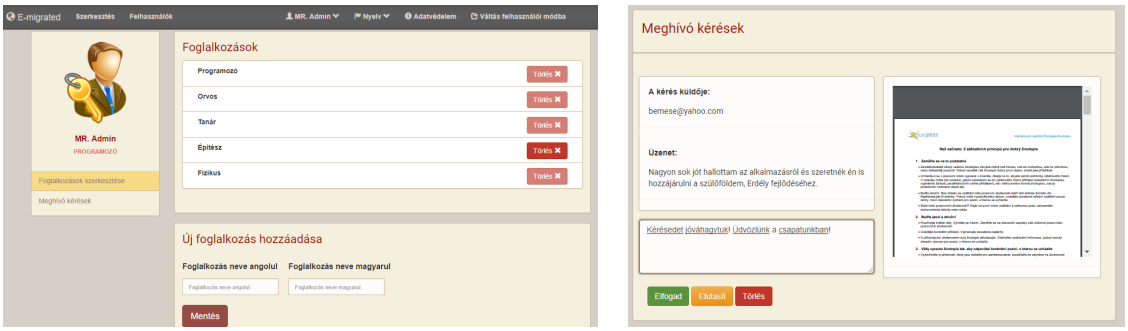


(a) (b)

17. ábra. a) Saját profil szerkesztése. b) Egy felhasználó publikus profil oldala, melyet a térképen az adott felhasználó nevére kattintva érhetnek el a bejelentkezett felhasználók.

Az adminisztrátorok számára lehetőség nyílik váltani az általános menü és az adminisztrátor menü között. Az adminisztrátoroknak készített menü öt menüpontot tartalmaz: *Szerkesztés*, *Felhasználók*, az adminisztrátor neve, *Nyelv* és *Váltás felhasználói módba.*

A *Szerkesztés* menüpont alatt (18a. ábra) az adminisztrátor számára megjelennek a foglalkozások kilistázva, amelyeket törölhet, ha nem tartozik hozzájuk felhasználó, valamint új foglalkozásokat vezethet be a rendszerbe. Ugyanezen menüpont alatt a bal oldali menüből kiválasztva a megfelelő menüpontot, megtekintheti a beérkezett meghívó kéréseket, ezeket egy rövid üzenet kíséretében elfogadhatja, elutasíthatja vagy törölheti.



(a) (b)

18. ábra. a) Foglalkozások szerkesztése. b) Meghívó kérések elbírálása.

A *Felhasználók* menüpont lehetőséget nyújt a felhasználók kilistázására, név szerinti keresésre, illetve itt tud az adminisztrátor felfüggeszteni vagy újra aktiválni egy felhasználói fiókot, megindokolva döntését.

Következtetések és továbbfejlesztési lehetőségek

Az E-migrated projekt esetén az elindulás nem volt teljesen zökkenőmentes, mivel a fejlesztés elkezdésekor megvolt a platform alapötlete, viszont a csapatnak nem állt rendelkezésére egy olyan specifikáció, amely részletesen bemutatta volna az alkalmazás funkcionalitásait, pontosítva minden felhasználói elvárást, illetve rendszer szintű követelményt. A kezdeti specifikáció elégségesnek bizonyult az alkalmazás architektúrájának a megtervezéséhez és a projekt elindulásához, de a funkcionalitások listája a fejlesztés során folyamatosan bővült, finomítódott, pontosítódott.

A fejlesztés elején a csapat kialakított egy közös célt, az E-migrated egy kezdeti prototípusának a létrehozását, amelyet sikerült megvalósítani, hiszen létrejött az alkalmazás egy funkcionalitásokban gazdag verziója, amely egy közös platformot kínál a székelyföldi és az elszármazott szakemberek számára, kapcsolatlétesítésre, együttműködésre ösztönözve őket.

Az alkalmazás többszöri bemutatása alkalmával kialakult beszélgetések során számos új ötlet, továbbfejlesztési lehetőség merült fel. Ezek közül a legfontosabbak:

- más szociális hálók, például LinkedIn, Google+ integrálása az alkalmazásba;
- a felhasználók bejegyzéseinek csoportosítása, rendezése témák szerint, és keresés a bejegyzések közt;
- bizonyos bejegyzések publikussá tétele, így azok elérhetővé válnak nem regisztrált felhasználók számára is;
- bemutatkozó videók és oktató anyagok, dokumentumok feltöltése és publikussá tétele;
- szakmai segítség, konzultáció kérése más felhasználóktól;
- felhasználók együttműködésének támogatása „gamification” eszközökkel: a felhasználók megköszönhetik egymásnak a szakmai segítséget KÖSZI pontok formájában, ezáltal a segítő nyilvános elismerésben részesül és a legtöbb KÖSZI pontok tulajdonosai kaphatnának egyéb kiváltságokat is, például több meghívót stb.;
- teljes szöveg alapú keresés a térképen profil adatok alapján.

Hivatkozások

[1] *A Bootstrap hivatalos oldala*. URL: https://getbootstrap.com/ (utolsó elérés dátuma: 2018. ápr. 17.)

[2] *A Checkstyle hivatalos oldala*. URL: http://checkstyle.sourceforge.net/ (utolsó elérés dátuma: 2018. ápr. 17.)

[3] *A Docker Compose hivatalos oldala*. URL: https://docs.docker.com/compose/ (utolsó elérés dátuma: 2018. ápr. 16.)

[4] *A Docker hivatalos oldala*. URL: https://www.docker.com/ (utolsó elérés dátuma: 2018. ápr. 17.)

[5] *A Docker Hub hivatalos oldala*. URL: https://hub.docker.com/ (utolsó elérés dátuma: 2018. ápr. 17.)

[6] *A FindBugs hivatalos oldala*. URL: http://findbugs.sourceforge.net/ (utolsó elérés dátuma: 2018. ápr. 17.)

[7] *A Gradle hivatalos oldala*. URL: https://gradle.org/docs/ (utolsó elérés dátuma: 2018. febr. 15.)

[8] *A gulp.js hivatalos oldala*. URL: https://gulpjs.com/ (utolsó elérés dátuma: 2018. ápr. 17.)

[9] *A JSHint hivatalos oldala*. URL: http://jshint.com/docs/ (utolsó elérés dátuma: 2018. ápr. 17.)

[10] *A MailGun hivatalos oldala*. URL: https://documentation.mailgun.com/ (utolsó elérés dátuma: 2018. febr. 15.)

[11] *A Visual Studio Code hivatalos oldala*. URL: https://code.visualstudio.com/ (utolsó elérés dátuma: 2018. ápr. 16.)

[12] *Az Angular Google Maps hivatalos oldala*. URL: http://angular-ui.github.io/angular-google-maps/#!/ (utolsó elérés dátuma: 2018. febr. 16.)

[13] *Az Angular UI-Router hivatalos oldala*. URL: https://ui-router.github.io/about/ (utolsó elérés dátuma: 2018. febr. 19.)

[14] *Az Eclipse hivatalos oldala*. URL: https://www.eclipse.org/ (utolsó elérés dátuma: 2018. ápr. 16.)

[15] *Az EU GDPR hivatalos oldala*. URL: https://www.eugdpr.org/eugdpr.org.html (utolsó elérés dátuma: 2018. márc. 29.)

[16] *Az npm hivatalos oldala*. URL: https://www.npmjs.com/ (utolsó elérés dátuma: 2018. ápr. 17.)

[17] *Buildship Gradle Integration - Eclipse Marketplace*. URL: https://marketplace.eclipse.org/content/buildship-gradle-integration (utolsó elérés dátuma: 2018. ápr. 16.)

[18] Nick Chamberlain. *A Better Way to Project Domain Entities into DTOs*. URL: https://buildplease.com/pages/repositories-dto/ (utolsó elérés dátuma: 2018. márc. 01.)

[19] *Continuous Deployment | Agile Alliance*. URL: https://www.agilealliance.org/glossary/continuous-deployment (utolsó elérés dátuma: 2018. ápr. 16.)

[20] *Convention over configuration*. URL: https://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch16s10.html (utolsó elérés dátuma: 2018. ápr. 04.)

[21] *Core Java2EE Patterns - Data Access Object*. URL: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html (utolsó elérés dátuma: 2018. febr. 27.)

[22] *DAO support*. URL: https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/dao.html (utolsó elérés dátuma: 2018. febr. 14.)

[23] Csala Dénes. *Hol vagytok székely(földi)ek? – a teljes diaszpóra!* 2015. URL: http://csaladenes.egologo.ro/?p=773 (utolsó elérés dátuma: 2018. febr. 27.)

[24] Vincent Driessen. *A succesful Git branching model*. 2010. URL: http://nvie.com/posts/a-successful-git-branching-model/ (utolsó elérés dátuma: 2018. ápr. 16.)

[25] *FindBugs Eclipse Plugin - Eclipse Marketplace*. URL: https://marketplace.eclipse.org/content/findbugs-eclipse-plugin (utolsó elérés dátuma: 2018. ápr. 16.)

[26] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002.

[27] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. URL: https://martinfowler.com/articles/injection.html (utolsó elérés dátuma: 2018. ápr. 16.)

[28] Martin Fowler. *PresentationDomainDataLayering*. 2015. URL: https://martinfowler.com/bliki/PresentationDomainDataLayering.html (utolsó elérés dátuma: 2018. márc. 01.)

[29] *GitLab Container Registry*. URL: https://about.gitlab.com/2016/05/23/gitlab-container-registry/ (utolsó elérés dátuma: 2018. ápr. 17.)

[30] Brad Green és Shyam Seshadri. *AngularJS*. O'Reilly Media, 2013.

[31] Juergen Hoeller. *JavaMail*. 2003. URL: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/mail/javamail/JavaMailSender.html (utolsó elérés dátuma: 2018. febr. 15.)

[32] *IT Plus Cluster*. URL: http://www.itpluscluster.ro/ (utolsó elérés dátuma: 2018. febr. 27.)

[33] N Jain, P Mangal, és D Mehta. „AngularJS: A modern MVC framework in JavaScript”. 5. évfolyam, (2015. jan.), 17–23. oldal.

[34] *JS-CSS-HTML Formatter - Visual Studio Marketplace*. URL: https://marketplace.visualstudio.com/items?itemName=lonefy.vscode-JS-CSS-HTML-formatter (utolsó elérés dátuma: 2018. ápr. 16.)

[35] *jshint - Visual Studio Marketplace*. URL: https://marketplace.visualstudio.com/items?itemName=dbaeumer.jshint (utolsó elérés dátuma: 2018. ápr. 16.)

[36] Dmitriy Kopylenko és Juergen Hoeller. *MailSender*. 2003. URL: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/mail/MailSender.html (utolsó elérés dátuma: 2018. febr. 15.)

[37] *Modular Programming*. URL: https://www.techopedia.com/definition/25972/modular-programming (utolsó elérés dátuma: 2018. márc. 02.)

[38] Adam Morgan. *Karma and Jasmine*. 2016. URL: https://scotch.io/tutorials/testing-angularjs-with-jasmine-and-karma-part-1 (utolsó elérés dátuma: 2018. febr. 16.)

[39] PascalPrecht. *Az Angular Translate hivatalos oldala*. URL: https://angular-translate.github.io/docs/#/guide (utolsó elérés dátuma: 2018. febr. 16.)

[40] Chris Richardson. *Untangling Enterprise Java*. 2006. URL: https://queue.acm.org/detail.cfm?id=1142045 (utolsó elérés dátuma: 2018. febr. 27.)

[41] Ken Schwaber és Jeff Sutherland. *The Scrum Guide*. 2017. URL: http://www.scrumguides.org/scrum-guide.html (utolsó elérés dátuma: 2018. febr. 14.)

[42] Paweł Skólski. *Single-page application vs.multiple-page application*. 2016. URL: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (utolsó elérés dátuma: 2018. ápr. 17.)

[43] *Spring Boot Reference Guide*. URL: https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/ (utolsó elérés dátuma: 2018. febr. 14.)

[44] *Spring Core Technologies*. URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#spring-core (utolsó elérés dátuma: 2018. febr. 14.)

[45] *Spring Data JPA*. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (utolsó elérés dátuma: 2018. febr. 15.)

[46] *Spring Framework Reference Documentation*. URL: https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/overview.html (utolsó elérés dátuma: 2018. febr. 27.)

[47] *Spring Security*. URL: https://docs.spring.io/spring-security/site/docs/5.0.0.RELEASE/reference/htmlsingle/ (utolsó elérés dátuma: 2018. febr. 15.)

[48] *Spring Social Reference*. URL: https://docs.spring.io/spring-social/docs/2.0.0.M4/reference/htmlsingle/ (utolsó elérés dátuma: 2018. ápr. 16.)

[49] Roy Thomas. *Representational State Transfer*. 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (utolsó elérés dátuma: 2018. febr. 12.)

[50] Lars Vogel és Simon Scholz. *The Gradle build system*. 2014. URL: http://www.vogella.com/tutorials/Gradle/article.html (utolsó elérés dátuma: 2018. febr. 15.)

[51] *Web MVC framework*. URL: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html (utolsó elérés dátuma: 2018. febr. 12.)

[52] *What is Continuous Integration | Agile Alliance*. URL: https://www.agilealliance.org/glossary/continuous-integration (utolsó elérés dátuma: 2018. ápr. 16.)

[53] *What is Docker?* URL: https://opensource.com/resources/what-docker (utolsó elérés dátuma: 2018. ápr. 17.)

[54] Ken Williamson. *Learning AngularJS*. O'Reilly Media, 2015.

[55] Avinash Zala. *Design Patterns- The Facade Pattern*. 2014. URL: https://code.tutsplus.com/tutorials/design-patterns-the-facade-pattern--cms-22238 (utolsó elérés dátuma: 2018. márc. 01.)