Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszéke

JAVA ALAPÚ FEJLESZTÉST TÁMOGATÓ ESZKÖZÖK ÉS TECHNIKÁK

TÉMALABORATÓRIUM (BMEVIAUAL00)

Kiss Előd	U479JN	k.elod98@gmail.com
Koncz Ádám	MOENI1	konczdam98@gmail.com
Tóth-Baranyi Stella	GKBPUJ	stellasipi@gmail.com

Konzulens:
Imre Gábor

Budapest, 2019

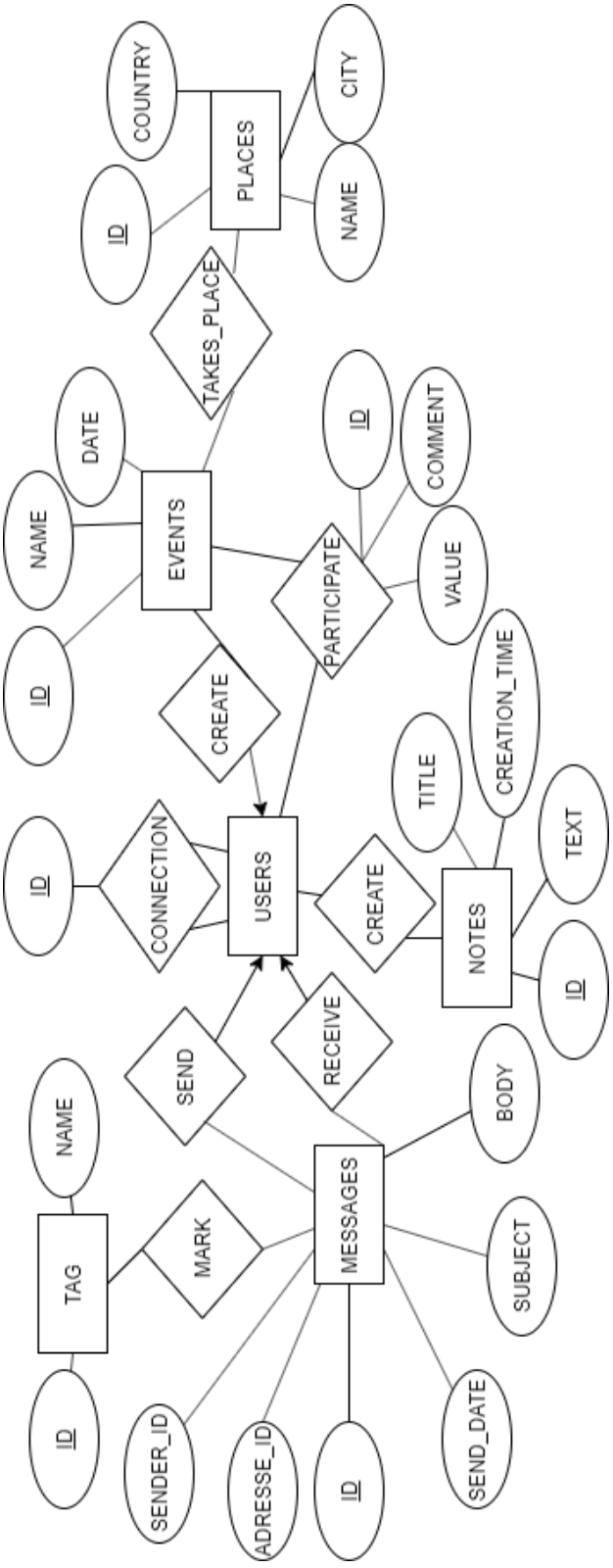
Tartalom

Bevezetés

A dokumentációnk felépítése már a tartalomjegyzékben látható, viszont kiegészítésképp elmondanánk, hogy miért választottuk ezt a formát, hogy mindenki ír egy rövidebb beszámolót és azokat fésüljük össze egy dokumentumba. Mindhárman szerettünk volna írni az összes témáról, ami alapvetően megvalósítható lett volna, de ki is akartunk egészíteni a személyes tapasztalatainkkal, hogy ki mit tanult az adott témakörből vagy hogy látta/élte meg a csoportmunkát/fejlesztést.

Továbbá fontos megjegyeznünk, hogy az alkalmazás az indulás után a *http://localhost:8080/login*-on figyel és lehet bejelentkezni a felhasználók fiókjába.

ER diagram



1. ábra

Kiss Előd beszámolója

A Java alapú fejlesztést támogató eszközök és technikák című specializációt a Java nyelv iránti érdeklődésem, és abban való fejlődni akarásom miatt választottam, mivel előtte egyedül a Programozás alapjai 3 című tárgy keretein belül találkoztam vele. A projektet 3 fős csapatokban kellett elvégezni ebben is volt már szerencsére némi tapasztalatom a Projektlaboratórium című tárgy keretein belül.

A választott projektünknek adatokkal kellett foglalkoznia (adatbázisban való tárolással), így némi gondolkodás és ötletelés után egy kezdetleges mini közösségi portál ötlete merült fel bennünk. Az oldal víziója az volt, hogy bárki szabadon regisztrálhasson rá, amihez egy felhasználó név és jelszó megadása szükséges. A belépés után a felhasználók láthatják ismerőseiket, vehetnek fel új ismerősöket, írhatnak jegyzeteket, illetve üzeneteket egymásnak (akár körüzenetet is minden ismerősüknek).

Az első újdonság az volt számomra, hogy a projektnek Maven projektnek kellett lennie, ezzel még nem volt dolgom előtte. Viszont nagyon gyorsan meg lehetett érteni, pont az egyszerűségéből fakad a Maven egyik ereje, és ennek köszönhetően nagyon könnyen tudtunk külső függőségeket felhasználni a projektben. Ilyen függőség volt például a lombok használata is. Ezt a projekt készítésének elejétől kezdve használtuk, hiszen ekkor kezdtük el megírni az entitás osztályainkat a model packagen belül. Az entitás osztályok megírásához nagy segítségünkre volt az általunk készített ER diagram amelyen tisztán látszódott milyen entitás osztályaink lesznek, és azok milyen attribútumokkal fognak rendelkezni. Összesen kilenc darab ilyen osztály lett, amelyek közt akadtak egy-egy kapcsolattal bíró, több-egy kapcsolattal bíró (ilyen volt például a User-Note kapcsolat), illetve több-több kapcsolattal bíró osztályok (például az ismerettség ilyen volt). Azonban a több-egy, és egy-egy kapcsolatokat nem képeztük le külön entitás osztályokba, hanem az entitás osztályokon belül vettük fel a kapcsolatok ”végpontjait”. Ilyen volt például a User-Note kapcsolat, ahol a User osztályon belül egy private Set-en belül voltak a hivatkozások a létrehozott jegyzetekre a Note osztályon belül egy tagváltozó az őt létrehozó Userre. Ezeket a tagváltozókat annotációkkal láttuk el,

amelyekben megadtuk a kardinalitás irányát (több vagy egy irány). Így például a User osztályon belül a private Set<Note> notes tagváltozó @OneToMany(mappedby = ”creatorUser”) annotációt kapott, amelyen belül megadtuk, hogy az adott tagváltozó a Note osztályon belül melyik tagváltozóra hivatkozik. Ennek analógiájára a Note osztályon belül a creatorUser a következő annotációt kapta: @ManyToOne így kifejezve a kardinalitás irányát. Az entitás osztályokhoz még megemlítendő a lombok használata amely automatikusan generált getterekkel és setterekkel segítette a fejlesztési munkánkat (ezt ezúton is köszönjük neki). A sok tagváltozóval bíró entitás osztályokat a builder minta segítségével példányosítottuk később. Ez a minta arról szól, hogy egy statikus builder() függvény hívása után tetszőleges sorrendben és számban állíthatjuk be egy osztály tagváltozóit a kívánt értékekre a be nem állított tagváltozók pedig nem inicializálódnak. A beállítások után pedig egy build() hívással példányosíthatjuk az osztályt. Ez jóval gyorsabb és kellemesebb, mint a sok paraméteres konstruktorok használata.

A következő lépés a repository interfészek megírása volt számunkra. Ezek az interfészek biztosították az adatbázishoz való hozzáférést, az adatok lekérdezését abból, illetve azok módosítását, törlését. Természetesen a legtöbb entitás osztályhoz készült ilyen interfész. Azonban mivel ezek a JpaRepository-ból származtak le, alapvetően biztosították számunkra az alapvető findById(Integer id) és hasonló lekérdező függvényeket, így ezeket nem kellett nekünk megírnunk. Nekünk csupán speciálisabb lekérdezéseket kellett felvennünk, ilyen volt például a MessageRepository.findAllMessagesByAddresseeAndTagIs(User user, Tag tag) is, amely egy Userhez tartozó összes levelet visszaadja amire egy bizonyos Tag rá volt rakva, ezt ugye a JpaRepository nem támgatná alapból. A repository interfészek megírása után a Service osztályokon volt a sor.

Ezeket az osztályokat @Service annotációval láttuk el. Ezekben az osztályokban kellett megvalósítsuk a bonyolultabb üzleti logikákat, így ehhez fel kellett használjuk bennük, a már elkészített repository interfészeket. A példányosítás leegyszerűsítésében és a függőségek csökkentésében nagy segítségünkre volt a Dependency Injection és a Spring által biztosított @Autowired annotáció ami pont ezt a célt szolgálja. Így a

repository interfészeinket a Service osztályon belül ezekkel az annotációkkal láttuk el. A Service osztályok függvényei (az üzleti logika) pedig @Transactional annotációt kaptak. Ez azt jelenti, hogy a benne található utasítások tranzakcióként értelmezendők, tehát vagy mindegyiket végrehajtjuk, vagy ha valahol félbeszakadna a végrehajtás akkor mindegyiket ”rollback”-elni kell, tehát úgy visszaállítani a hatásukat, hogy az adatbázis úgy nézzen ki, mint a végrehajtás előtt. Ezek a metódusok tipikusan úgy néztek ki, hogy az adatbázistól elkértük a módosítani kívánt entitást (a megfelelő repository interfész findById hívásával, megtettük a módosító lépéseket, majd a repository interfészen keresztül a save metódussal elmentettük a módosított entitást). Az elkészített Service, Repository és entitás osztályokat pedig Unit és integrációs tesztekkel ellenőriztük.

Ezekben a tesztekben építettünk a Mockito használatára. Ennek lényege az volt, hogy a tesztek során, az elkért adatokat nem egy valós adatbázisból kértük el, hanem egy ”mock”-olt adatot adtunk vissza, így az adatbázishoz közvetlenül nem nyúlunk, ahhoz való hozzáférés nélkül tudunk tesztelni.

Az utolsó lépés pedig a Controller osztályok elkészítése volt. Ekkor kezdett számomra is teljesen összeállni a kép, és helyére kerültek a dolgok, hogy mit miért is csináltunk úgy. Természetesen a Controller osztályok kaptak @Controller annotációt. A Controller osztályok feladat, hogy a böngésző által küldött kéréseket megfelelően feldolgozzák. Így ezek egyaránt alkalmazzák a már elkészített Service és Repository osztályokat. Egy-egy függvénye előtt a @GetRequest vagy @PostRequest annotációval írhatjuk le, hogy milyen http metódus esetén hvódjon meg, attribútumként pedig beállíthatjuk, hogy melyik URL esetén. Ilyen Controller osztályokkal oldottuk meg thymeleaf segítségével a webes felület kialakítását, amihez természetesen html kódot is írnunk kellett. Ez biztosította a megjelenített oldal struktúráját, és ebből továbbíthattuk az ott bevitt input értékeket a Java kódba a Controller osztályokba.

Összeségében úgy érzem sikeresen és jól teljesítettük az elvárt feladatot, amelynek elvégzése élvezetes kihívás volt hasznos dolgok tanulásával egybekötve. A csapat nagyszerűen működött együtt, GitHub-ot használtunk VersionControl gyanánt, ahol mindenki külön branchben dolgozott, így folyamatosan és készséggel tudtuk egymás

munkáját ellenőrizni. Úgy gondolom tehát, hogy mindenféleképp hasznos és célravezető volt ez a tárgy és projektmunka, amelyből sokat tanultam a Java és a Spring világáról.

Koncz Ádám beszámolója

BEVEZETÉS

Én azért választottam ezt a témát, mert az eddigi tanulmányaim során megismert programozási nyelvek közül a félév elején a Java volt a legszimpatikusabb nekem. Nyáron már dolgoztam néhány hónapot Java fejlesztőként, és a terveim szerint a későbbiekben is szeretnék majd vele foglalkozni, és ezért akartam az ismereteimet bővíteni a nyelvvel kapcsolatban.

A FÉLÉV ELEJI MUNKA

Az első feladatunk az volt, hogy alakítsunk ki 3-4 fős csapatokat. Ez szerintem azért jó dolog, mert egy szoftverfejlesztő egész életében csapatban fog dolgozni, és emiatt egészen hasznos, hogy az egyetemen belül is néhány tárgyat így kell teljesíteni. A következő, első igazi feladatunk az volt, hogy Joshua Bloch: Effective Java című könyvéből olvassunk el néhány fejezetet, és a számunkra érdekes részeket jegyzeteljük ki, majd csináljunk belőle egy néhány perces előadást. A következő feladat pedig az volt, hogy Robert „Uncle Bob" Martin: Clean Coders című videó sorozatából nézzünk meg néhány részt. Véleményem szerint ez a két feladat nagyon hasznos lenne minden szaktársunknak, mivel sajnos az egyetemen belül senki nincs nagyon rákényszerítve, hogy megtanuljon szép kódot írni. Nyilván mindenki, aki nem ír szép kódot, legbelül érzi, hogy valamit rosszul csinál, mert amint néhány száz sorosra duzzad a kódja, már teljesen el van veszve benne, és gondolatok is megfogalmazódnak benne, hogy mit kellene másképpen csinálni, hogy ne így legyen, de mégse teszi, mert éppen szűkek a határidők, vagy valami egyéb indok miatt. Véleményem szerint a legtöbb embernek elég lenne, ha csak egyszer látnák konkrétan leírva ezeket a dolgokat (vagy éppen valaki az arcukba mondaná, mint Uncle Bob), és máris beépítenék a kódolási stílusukba, amivel nagyot ugrana a kód minősége, amit produkálnak. Az is hasznos volt, hogy ki kellett állnunk néhány ember elé, és beszélni előttük, még ha csak ilyen rövid időt is, mert később, akár az egyetemen belül, akár utána, majd még sokszor kerülünk ilyen szituációba, ezért jó dolog az, hogy ha hozzászokunk.

A NAGYFELADAT

A félév további részében azt a feladatot kaptuk, hogy lépésről-lépésre bizonyos módszertanokat követve fejlesszünk le egy egyszerű webalkalmazást, amiben legalább 8 féle adatbázis entitás van.

Első lépésben ki kellett találnunk, milyen témájú legyen az alkalmazásunk, mi legyen az a sok fajta entitás, hogy mindegyiknek legyen értelme is, és valahogy kapcsolatban álljon a többivel. Ez nem ment a lehető leggördülékenyebben, nem igazán volt gyors ötletünk a témára, késve is adtuk le a specifikációt, de végül sikerült. Az volt az ötletünk, hogy egy olyan oldalt készítünk, ahova a felhasználók beregisztrálhatnak, ismerősöket szerezhetnek, üzeneteket küldhetnek egymásnak, eseményeket hozhatnak létre, meghívhatják rá a többi felhasználót, stb.

A következő feladatban végre kódolnunk kellett kicsit. Első dolgunk az volt, hogy létrehozzuk a github repót a projektnek. Ez most konkrét elvárás volt, és jó is, hogy így volt, mivel ez megint egy olyan dolog egy szoftverfejlesztő életében, ami kikerülhetetlen és rendkívül hasznos, ezért jobb minél hamarabb megbarátkozni vele. A projektben végig használtuk a lombok nevű könyvtárat, ami azért nagyon hasznos, mert rengeteg ismétlődő, boilerplate kódtól mentette meg a forráskódunkat. A feladatban a választott alkalmazásnak el kellett készítenünk egy többrétegű architektúrát megvalósító vázát, teljesen keretrendszer függetlenül. A megvalósítandó rétegek a következők voltak:

- A perzisztensen tárolandó entitások rétege. Ezek az osztályok az objektum-relációs leképezésben az objektumokat valósítják meg, tehát minden egyes példányuk egy adatbázis táblának egy sorának feleltethető meg.
- A repository réteg. Ez a réteg egyszerű adatbázis műveleteket (Create, Read, Update, Delete) szolgáltat az entitásokkal kapcsolatban a felette található rétegnek. Itt a leírás alapján csak interfészeket definiáltunk.
- A service réteg. Ez a réteg a repository réteg egyszerű adatelérést biztosító függvényei segítségével valamilyen összetettebb műveleteket hajt végre.

Már erre a feladatra, és az összes későbbire is kijelenthető, hogy nem ment egyszerűen, mivel még egyikőnk se csinált ilyesmit, és ezért mindig vagy nem igazán tudtuk elképzelni, pontosan hogy is kéne kinéznie annak, amit csinálni akarunk, vagy mert rengeteg új dolgot kellett megtanulni, megérteni és alkalmazni. Ennél az első indok

miatt voltak kicsi nehézségeink, nem értettük pontosan, hogy a kapcsolatok leképezését szolgáló kapcsolótáblák bejegyzéseit reprezentáló osztályokat is meg kellene-e írnunk, illetve hogy a service rétegben pontosan milyen komplexebb üzleti logikát kellene kitalálnunk.

KERETRENDSZEREK HASZNÁLATA

A következő feladatban végre elkezdhettünk keretrendszereket használni, hogy fusson is a kódunk, és ténylegesen csináljon valamit. Az általunk használt főbb könyvtárak a következőek: Java Persistence Api-ban megtalálható annotációk és interfészek (illetve az ezeket megvalósító Hibernate) és Spring boot. Ezek segítségével átalakítottuk az előzőleg megírt kódunkat. Az általunk választott adatbázis a MySQL, ezzel annyi dolgunk volt, hogy az ehhez megfelelő connector-t kellett hozzáadnunk függőségként a projekthez.

Az entitás rétegbeli osztályokat JPA-s annotációkkal láttuk el. Ezek segítségével tudjuk megadni, hogy milyen oszlopokat szeretnénk az egyes táblákban, hogy egyes táblák között milyen kapcsolatok vannak (egy-egy, több-egy, vagy több-több), hogyan generálódjon az egyedi kulcs, stb. A spring és a hibernate teljesen leveszi a vállunkról az SQL kódok írásának a terhét, megfelel felannotálás után teljesen automatikusan generáltathatjuk a segítségükkel az adatbázist.

A repository rétegbeli osztályokban sok változás történt. A spring datás JpaRepository interfészből leszármaztatva őket, az alapműveleteket alapból megöröklik (így ezeket kitörölhettük), és ezeknek az implementálásával sem kell foglalkoznia a fejlesztőnek. Ezen kívül a repository-k így injektálhatóvá váltak más spring boot-os osztályok számára. Az egyedi lekérdezések írásakor legtöbbször a metódusnévből leszármaztatott query-ket használtuk. Ez azt jelenti, hogy ha egy megadott szabályrendszer szerint írjuk meg a metódus nevét, akkor a spring kitalálja, hogy mi milyen lekérdezést akartunk megírni, és fordításidőben ugyan úgy legenerálja az ehhez szükséges implementációt, mint az alapműveleteket megvalósítóakat.

A service rétegbeli osztályok @Service annotációt kaptak, így ők is injektálhatóvá váltak a spring által. Azt, hogy a repository osztályokat használni akarjuk a service-ekből, nagyon egyszerűen, ugyancsak egy annotáció segítségével mondhatjuk meg a keretrendszernek (@Autowired). A metódusok @Transactional annotációval lettek

ellátva, ezzel mondjuk meg deklaratívan, hogy adatbázis tranzakción belül szeretnénk futtatni őket.

TESZTEK

A feladatban követelmény volt, hogy legyenek két féle tesztjeink is legyenek; Unit és integrációs tesztek is. A unit teszteknek adatbázistól függetlenül kell a service-ek logikáját tesztelnie, ezért az adatbázisból származó adatok „mock”-olva vannak benne. Ez azt jelenti, hogy amikor a kód le akar nyúlni az adatbázisba adatért, akkor egy előre beállított értéket/listát kap vissza. Ehhez külső segítségként a Mockito nevű könyvtárat használtuk. Az integrációs tesztek már konkrétan azt is nézték, hogy bekerül-e adatbázisba a megváltoztatott adat. Ahhoz, hogy ne a production adatbázist szemetelje a program, tesztek alatt egy kifejezetten erre a célra kifejlesztett, csak memóriában tárolt adatbázist használtunk (H2 Database).

Egy másik követelmény volt, hogy mindenki valósítson meg legalább egy komplexebb üzleti logikai funkciót TDD-t, azaz Test Driven Development-et alkalmazva. Ez egy első hallásra nagyon furcsa koncepció, először írja meg az ember a tesztet a még nem létező kódjához, majd implementálja a tesztelt programrészletet, és ha minden teszt sikeresen lefut, akkor végül refaktorálás következik.

WEB RÉTEG

Az utolsó feladat az volt, hogy valamilyen kezdetleges webes felületet adjunk az alkalmazásunknak. A feladat megvalósítására két út volt felvázolva előttünk, mi végül a thymeleaf használata mellett döntöttünk, mert valamivel egyszerűbbnek találtuk, mivel így elég volt néhány html fájlt megírni nem java nyelven, nem kellett egy külön ismeretlen kliensoldali technológiával foglalkoznunk.

A spring-boot-starter-web függőség miatt a program futtatása során automatikusan indul egy webszerver, ami az egyik porton alapból figyel, így ennek a konfigurálásával sem kellett foglalkoznunk, csak a portra bejövő böngésző kérések feldolgozására koncentrálhattunk. A legtöbb idő azzal ment el, hogy én szerettem volna, hogy ha rendesen be lehet jelentkezni az oldalon, és nem csak egy globális változóban eltároljuk, hogy melyik felhasználó van éppen bejelentkezve. Szerencsére az interneten talált példaprogramok segítségével sikerült megoldani ezt. Az én feladatom volt még ezen kívül a regisztráció, barátnak jelölés és törlés, illetve a jegyzetek létrehozásának a

megvalósítása. Ezek már a megadott példaprogram alapján egyszerűen megoldhatóak voltak.

ÖSSZEGZÉS

Én összességében nagyon örülök ennek a projektnek, a spring megismerése amúgy is terveim között szerepelt, és ennek hála egy alap szinten sikerült is. Tényleg egy viszonylag egyszerűen használható, gyorsan tanulható és kényelmes eszköz, ami nem mellesleg nagyon keresett, ezek miatt szeretnék majd jobban elmélyedni benne. Jó volt két eddig ismeretlen emberrel együtt dolgozni, szerintem egész jó csapat voltunk, és mindig jó hangulatban teltek az összeülős kódolós alkalmak, és mindenki kivette a részét a végeredmény kialakításában.

Tóth-Baranyi Stella beszámolója

1. Bevezetés

A témát elsősorban azért választottam, mert a Java programozási nyelv áll hozzám legközelebb, nagyon érdekelnek az ehhez kötődő technológiák. Jelen esetben a Spring keltette föl a figyelmem, hiszen ma már legtöbb álláshirdetésben ez egy alapkövetelmény. Magától a témától arra számítottam, hogy egy olyan alaptudást ad, amivel közelebb kerülhetek a backend fejlesztői világához, ezt majd a beszámolóm végén fejtem ki, mint összegzés.

2. Könyv és videók

Mielőtt belevethettük volna magunkat a nagy kódolásba kaptunk egy könyvet, az *Effective Java* című könyvet, amit el kellett olvasnunk és néhány részt a *Clean code* című sorozatból, amiket pedig meg kellett néznünk.

A könyvet 3 részre osztottuk, így szermély szerint én csak a *Generics, Enums and Annotations* és a *Methods* fejezeteket olvastam el. Erről kellett egy kisebb bemutató is tartanunk, ahol kiválasztottuk a kedvencünket és azt bemutathattuk. Amiket én választottam, az a *kötött wildcard-ok használata*¸ a *naming patternek helyett annotációk használata,* illetve az *overloading helyes használata*. Azért választottam ezeket, mert úgy gondolom, hogy ezek viszonylag sűrűn fordulnak elő a hétköznapi kódolásba és nem árt ezeket betartani, hisz ezek által sokkal hatékonyabb kódot tudunk írni.

Ezutáni héten a sorozatból kellett pár részt megnéznünk. Én ekkor hallottam először róla és mikor először elkezdtem nézni furcsa volt nagyon, de rájöttem, hogy igazából nagyon jó ez a stílus, amiben elmondja („elmeséli”) hogy miként kódoljunk jobban és hatékonyabban, hiszen ezzel a stílussal végig föntudta tartani a figyelmet.

3. A modell

A következő hét hétben úgymond már csak kódolnunk kellett, aminek az első része magának az alkalmazásnak kitalálása volt és az entitások létrehozás. Egyfajta leegyszerűsített közösségi oldalra/üzenetkezelőre gondoltunk, ami képes felhasználókat, üzeneteket, jegyzeteket, eseményeket és helyszíneket kezelni. Elkészítettük az ehhez

tartozó egyed-kapcsolat diagrammot (1. ábra), illetőleg részben a logikát is kitaláltuk mellé, hiszen anélkül nem igazán tudtuk volna elkészíteni.

Az ER diagram központi entitáshalmaza a *Users*, hiszen a felhasználó küldi az üzeneteket, hoz létre eseményt és mendzseli a jegyzeteit. A *Messages* egyednél az egyes üzeneteket meg lehet jelölni bizonyos tag-ekkel, a *Notes*-ban jegyzeteket tudunk létrehozni, míg az *Events*-nél eseményeket tudnak létrehozni egyes felhasználók, ahol a helyszínt a *Places* entitásból nyerjük, továbbá akik résztvesznek egy eseményen, azok tudják azt értékelni. Magyarázatra szorul még, hogy a *Users* miért is van önmagával kapcsolatban, azért mert egy felhasználónak több ismerőse lehet, emiatt nem hoztunk létre egy másik entitást, hanem egyben oldottuk meg az egészet.

Ebben a szakaszban még csak minimális kódot kellett írnunk, hiszen csak az entitásokat kellett létrehoznunk, inkább a *Maven*-nel ismerkedtünk kicsit, illetve a *Lombok*-kal. Személy szerint az előbbit már ismertem, szóval nem volt nagy újdonság, viszont az utóbbiról itt hallottam először és nagyon felkeltette az érdeklődésem. Kicsit utána is jártam, hogy milyen egyéb hasznos annotációkat lehet még használni vele és az egyik leghasznosabbnak még a @Builder-t mondanám, de ez csak egy számos többi funkció mellett.

Ezen túl még implementálunk kellett a CRUD műveleteket minden egyes osztályhoz *Service* és *Repository* osztályokon keresztül. Ezeket még nem használtuk, de itt még nem is ez volt a lényeg.

4. Spring, logikai részek és a tesztek

Erre részre kivételesen három hetet kaptunk, hiszen jóval többet kellett kódolnunk. Első körben a *SpringBoot Framework*-öt kellett a függőségeink közé berakni, hiszen csak így tudtuk használni, aztán az egyedeinket spring-es entitásokká tenni, illetve a reporsitiry és a service osztályokat is szintén „springesíteni” kellett. Eddig csak kevésbé foglalkoztunk azzal, hogy a valóságban milyen ténylegesen használható logikai függvények kellenének és ezeket meg is valósítottuk.

Mindezek után jöhettek a tesztelések. Kétfajta teszttípusból kellett teszteket írnunk az összes Service osztály összes függvényére. Volt a sima JUnit tesztek, amiket a *Mocikto* segítségével tudtunk azt a körülményt, amibe körülbelül majd futhat a valóságban az osztályunk. Majd pedig jöttek az integrtaion test-ek, röviden csak IT-k,

amikkel a teljes valós működést tudtuk szimulálni, ami sokkal megbízhatóbb eredményt produkál, mint egy egyszerűbb JUint tesztt.

Végül pedig kötelező jelleggel kipróbálhattuk milyen is a Test Driven Development. Írnunk kellett egy tesztet és aztán kellett létrehozni a hozzájuk tartozó logikát, először nem igazán tetszett, de azután rájöttem, hogy sokkal okosabb dolog így fejleszteni, hiszen már rögtön a tesztet kezdjük fejleszteni és sokkal gyorsabban kijönnek azok a hibák, amikre valószínűleg nem is számítottam volna, ha fordítva fejlesztek.

5. Webesréteg

Az utolsó kettő hétben, pedig a logikához tartozó végpontokat fejlesztettük le, illetve egy ehhez tartozó felhasználói felületet. Mi a thymeleaf-et választottuk és azzal generáltattuk a HTML-t. Őszintén nekem ez annyire nem nyerte el a tetszésem, továbbá az sem annyira, hogy a UI-jal is kellett foglalkozni, de persze látom azt a részét, hogy sokkal kézzelfoghatóbb így az egész féléves munkánk, minimális plusz tudást szereztem a webfejlesztéssel kapcsolatban és ez a fajta megoldás volt a legpraktikusabb megoldás, így, hogy nem volt sok időnk.

A REST API elkészítésében Ádám csinálta az autentikációt, a felhasználóhoz tartozó logikákat (barátnak jelölés), illetve a jegyzetek kezelését. Előd az üzenetküldést és az ehhez tartozó egyéb funkciókat valósította meg. Én pedig az eseménykezelést végeztem el.

Először is létrehoztam, hogy a bejelentkezett felhasználó láthassa, hogy milyen eseményeket hozott létre és milyen eseményekre invitálták meg őt. Majd elkészítettem az új esemény létrehozásának a felületét és az ehhez szükséges egyéb segédosztályokat, végül pedig az esemény törlése gombot is, amivel az adott felhasználó törölheti az általa létrehozott eseményeket.

Esetleg még megvalósulhatott volna meghívott események lemondás és a létrehozott eseményekhez emberek meghívása, de sajnos idő hiányában erre már nem került sor.

6. Összegzés

Először is kezdeném a csapatmunkával, ami szerintem nagyon jó volt, de nyilván nem minden ment olyan gördülékenyen, viszont szerintem egy nagyon jó kis csapat alakult ki, senki sem húzta ki magát a munka alól és szerintem mindenki érdemben tudott

dolgozni, továbbá azt is megjegyezném, hogy szerintem az Ádám dolgozott a legtöbbet ezen az egész projekten, ez commitokon is látszik.

Végül pedig én úgy érzem, hogy sikerült azt a tudást megszereznem, amire gondoltam téma kiválasztásánál. Nyilván ezt még bőven kell gyakorolni, de úgy gondolom, hogy nagyon jó kiindulási alap, továbbá nagyon is érdekel, szóval biztosan szeretnék még ezzel foglalkozni.

Munkaórák százalékos eloszlása

Kiss Előd	32%
Koncz Ádám	35%
Tóth-Baranyi Stella	33%