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

Eseménykereső webalkalmazás fejlesztése Spring, Android és Angular felhasználásával

Szakdolgozat

Készítette
Végh Éva

Konzulens
Imre Gábor

2017. december 1.

Tartalomjegyzék

1. fejezet

Bevezető

1.1. A feladat értelmezése, a tervezés célja

Napjainkban a technológia, a különböző elektronikus eszközök egyre nagyobb szerepet játszanak életünk különféle területein. A napi munka mellett segíthetnek a maradék szabadidőnk hasznos eltöltésében, egy-egy pihentető, kulturális vagy szórakoztató program megszervezésében.

Az utóbbi években egyre jobban nőtt a szórakoztatóipar kínálata, így rengeteg választási lehetőségünk van a kikapcsolódásra. Egy nap kellemes eltöltésének megszervezése azonban nem mindig egyszerű: egy adott időpontban lévő, megfelelő típusú program megtalálásához gyakran több Google-keresés, különböző honlapok megtekintése, Facebook események keresgélése szükséges, és ekkor sem feltétlen kapunk átfogó képet a lehetőségeinkről.

Léteznek honlapok, szolgáltatások, amik egy-egy közös platformot biztosítanak az események egységes formában való megjelenítésére, de ezeknél több probléma is fellép, ami megnehezíti használatukat.

1.2. Létező megoldások

A következőkben felsorolok néhány létező megoldást, részletezve előnyeiket és hátrányaikat:

port.hu

A port.hu talán a leglátogatottabb hazai kulturális programajánló portál, ami meglehetősen nagy adatbázissal rendelkezik. Legerősebb oldala tévé- és moziműsorok kínálata. A programok kereshetőek kategória, dátum és helyszín szerint (a nagyobb magyarországi városok közül választhatunk). Hátránya, hogy kevés városban kereshetünk programokat, illetve kevés programtípus szerint (TV, mozi, színház, koncert, fesztivál, kiállítás, könyv). Az oldal csak magyarul elérhető, így egy külföldi számára szinte biztosan használhatatlan, illetve a programok nagy része is inkább a magyar közönségnek szól.

lobu.hu

A lobu.hu, azaz Life of Budapest már változatosabb programokat kínál, itt már megjelennek a rendezvények, bulik, gasztronómiai események is. A programok dátum és kategória szerint kereshetőek, de az események száma elég korlátozott és egyhangú, úgy tűnik, ritkán frissítik az adatbázist. Az oldal szintén csak magyarul elérhető és a felület nem túl felhasználóbarát.

programturizmus.hu

A programturizmus.hu által nyújtott szolgáltatás tűnik a legígéretesebbnek: rengeteg változatos programtípus közül választhatunk szerte az országban, nem csak a nagyobb városokra összpontosítva. Az események tematikus elrendezésben jelennek meg, évszakoknak, ünnepeknek megfelelően. Szintén kereshetünk dátum szerint, és szinte mindig rengeteg találatot kapunk. Fő hátránya, hogy csak magyarul elérhető, annak ellenére, hogy elsősorban turisztikai oldalként funkcionál.

welovebudapest.com

A welobebudapest.com egy barátságos felületű, átlátható oldal, ami elérhető angol nyelven is. Kereshetünk dátum és kategória szerint is, ezzel a legnagyobb probléma az elérhető események csekély száma.

eventful.com

Külföldi oldalak közt több ígéretes is akadt, például az eventful.com, ami Android alkalmazással is rendelkezik, de ez érthető módon nem nyújt túl sok lehetőséget egy magyar felhasználó számára.

Mobil alkalmazások

A Google Play Store-ban keresgélve még kevesebb magyar felhasználó által használható alkalmazást találunk, ezért úgy tűnik, hogy lenne igény egy ilyen web- és mobilalkalmazásra.

Az első szakaszban leírt problémákra nyújt korszerű megoldást a szakdolgozatban tárgyalt projekt. Ugyan az alkalmazás által nyújtott funkciók nagy mértékben megegyeznek az előbbiekben felsorolt már létező szolgáltatások funkcióival, úgy gondolom, ez nem probléma, hiszen a magyar szolgáltatások funkcióit mindenképpen bővíti az új alkalmazás, és úgy érzem, a korábbi megoldások előnyeiből és hibáiból tanulva tudtam valamivel többet nyújtani. Például nagy előrehaladásnak érzem a már létező szolgáltatásokhoz képest a többnyelvűség támogatását, mivel az Android alkalmazás már magyarul és angolul is elérhető, és a webalkalmazások is könnyen bővíthetőek.

Az alkalmazás elkészítésén kívül a szakdolgozat másik célja minél több technológia megismerése, használatának elsajátítása, kísérletezés különböző megoldásokkal, ezek előnyeinek és hátrányainak összehasonlítása volt. Ennek köszönhető például, hogy a webes felület adminisztratív részéhez Thymeleaf technológiát használtam, majd a technológia korlátait és hátrányait megismerve a publikus felületet már Angular használatával készítettem el.

1.3. A diplomaterv felépítése

//TODO FELÉPÍTÉS összegzés a dolgozatról majd a végén

2. fejezet

Feladat specifikáció

A szakdolgozatban tárgyalt projekt célja egy olyan webalkalmazás, illetve egy ehhez tartozó Android alkalmazás elkészítése, ami az előző szakaszban felsorolt problémákra nyújt korszerű megoldást.

Az elkészült projekt két részre tagolható: egy webalkalmazásból és egy Android alkalmazásból. A webalkalmazás tovább bontható a kliens oldal kéréseit kiszolgáló szerver oldalra és a két megjelenítési felületre: a kliens oldal adminisztrációs felületére, illetve a kliens oldal „kliens" felületére, ami a publikus funkciókat biztosítja.

A programok publikus megjelenítésére a webes kliens alkalmazás, illetve az Android alkalmazás ad lehetőséget:

2.1. Az Android alkalmazás

Az Android alkalmazás egy olyan androidos programkereső alkalmazás, amiben a felhasználó megadhatja, hogy milyen kategóriákban és milyen távolságban érdeklik programok, rendezvények. Ezek alapján lehet listázni a programokat, keresni köztük, megnézni egy részletesebb leírást, illetve megnyitni egy térképes és egy naptáras megjelenítést is. Az események elmenthetőek, ekkor Android értesítés érkezik róluk.

2.2. A webalkalmazás

2.2.1. A szerver oldal

A szerver oldal tartalmazza az üzleti logikát, ez szolgálja ki a kliens kéréseit: a webes kliens és az Android kliens kéréseit egyaránt.

2.2.2. A kliens felület

A kliens felület az Android alkalmazás webes megfelelője. A felhasználó listázhatja a programokat, kereshet helyszín, időpont, programtípus vagy kulcsszó alapján. Adott eseményre kattintva megjelennek a részletes adatok, illetve térképen az esemény helyszíne.

Regisztráció után a felhasználó „megcsillagozhat", elmenthet eseményeket, amiket így később könnyen megtalálhat, illetve email értesítést kaphat róluk.

2.2.3. Az adminisztrációs felület

Az elkészített alkalmazáshoz tartozik egy adminisztratív felület, ahol az arra jogosult programszervezők létrehozhatnak saját eseményeket, majd szerkeszthetik, törölhetik azokat. A webalkalmazással felvett események bekerülnek az adatbázisba, amit a webes kliens felület és az Android alkalmazás használ. Az adminisztratív felületen két jogosultságot különböztetünk meg, az admint és a szuperadmint.

Az admin bejelentkezés után megtekintheti a korábban létrehozott eseményeit listanézetben. Az egyes eseményeket törölheti vagy megnyithatja a szerkesztés módot, ahol egy űrlapon módosíthatja a program adatait, majd elmentheti a változásokat. Létrehozhat új eseményt is, amit a szerkesztéshez hasonlóan egy űrlap kitöltésével tehet meg. Itt a következő adatok beállítására van lehetőség:

- esemény neve
- helyszín
- rövid összefoglaló leírás
- részletes leírás
- kezdés időpontja
- befejezés időpontja
- belépő
- kategória
- honlap URL
- Facebook URL

A szuperadmin mindent megtehet, amit az admin, azzal a különbséggel, hogy ő az összes admin által létrehozott eseményhez hozzáfér, és módosíthatja, törölheti azokat.

2.3. Use case

A 2.1. ábra összegzi a különböző szerepkörök use case-eit. A be nem jelentkezett felhasználó az anonim felhasználónak felel meg, a regisztráció után belépett egyszerű felhasználó Regisztrált felhasználónak. A hirdetők adminisztrátor szerepkört kapnak, míg az üzemeltető a szuperadminnak felel meg. Megkülönböztetjük még az androidos felhasználót bejelentkezéstől függetlenül.

2.3.1. Az egyes use case-ek részletes leírásai

Use case: események listázása

Leírás: A felhasználó listázhatja az elérhető eseményeket, ekkor az események néhány fontosabb tulajdonsága jelenik meg.

Szereplők: Anonim, Regisztrált, Androidos, Hirdető, Üzemeltető

Megkötések: Csak jövőbeli események jelennek meg.

Use case: események keresése

Leírás: A felhasználó különböző keresési feltételek szerint kereshet az események között, úgy mint dátum intervallum, helyszín, névbeli egyezés, kategória.

Szereplők: Anonim, Regisztrált, Androidos, Hirdető, Üzemeltető

Megkötések: -

Use case: esemény részletes nézetének megnyitása

Leírás: A felhasználó megtekintheti egy adott esemény részletes nézetét, hogy bővebb információhoz jusson az esemény részleteit illetően.

Szereplők: Anonim, Regisztrált, Androidos

Megkötések: -

Use case: térképes nézet megnyitása

Leírás: A felhasználó megnyithatja a térképes nézetet, amin az egyes eseményekhez rendelt markerek jelennek meg, rájuk kattintva az eseményhez tartozó infobox jön elő.

Szereplők: Anonim, Regisztrált, Androidos

Megkötések: -

Use case: naptáras nézet megnyitása

Leírás: A felhasználó megnyithatja a naptáras nézetet, amin az egyes napokra kattintva az aznap történő eseményeket listázza a program.

Szereplők: Androidos

Megkötések: Csak jövőbeli dátumokat lehet kiválasztani.

Use case: események mentése

Leírás: A felhasználó menthet eseményeket, amikről a webes felületen email értesítés, az Android alkalmazásban Android értesítés érkezik az esemény kezdete előtt 5 órával.

Szereplők: Regisztrált, Androidos

Megkötések: -

Use case: feliratkozás hirdetőre

Leírás: A felhasználó feliratkozhat hirdetőkre, ekkor a hirdető által közzétett új eseményekről a webes felületen email értesítés, az Android alkalmazásban Android értesítés érkezik.

Szereplők: Regisztrált, Androidos

Megkötések: -

Use case: feliratkozás helyszínre

Leírás: A felhasználó feliratkozhat helyszínre, ekkor a helyszínen közzétett új eseményekről a webes felületen email értesítés, az Android alkalmazásban Android értesítés érkezik.

Szereplők: Regisztrált, Androidos

Megkötések: -

Use case: adatok módosítása

Leírás: A felhasználó módosíthatja az email címét, jelszavát és az értesítési beállításokat.

Szereplők: Regisztrált, Androidos, Hirdető, Üzemeltető

Megkötések: -

Use case: témaválasztás

Leírás: A felhasználó választhat világos és sötét téma közül, ekkor az alkalmazás színei, az ikonok és stílusok ennek megfelelően módosulnak.

Szereplők: Androidos

Megkötések: -

Use case: regisztráció

Leírás: A felhasználó létrehozhat egy új felhasználói fiókot megfelelő fióknév, email cím és jelszó megadásával.

Szereplők: Anonim

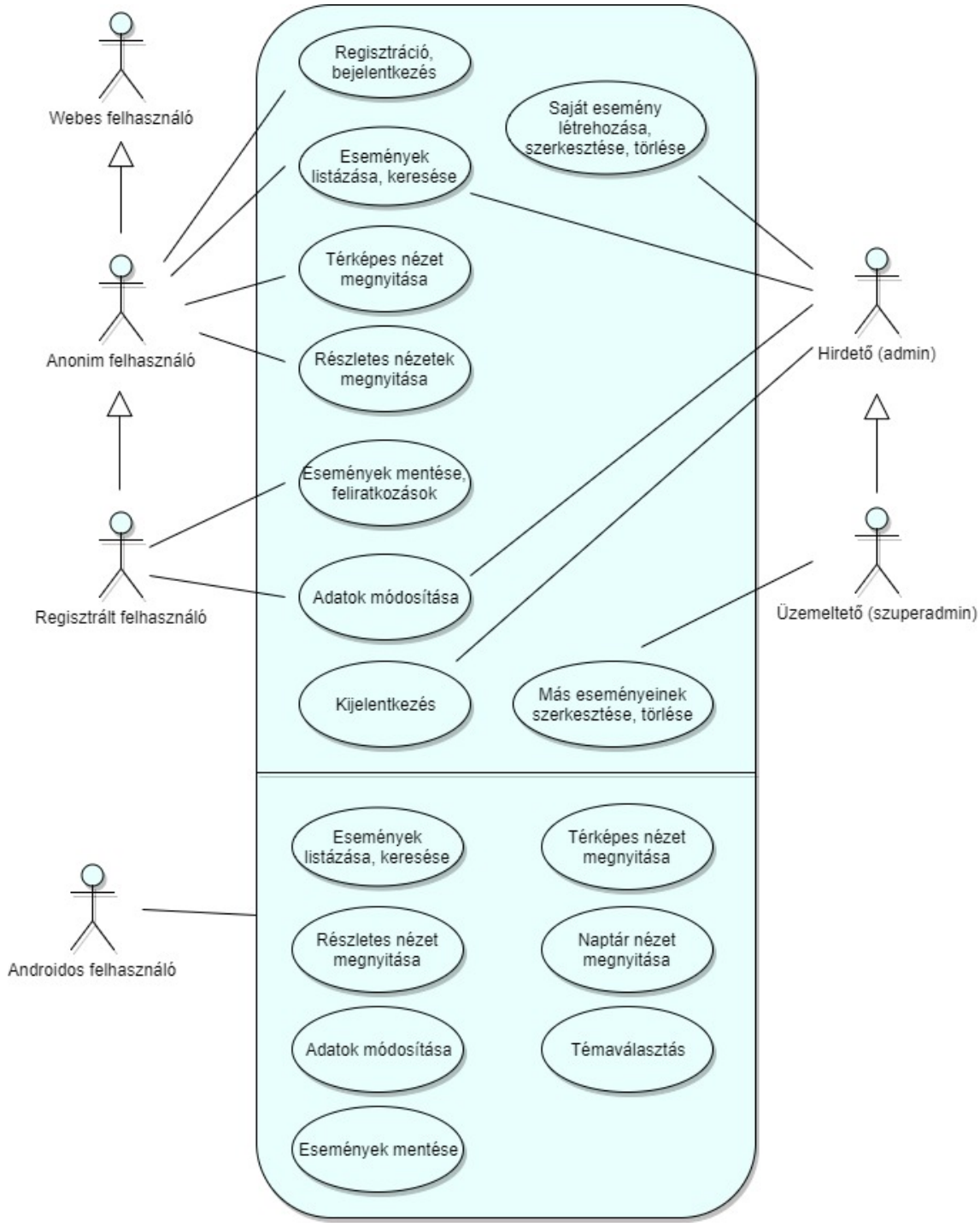
Megkötések: -

Use case: bejelentkezés, kijelentkezés

Leírás: Az Anonim felhasználó bejelentkezhet, a bejelentkezett felhasználó kijelentkezhet az alkalmazásból.

Szereplők: Anonim, Regisztrált, Hirdető, Üzemeltető

Megkötések: -



2.1. ábra. *Az alkalmazás szerepköreinek összevont use case diagramja*

3. fejezet

Választott technológiák

A következőkben felsorolom a projekt elkészítése során használt főbb technológiákat. A kisebb, kevésbé fontos technikákat, könyvtárakat itt nem részletezem, ezek használatára csak a következő fejezetekben térek ki (4 fejezet és Megvalósítás fejezetek).

3.1. Maven

A Maven egy parancssori build automatizáló eszköz, amely képes a függőségek (.jar-ok) feloldására és automatikus letöltésére, akár tranzitív függőségeknél is. Leírja, hogyan épül fel a projekt, illetve meghatározza a projekt más moduloktól vagy könyvtáraktól való függőségeit. Ezeket az információkat a pom.xml (Project Object Model) nevű fájlban tárolja, ami magában foglalja többek között a projekt nevét, tulajdonosát és a függőségek listáját.

A tesztelést integrálja a buildelési folyamatba, Unit és integrációs tesztek is támogatottak. Erősen testre szabható, de csak akkor van rá szükség, ha eltérünk a default-októl (pl. könyvtárstruktúra). Ez annak köszönhető, hogy a Maven fontos alapelve a Convention over Configuration (konvenciók a beállítások előtt), amivel kevesebb konfigurációs beállítást kell elvégeznünk, arra ösztönöz, hogy az alapértelmezett működést használjuk.

3.2. Spring

A Spring egy nyílt forráskódú, Java alkalmazás keretrendszer, amelynek elsődleges célja, hogy a vállalati környezetbe szánt Java alkalmazások fejlesztését egyszerűbbé tegye. Az IoC (Inversion of Control) tervezési minta megvalósításával segít az alkalmazás objektumainak konfigurálásban, „összedrótozásában". Az objektumgráfok előállítását az injektor végzi el. Az általunk végzett konfiguráció megoldható annotációk használatával. A Spring kezeli az infrastruktúrát, így a fejlesztés során az alkalmazás fejlesztésére koncentrálhatunk. Használatával biztosíthatjuk az elemek közötti laza csatolást, az alkalmazás különböző rétegeinek szétválasztását, többek között a megjelenítés leválasztását a működési logikáról.

Moduláris felépítésű, így csak azokat a részeit kell használnunk, amire szükségünk van.

Támogatja a tranzakciókezelést, a távoli hozzáférést, teljes értékű MVC kertrendszert biztosít. Lehetővé teszi, hogy POJO-k (plain old Java object) segítségével építsük fel az alkalmazásunkat.

3.3. Spring Boot

Az alkalmazás megvalósításához a Spring Boot keretrendszert választottam. A Spring keretrendszer szintén támogatja a CoC-t (ld. 3.1 szakasz), tehát nem szükséges külön konfigurációt vagy programkódot írnunk, ha az elterjedt konvenciókat követjük. A szoftverünk csak azon részeihez szükséges konfigurációt vagy extra kódsorokat létrehoznunk, amelyek eltérnek a praktikusan megválasztott alapértelmezett működéstől.

A Spring Boot ezeket az alapelveket emeli még magasabb szintre. Tovább egyszerűsíti az alkalmazás konfigurálását és a komponensek közti integrációt. A megírt kód alapján feltérképezi a programozó szándékait, és automatikusan konfigurálja a kapcsolódó háttérszolgáltatásokat. Az így hozzáadott funkcionalitás programozói erőfeszítés nélkül jön létre, így jelentősen leegyszerűsödik a fejlesztés folyamata.

3.4. Spring MVC

A Spring MVC a Spring keretrendszer része, ami egy kérés (request) alapú keretrendszer és szorosan kapcsolódik a Servlet API-hoz.

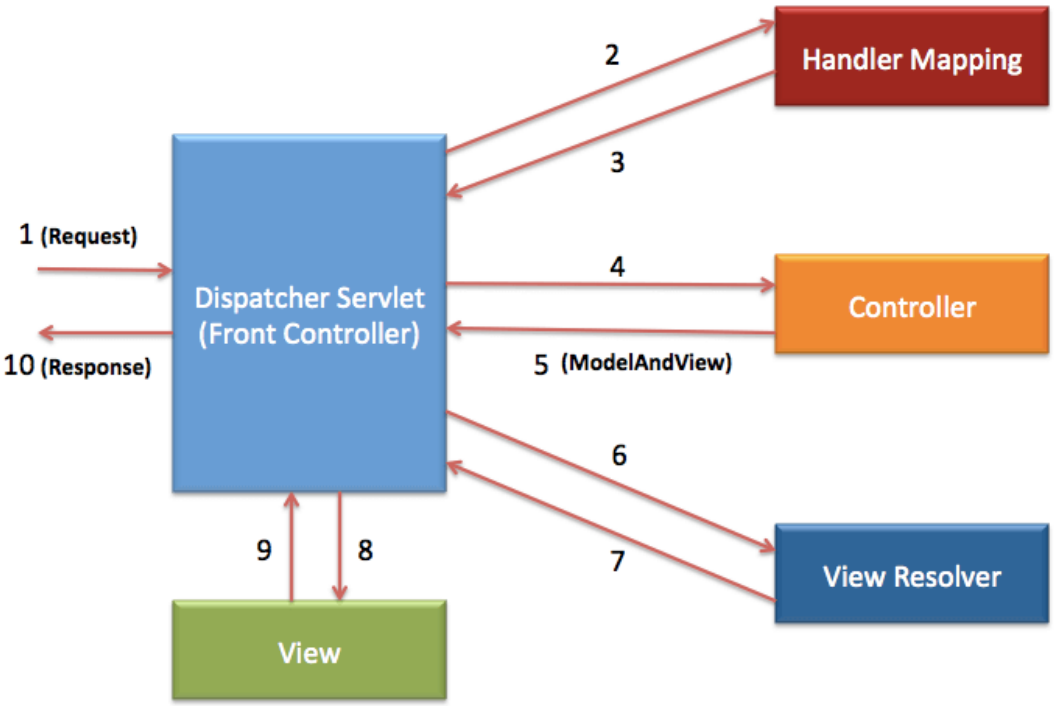
A Spring MVC használata során a kérések egy DispatcherServlet nevű szervlethez érkeznek be, aminek a felelőssége, hogy a kéréseket a megfelelő interfészekhez delegálja, úgy mint:

HandlerMapping: Kiválasztja a megfelelő objektumokat a kérések lekezelésére valamilyen paraméter, vagy feltétel alapján.

Controller: A modell és a felület közötti réteg az MVC struktúrában, feladata a bejövő kérések kezelése és a megfelelő válaszok visszaküldése.

ViewResolver: Visszaadja a felület sablonját a logikai neve alapján, amit lerenderelve vissza lehet küldeni a kliensnek válaszként.

View: A kliensnek küldött felület a modellből érkezett adatokkal kitöltve.



3.1. ábra. *Kérés kiszolgálása a Spring MVC-ben*

3.5. Spring Data

A Spring Data egy külön modul az adatelérés támogatására, amivel az adatelérési kód nagy része megspórolható, hiszen egyszerűen tudunk saját entitásra specifikus repository-t írni, ami a JpaRepository<T, ID extends Serializable> interfész leszármazottja.

Az interfészbe felvett findBy... metódusok nevei alapján egyéb lekérdezések generálhatók.

3.6. JPA

A JPA egy szabványos ORM API specifikáció, aminek az alkalmazás relációs adatainak kezelése a fő feladata.

Az ORM keretrendszerek legfőbb hátránya, hogy sokszor nehézkes az általánostól eltérő, összetett funkciókat megvalósítani, mert túl magas szinten működnek. Ezt a JPA azzal oldja meg, hogy csak interfészeket specifikál, ezzel téve lehetővé a több lehetséges implementációt. A JPA legnépszerűbb implementációja a Hibernate, én is ezt használtam a projekt elkészítése során.

Egy JPA entitás egy olyan osztály, melynek példányait a JPA relációs adatbázisban perzisztensen tárolja. Az O – R leképezés annotációkkal konfigurálható.

Az entitásokat az EntityManager interfészen keresztül tudjuk kezelni, ezen keresztül érhetjük el a perzisztenciakontextust, ami a memóriabeli entitások és az adatbázis közti kapcsolatot jelenti.

3.7. REST

A szabványos HTTP kérés – válaszokra építve, az elérendő erőforrásokat külön-külön egyedi URI-hoz rendelve építjük fel szolgáltatásunkat. Az URI-k egységes interfészt biztosítanak a kliens számára. Minden kérésre azonos formátumban reagál a szerver, ez általában JSON, HTML vagy XML.

REST használatával biztosíthatjuk a kliens és a szerver egymástól való elkülönítését, hiszen a köztük lévő kommunikáció az interfészeken zajlik. A kommunikáció állapotmentes, tehát nem tárolhatunk semmilyen információt a klienstől a kérések között, a kéréseknek minden szükséges adatot tartalmazniuk kell.

3.8. Thymeleaf

A Thymeleaf egy olyan Java alapú template engine, ami XML/HTML/HTML5 template fájlok kezelését biztosítja rugalmasan bővíthető dialektusokkal. A th: névtérbe tett attribútumokkal bővítve egyszerűen designolható, a template fájl szerkesztéséhez nincs szükség szerver futtatására. Ez nagyban meggyorsítja a fejlesztési folyamatot, illetve a template fájlok továbbíthatók a designernek, aki egyszerűen dolgozhat velük segédprogramok használata nélkül.

Thymeleaf fragmentek használatával az egyes elemek, layoutok könnyen újrahasznosíthatóak, ezzel is átláthatóbbá téve a kódunk szerkezetét.

3.9. Angular 4

Az Angular egy olyan frontend keretrendszer, amelynek telepítésével egy olyan JavaScript könyvtárhoz jutunk, amivel szinte bármit meg tudunk valósítani kliens oldalon. Szerver oldaltól függetlenül használható, az alkalmazás futtatása nem igényel telepítést. Moduláris felépítésének köszönhetően elődeinél gyorsabb betöltést és könnyű tesztelhetőséget biztosít.

Saját validálási módszereivel egyedi ellenőrzéseket végezhetünk kliens oldalon, így csökkentve a felesleges kérések számát a szerver felé. Fejlett eseményvezérelt rendszerén keresztül megoldható a többirányú adatkötés: egy adat több helyen történő megjelenését összehangolja, ezzel is csökkentve a lekérések számát.

Elsődleges nyelve a TypeScript, ami erős típusosságával sokkal megbízhatóbb és biztonságosabb kódok írását teszi lehetővé, hiszen számos hiba már kódolás közben vagy fordítás alatt kiderülhet.

Az Angular 2-ben mutatkozott be az Angular Universal, amely még szerveroldalon elkészíti, előrendereli a first viewt, és statikus tartalomként küldi el a kliensnek. Ezután kezdi meg a webapp elemeinek letöltését és dinamikus beágyazását az oldalba. Ennek köszönhetően oldható meg az SPA (Single Page Application) kivitelezése, hiszen az adatok változása esetén csak a tartalmat kell frissíteni, nem az egész oldalt.

3.10. AJAX, jQuery, KnockoutJS

Az egyszerű HTTP kérésekkel szemben az AJAX használata lehetővé teszi, hogy a böngészőből aszinkron módon küldjünk kéréseket a szervernek. A szerver válaszát szintén JavaScript segítségével fogadjuk. Ennek köszönhetően az oldal elemei egymástól függetlenül módosíthatóak, így például adatfrissítés esetén nem szükséges az egész oldalt újratölteni, csak az egyes elemeket, és a felhasználói élmény folyamatosabb, kellemesebb lesz.

A jQuery segítségével tudjuk a DOM-ot (Document Object Model) manipulálni, eseménykezelőket definiálni az oldalon.

A KnockoutJS ezeket kiegészíti azzal, hogy kétirányú adatkötést valósít meg, a HTML elemeket hozzárendeli a ViewModel objektumokhoz, így az egyik módosítása maga után vonja a másik változását.

Ezt a három felsorolt technológiát a Thymeleaffel együtt használtam, az Angular alapú oldalaknál mindezt az Angular magában foglalja, ezért ott nem kellett vele külön foglalkozni.

3.11. Bootstrap

A Bootstrap egy olyan front-end keretrendszer, aminek segítségével egyszerűen készíthetünk reszponzív weboldalakat. Olyan CSS szabályokat, JavaScript függvényeket és HTML mintákat tartalmaz, amelyek segítségével összetettebb komponensek is könnyen elkészíthetők a saját ízlésünkre szabva.

4. fejezet

Tervezés

A feladat specifikációban leírtam, hogy a projekt egy webalkalmazásból és egy Android alkalmazásból áll. A szakdolgozat elkezdésekor az Android alkalmazás már részben kész volt, a később elkészült szerver nélkül is megállta a helyét: az adatokat egy külső REST API-n keresztül (http://api.eventful.com) kapta, és ezekkel dolgozott. Miután megszületett az igény, hogy saját eseményekkel lehessen feltölteni, az elkészítendő back-end alkalmazás az Android alkalmazás igényeihez igazodott (pl. modell osztályok felépítése), és ez nagyban meghatározta a back-end alkalmazás felépítését.

4.1. Architektúra

A modern webes alkalmazások tipikusan többrétegű architektúrára építenek. Minimum három réteggel rendelkeznek: megjelenítés, service (üzleti logika) és repository réteg (adatkezelés).

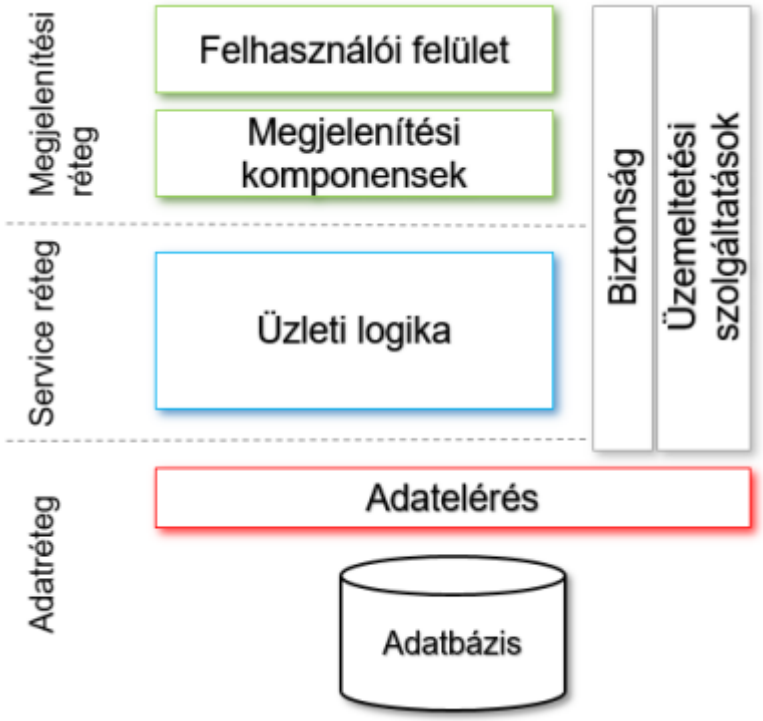
Az alkalmazás rétegekre bontása lehetővé teszi az egyes rétegek egymástól függetlenül történő fejlesztését, sőt, akár teljes cseréjét is, mert a rétegek csak egymás interfészeitől függenek. Így egy könnyebben fejleszthető és karbantartható programot kapunk, ahol a rendszer különböző szintjei egymástól függetlenül tesztelhetők.

A projekt architektúráját ennek megfelelően terveztem kialakítani:

Adatréteg: Az adatok tartós tárolásáért felelős réteg: magában foglalja a modellt, az adatelérést és logikailag magát az adatbázist, bár gyakran az adatbázis önálló, külső rétegként jelenik meg. A modellben találhatóak az entitások, amik a tárolandó adatokat reprezentálják. Az adatelérés részei a Repository interfészek, amelyek az elemi adatelérési műveletekhez (entitás létrehozása, törlése, módosítása, keresése) szükséges metódusokat deklarálják.

Service: A Service rétegben található osztályok az üzleti logikát tartalmazzák, ezek futtatják az üzleti folyamatokat. A Repository interfészeket felhasználva egy-egy használati esetet valósítanak meg, és kikényszerítik a több entitást is érintő üzleti szabályokat.

Megjelenítés: A megjelenítési réteg feladata az adatok valamilyen módon történő megjelenítése, a felhasználói események kezelése az üzleti logikai réteg segítségével.



4.1. ábra. *A projekt felépítése*

4.2. Adatréteg

4.2.1. Az entitások

Az adatokat reprezentáló entitások a projekt model csomagjában kapnak helyet. Az egyszerű POJO (plain old Java object) osztályok megírása után az entitások megfelelő JPA annotációkkal történő ellátása biztosítja, hogy automatikusan megtörténjen az O – R leképezés.

A model csomagban található osztályok csak adattárolásra valók: nincs igazán viselkedésük, egyszerű konstruktorokkal, getter, setter metódusokkal rendelkeznek, illetve add...(), remove...() metódusokkal, ahol kollekció kezelésére van szükség. Ezen felül a felüldefiniált hashCode(), equals(), toString() metódusok fordulnak elő. Éppen ezért ezeket nem fogom jobban részletezni.

Meg kell említenem, hogy a model csomagban több enum is helyet kap, amik nem számítanak entitásnak, hiszen nem képződnek le adatbázis táblákra, de az entitásokkal együtt fogom felsorolni őket, mert logikailag ide tartoznak.

Az osztályok tagváltozói mind beszédes nevet kapnak, amiből egyértelműen kiderül milyen célt szolgálnak (lásd 4.2. ábra), ezért a következőkben csak az osztályok funkcióját és kapcsolatait részletezem:

User: Az alkalmazásba regisztrált felhasználók adatait tárolja. Rendelkezik névvel, jelszóval, email címmel és a regisztráció idejével.

Kapcsolatai:

Role [0..]:* Ezeket a szerepet birtokolja, ez határozza meg, milyen akciók végrehajtására van jogosultsága.

UserSettings [1]: Az általa mentett fiókbeállítások.

Location [0..]:* A helyszínek, amikre feliratkozott.

User [0..]:* A hirdetők, akikre feliratkozott.

Role: Az alkalmazásban a felhasználók által felvehető szerepeket tárolja.

Kapcsolatai:

User [0..]:* Azok a felhasználók, akik ezzel a szereppel rendelkeznek.

UserSettings: A felhasználók által mentett fiókbeállításokat tároló osztály.

Kapcsolatai:

User [1]: Az a felhasználó, akihez a beállítások tartoznak.

Event: Az eseményt reprezentáló osztály.

Kapcsolatai:

User [0..]:* Kétféle kapcsolatban állhat felhasználóval: az egyik, ha ez a felhasználó hozta létre, a másik, ha a felhasználó elmentette ezt az eseményt.

Price [3]: Az eseményhez tartozó belépő árak: normál, diák és nyugdíjas jegyárak.

Location [1]: Az esemény helyszíne.

EventType [1..]:* Az eseménytípusok, amikbe az esemény besorolható.

Price: Egy belépőjegy árat reprezentáló osztály, amelyet az összeg, a pénznem és belépő típus határoz meg.

Kapcsolatai:

Event [1]: Az esemény, amihez a belépőjegy tartozik.

Currency [1]: A belépő pénzneme (például Ft, $, €).

PriceType [1]: A belépő típusa (normál, diák vagy nyugdíjas jegy).

Currency: Egy pénznemet reprezentáló enum.

Kapcsolatai:

Price [0..]:* A pénznemhez tartozó belépő.

PriceType: Egy belépőjegy típust reprezentáló enum.

Kapcsolatai:

Price [0..]:* A jegytípushoz tartozó belépő.

EventType: Egy eseménytípus reprezentálására alkalmas enum.

Kapcsolatai:

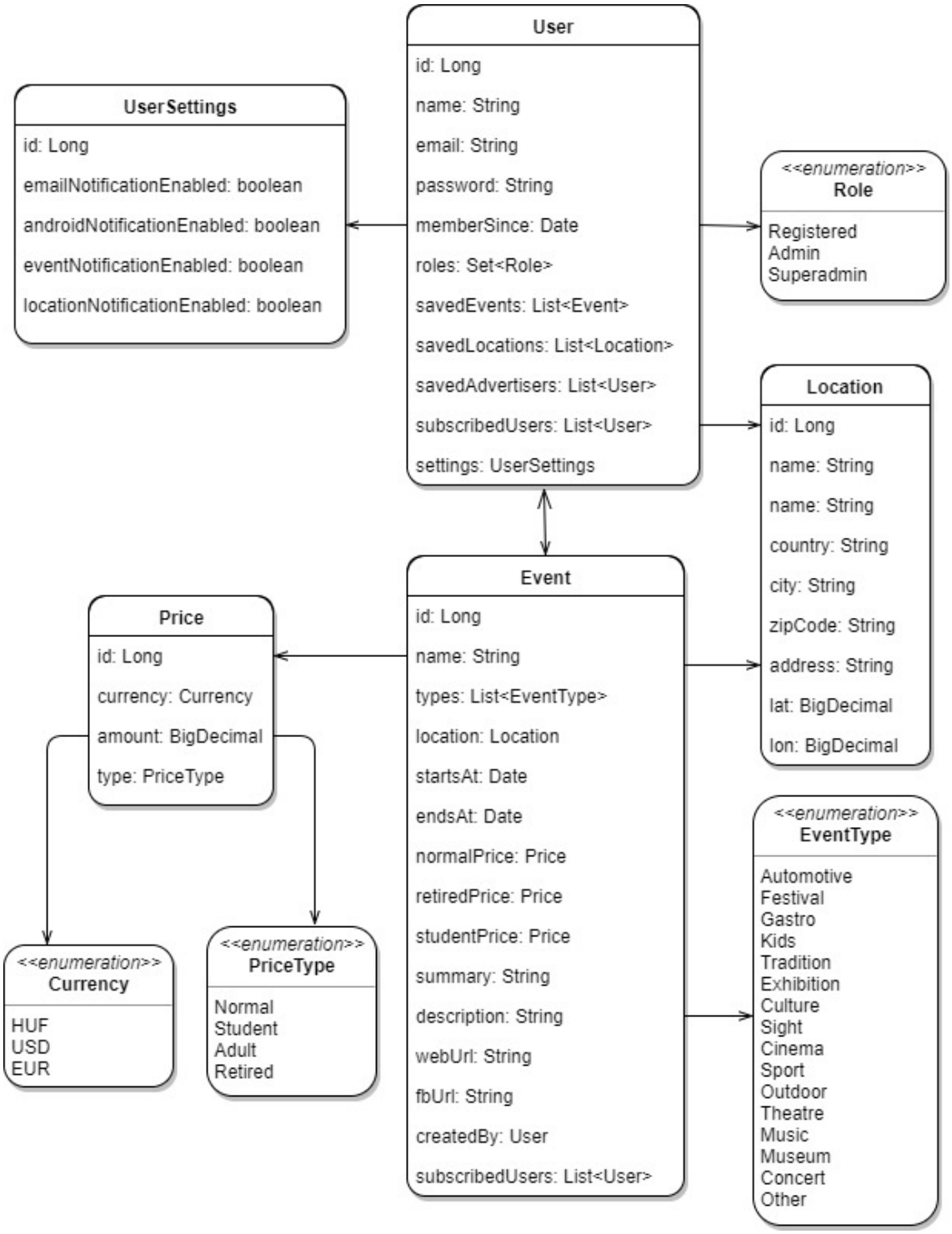
Event [0..]:* Azok az események, amik besorolhatók ebbe a típusba.

Location: Egy helyszínt reprezentáló osztály.

Kapcsolatai:

Event [0..]:* Azok az események, amelyek itt zajlanak le.

User [0..]:* Azok a felhasználók, akik feliratkoztak erre a helyszínre.



4.2. ábra. *Az alkalmazás entitásai és kapcsolataik*

4.2.2. Az adatelérés

Az adatelérést az egyes entitásokhoz tartozó Repository interfészek biztosítják a Spring Data segítségével. Nem minden entitáshoz szükséges Repository-t létrehozni, csak azokhoz, amit közvetlenül (azaz nem más entitások kapcsolatain keresztül) szeretnénk elérni az adatbázisból. Ilyen az Event, a User és a Location entitás.

LocationRepository

A LocationRepository-ban nincs szükség egyedi metódusok deklarálására, elég a Spring Data által alapból implementált CRUD (create, read, update, delete) műveleteket megvalósító metódusok használata.

UserRepository

A UserRepository-ban a defaulton kívül további metódusokra van szükség. Ezek egyszerűbb lekérdezéseket valósítanak meg, ezért könnyen generálhatók névkonvenciókból:

findByEmail(String email): User Visszatér a paraméterben kapott emaillel rendelkező felhasználóval.

findByName(String userName): User Visszatér a paraméterben kapott névvel rendelkező felhasználóval.

EventRepository

Az eseményeken végzett műveletekhez sokkal több és összetettebb lekérdezésre van szükség, mert több szempont szerint is szeretnénk keresni köztük és gyakran egyszerre több feltételt is ki kell elégítenie a keresésnek.

- Névkonvenciókból generált metódusok:

  findByTypes(List<EventType> type): List<Event> Visszatér a kapott típussal rendelkező eseményekkel.

  findByStartsAtAfter(Date date: List<Event>) Visszatér a kapott dátum után zajló eseményekkel.

  findByEndsAtBefore(Date date: List<Event>) Visszatér a kapott dátum előtt végződő eseményekkel.

  findByStartsAtBetween(Date starts, Date ends): List<Event> Visszatér a két dátum között kezdődő eseményekkel.

  findByLocationCountry(String country): List<Event> Visszatér a kapott országban zajló eseményekkel.

  findByLocationCity(String city) : List<Event> Visszatér a kapott városban zajló eseményekkel.

  findByNameContaining(String phrase) : List<Event> Visszatér a kapott szöveggel akár részegesen megegyező nevű eseményekkel.

  findByCreatedBy(User user) : List<Event> Visszatér a kapott felhasználó által létrehozott eseményekkel.

- Egyedi metódusok:

findByTypeLocationKeyword(List<EventType> types, Location location, String phrase): List<Event>
Névegyezés, helyszín és eseménytípus egyezése alapján ad vissza eseményeket.

Ahhoz, hogy egyszerre több szűrési feltétel alapján tudjunk keresni, egyszerűbb saját lekérdezéseket írni egy saját EventRepositoryImpl nevű osztályban, ami megvalósítja az EventRepositoryCustom interfészt.

A lekérdezés implementálását a Criteria API, illetve a Querydsl használatával végzem. A kettő hasonlóan működik: az entitásokhoz metamodel osztályokat generálnak, amikre hivatkozhatunk a lekérdezésben, és dinamikusan érhetjük el az entitások attribútumait. Pont a hasonlóságuk miatt döntöttem úgy, hogy mindkettőt kipróbálom, és összehasonlítom, hogy el tudjam dönteni, melyiket célszerűbb használni a projektben.

4.2.3. Az adatbázis

Az alkalmazás adatainak tárolásához a MySQL adatbázis-kezelőt választottam, főképp azért mert ingyenes, könnyű használni, és már volt vele tapasztalatom.

Az adatbázis létrehozása után be kell konfigurálni a projektben az adatbáziskapcsolatot. A Springnek köszönhetően ez nagyon egyszerűen elvégezhető, csak az application.yml fájlban kell beállítani néhány tulajdonságot.

Az adatbázis adatokkal való feltöltéséhez érdemes írni egy inicializáló szkriptet data.sql néven, amit a megfelelő könyvtárba való helyezése után (a projekt resources könyvtárába) a Spring automatikusan lefuttat az alkalmazás indításakor, amennyiben ezt szeretnénk. Ezt szintén az application.yml fájlban tudjuk beállítani egyetlen érték átírásával. Ez hasznos lehet a fejlesztés során, például ha hibás adatok kerülnek az adatbázisba, és nem szeretnénk azokat egyesével kijavítani, akkor nagyon gyorsan újrainicializálhatjuk az adatbázist.

4.3. Üzleti logikai réteg

4.4. Megjelenítés

Ábrák jegyzéke