XIX. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2016. május 19–22.

FestivApp

Általános szoftverrendszer rendezvények programjának böngészésére és menedzsmentjére

Szerzők:

Kiss Anna

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Magyarosi Botond

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

dr. Simon Károly, egyetemi adjunktus,

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Szilágyi Zoltán, szoftverfejlesztő,

Codespring

Kivonat

Napjainkban egyre gyakoribbak a több száz eseményt felölelő rendezvények, melyek programkínálatában eligazodni nehéz feladatnak bizonyulhat. Ezt hivatott megkönnyíteni a FestivApp alkalmazás, amely főleg abban mutat túl a hasonló applikációk kínálatán, hogy egyszerre több rendezvény programját tudja kezelni. A rendszert három fő komponens alkotja: a központi szerver, a mobil alkalmazás és a webes felület. A felhasználók a mobil alkalmazásban böngészhetik az aktuálisan kiválasztott rendezvény programját, a szervezők a webes felületen tudják felvezetni és módosítani rendezvényük adatait. Mindkét klienst a szerver szolgálja ki adatokkal.

A dolgozat a FestivApp projektet mutatja be. Megemlíti az architektúrával kapcsolatos fontosabb döntéseket, mindhárom komponens esetében kitér a megvalósítás részleteire, a felhasznált technológiákra, az igénybe vett eszközökre és módszerekre. Végül példákon keresztül mutatja be az alkalmazás működését.

Tartalomjegyzék

Bevezető

Napjainkban gyakoriak az olyan rendezvények, konferenciák vagy fesztiválok, ahol akár több száz programpont közül is lehet válogatni. Sok esetben a program már nehezen átlátható, mivel számos esemény zajlik egymással párhuzamosan, több helyszínen. Egy résztvevő számára a programfüzetben történő keresgélés kényelmetlenné és időigényessé válhat; ezenkívül a párhuzamosan zajló események megjelenítése a nyomtatott forma miatt csak lineárisan történhet, ami nehezen átlátható, könnyen előfordulhat, hogy emiatt a résztvevő lemarad számára fontos eseményről.

Egyes fesztiválok szervezői ezt felismerve olyan telefonos alkalmazásokat ajánlanak alternatívaként, amelyek megkönnyítik a program böngészését, és további információkat is nyújtanak a helyszínekről, előadókról. A szoftverfejlesztés viszont költséges, és nagyon sok rendezvény van, amely nem rendelkezik akkora költségvetéssel, hogy külön alkalmazást tudjon rendelni és kiadni. A FestivApp az ilyen rendezvények számára jelentene kézenfekvő megoldást.

Továbbá, az előzőekben említett specifikus mobilalkalmazások használatánál kényelmesebb, ha egy alkalmazáson belül több rendezvény programja is böngészhető és a megszokott funkcionalitások a megszokott módon érhetőek el minden rendezvény esetében. Ráadásul, ilyen módon új programokról, kapcsolódó hírekről is könnyebben értesülhet a felhasználó. Erre is megoldást jelenthet a FestivApp.

A FestivApp működését leginkább az határozza meg, hogy több rendezvényt tud egységesen kezelni. A felhasználók az alkalmazásban választhatják ki a fesztivált, melynek programját böngészni szeretnék, a szervezők pedig a webes adminisztrációs felületről módosíthatják a programot. Egy új rendezvény bevezetéséhez elégséges az adminisztrációs felület kezelését elsajátítani, programozói tudás nem szükséges. Ezáltal a FestivAppon keresztül olyan rendezvények is elérhetővé tehetik programajánlatukat, melyeknek egyébként nem lenne lehetőségük erre.

Az alkalmazás fejlesztése 2015 májusa óta zajlik; a Codespring Mentorprogram részeként indult, majd a Codespringnél folytatott szakmai gyakorlat, illetve a csoportos projekt tantárgy keretén belül folytatódott, és mindvégig csapatban zajlott. Kezdetben a jelen dolgozat szerzői dolgoztak a projekten, majd a csoportos projekt egyetemi tantárgy keretein belül csatlakozott Nemes Gábor, Szilágyi Csongor és Szilágyi Máté, akik egy egyetemi félév erejéig besegítettek a fejlesztésbe.

A rendszer egy-egy prototípusát már használták 2015-ben Tusványoson és a Kolozsvári Magyar Napokon. Természetesen, azóta az alkalmazás sokat fejlődött és általánossá vált. Új verziójának kiadása után elsőként a Szent György Napokon mutatkozott be és a közeljövőben

több rendezvényen használható lesz, többek között az Erdélyi Tudományos Diákköri Konferencián, Tusványoson és a Double Rise fesztiválon. A FestivAppot a Codespring csapata a Mobile World Congress kiállításon is bemutatta.

A dolgozat a projektet, a fejlesztési folyamatot, illetve a felhasznált eszközöket és technológiákat mutatja be. Az első fejezet megemlíti a projekt előzményeit, kitérve a Tusványosra és a Kolozsvári Magyar Napokra kiadott verziókra is. Ezután a FestivApp funkcionalitásait sorolja fel, majd megmutatja hogyan néz ki a rendszer architektúrája „felülnézetből". A második fejezet a szerver oldalon használt technológiákat és a megvalósítás részleteit mutatja be. Megemlíti a nagyobb kihívásokat, az implementálás nehézségeit, és bemutatja, hogyan is történt néhány korszerű tervezési minta, illetve szabvány integrációja a rendszerbe. A harmadik fejezet a web kliens megvalósítását tárgyalja. A negyedik fejezetben az Android alkalmazás megvalósításának ismertetése, a fejlesztésnél használt technológiák leírása, és ezek alkalmazására vonatkozó részletek találhatóak. Az ötödik fejezet ismerteti a fejlesztésben felhasznált eszközöket és módszereket. A hatodik fejezet példákon keresztül szemlélteti az Android alkalmazás és a webes felület működését. A dolgozat néhány következtetéssel és a továbbfejlesztési lehetőségek leírásával zárul.

1. A FestivApp projekt

A FestivApp fesztiválok és konferenciák résztvevői és szervezői számára készült. Olyan funkcionalitásokat nyújt, melyek megkönnyítik a program böngészését és az eligazodást egy rendezvény területén. A rendszer egységesen tud kezelni több rendezvényt: a felhasználók a telefonos alkalmazásban választhatják ki, hogy melyik fesztivál programját akarják böngészni, a szervezők az adminisztrációs felületen módosíthatják rendezvényük programját.

A projekt egy szerver modult, egy webes adminisztrációs felületet, illetve egy Android alkalmazást foglal magába.

1.1. Előzmények

2015 májusában a Bálványosi Nyári Szabadegyetem és Diáktábor szervezői felkeresték a Codespring céget, mivel szerettek volna egy mobil alkalmazást a rendezvény számára, melynek fejlesztését önkéntes diákok segítségével képzelték el. Ekkor kezdett el dolgozni a csapat az alkalmazáson, gyakorlatba ültetve a Codespring Mentorprogram képzésein tanultakat. Kezdetben a fejlesztést nehezítette és lassította, hogy új, addig ismeretlen technológiákkal kellett megbarátkozni.

Két hónap fejlesztés után a 26. Tusványosra elkészült egy kezdetleges alkalmazás, mely

alapvető funkcionalitásokat biztosított a fesztivál programjának böngészéséhez.

Az események napokra leosztva jelentek meg, kezdeti időpontjuk szerint rendezve. Lehetséges volt a helyszínek szerinti szűrés; ez Tusványos esetében azért volt célszerű, mert a helyszínek neve kategorizáló jellegű volt: jellemezte az ott zajló események témáját. Az alkalmazás lehetőséget nyújtott előadók nevére és esemény címére történő keresésre is; a találatok egy listában jelentek meg, időrendi sorrendben. Értesítések küldésére is lehetőség volt, ez főleg programváltozás esetén, illetve rövid közlemények szétküldésénél bizonyult hasznosnak.

A szerver modul REST API-n keresztül kommunikált a mobil kliensekkel; a szerverrel történő kommunikációra az adatok kezdeti lekérésénél, illetve a program frissítésekor volt szükség. A szerver biztosított egy kezdetleges adminisztrációs felületet, ahol lehetőség volt feltölteni a fesztivál programjának XML alapú reprezentációját. Az XML állományokat Tusványos szervezői biztosították, egy általunk meghatározott XSD szerint.

Az Android alkalmazást Tusványos ideje alatt körülbelül hatszáz felhasználó töltötte le, és az értékelések, visszajelzések is jók voltak.

Az alkalmazás a Kolozsvári Magyar Napok alkalmával is megjelent, a rendezvénynek megfelelő színvilággal és grafikus elemekkel. Újdonság volt, hogy a szűrés kategóriák szerint történt, mivel a helyszínek kevésbé voltak irányadóak. Az alkalmazás átalakításánál nagyobb kihívást jelentett a program feltöltése, ugyanis a Kolozsvári Magyar Napok szervezői nem biztosították a szükséges XML állományokat. Az adatokat a rendezvény weboldaláról egy Perl szkript töltötte le és alakította át XML formátummá. A szkript megírásánál nehézséget jelentett, hogy a programpontok megjelenítése nem volt következetes, több helyen különböztek az elválasztó írásjelek, illetve a fehér karakterek.

Az alkalmazás a Kolozsvári Magyar Napok után is jó értékeléseket kapott, több mint háromszázan töltötték le.

1.2. Funkcionalitások

Felismerve, hogy ugyanaz az alkalmazás kisebb módosításokkal két fesztivált is ki tudott szolgálni, a csapat úgy döntött, hogy továbbfejleszti és általánosítja a rendszert, felkészíti több rendezvény kezelésére. A FestivApp legfontosabb tulajdonsága emiatt jelenleg már az, hogy általános rendszerként működik, több fesztivál/konferencia számára biztosít egységes megjelenítési és kezelési felületet.

1.2.1. A mobil alkalmazás funkcionalitásai

Az Android alkalmazás biztosít minden olyan funkcionalitást, amelyet elődjei is, ezenkívül további, felhasználók által is hiányolt funkcionalitásokkal egészült ki. Az események napokra

lebontva jelennek meg külön oldalakon (tabokon), melyek között a felhasználó könnyen váltani tud; a megjelenítés időrendben történik, a nézet betöltésekor a listában mindig az éppen következő események tűnnek fel. A program szűrése kategóriák és helyszínek szerint is lehetséges. A felhasználó egy programpontról részletesebb információkat is olvashat, illetve ugyanitt kedvencnek választhatja az adott eseményt. A kedvencek külön listában jelennek meg, és kezdeti időpontjuk előtt félórával emlékeztető érkezik. Keresni lehet események címére, illetve előadók nevére. Részletes leírást a helyszínekről is lehet olvasni, és ugyanitt jelenik meg a fesztivál térképe, melyen az adott helyszín is be van jelölve.

Az említett funkcionalitások mindegyike elérhető internetkapcsolat nélkül is; egy nagyobb méretű rendezvény, fesztivál területén a világhálóhoz való hozzáférés sokszor problémás lehet, így a megfelelő működés biztosításához garantálni kell az offline működést is.

Internetkapcsolatra a rendezvények közötti váltásnál van szükség, mivel az alkalmazás ilyen esetben törli a programot a telefon memóriájából, ezután pedig az újonnan kiválasztott fesztivál programját tölti és menti le. Bár a későbbi frissítés (például programváltozás esetében), illetve üzenetfogadás szintén hálózati kommunikációt igényel, a többi funkcionalitás offline módban is működik. Kivételt képez még, ha a felhasználó navigációt kér adott helyszínhez, mivel ez a művelet a Google Maps segítségével történik, így szintén internetkapcsolat szükséges hozzá.

1.2.2. A webes felület funkcionalitásai

Az adminisztrációs felületre a szervezők és a rendszergazda tudnak belépni, innen jogosultságaiknak megfelelő tartalmakat érhetnek el.

A szervezők bejelentkezés után saját rendezvényeiket látják, ezek adatait tudják módosítani. A fesztiválhoz új eseményeket, helyszíneket, kategóriákat, előadókat tudnak hozzáadni, meglévő események adatait tudják törölni vagy frissíteni, a helyszínekhez térképet tudnak feltölteni, koordinátákat tudnak megadni. A műveleteket a megfelelő szövegdobozok és egyéb beviteli mezők segítségével végezhetik el. Ha módosításaikat mentik, az oldal egy kérést küld a szervernek, melynek válaszából kiderül, hogy a művelet sikeres volt-e. Sikertelen kérés esetén a felhasználó hibaüzenetet lát a weboldalon.

A rendszergazda a rendezvények adatait nem módosíthatja, csupán új rendezvényt tud létrehozni (események nélkül), szervezőket tud regisztrálni és meg tudja adni, hogy melyik szervező melyik rendezvényhez férhet hozzá. Ezt szintén a megfelelő beviteli mezők segítségével teheti meg, kéréseit az oldal a szerver fele továbbítja.

1.2.3. A szerver funkcionalitásai

A szerver egyetlen adatbázisban tárolja mindegyik rendezvény adatait, képes több fesztivál és konferencia egységes kezelésére. REST API-n keresztül kommunikál a kliensekkel, kiszol-

gálja a kéréseket, visszatéríti a lekérdezett adatokat, vagy végrehajtja az adatokkal kapcsolatos műveleteket, és hiba esetén megfelelő hibakódot térít vissza a válaszüzenet fejlécében.

A rendezvények adatait bármilyen felhasználó lekérdezheti, módosítani, törölni azonban csak a szervezők tudják. A felhasználók adataihoz csak a rendszergazda férhet hozzá, ő tud létrehozni fesztiválokat, regisztrálni szervezőket és jogokat rendelni hozzájuk. A hozzáférés korlátozását a szerver alkalmazás korszerű biztonsági megoldásokkal éri el.

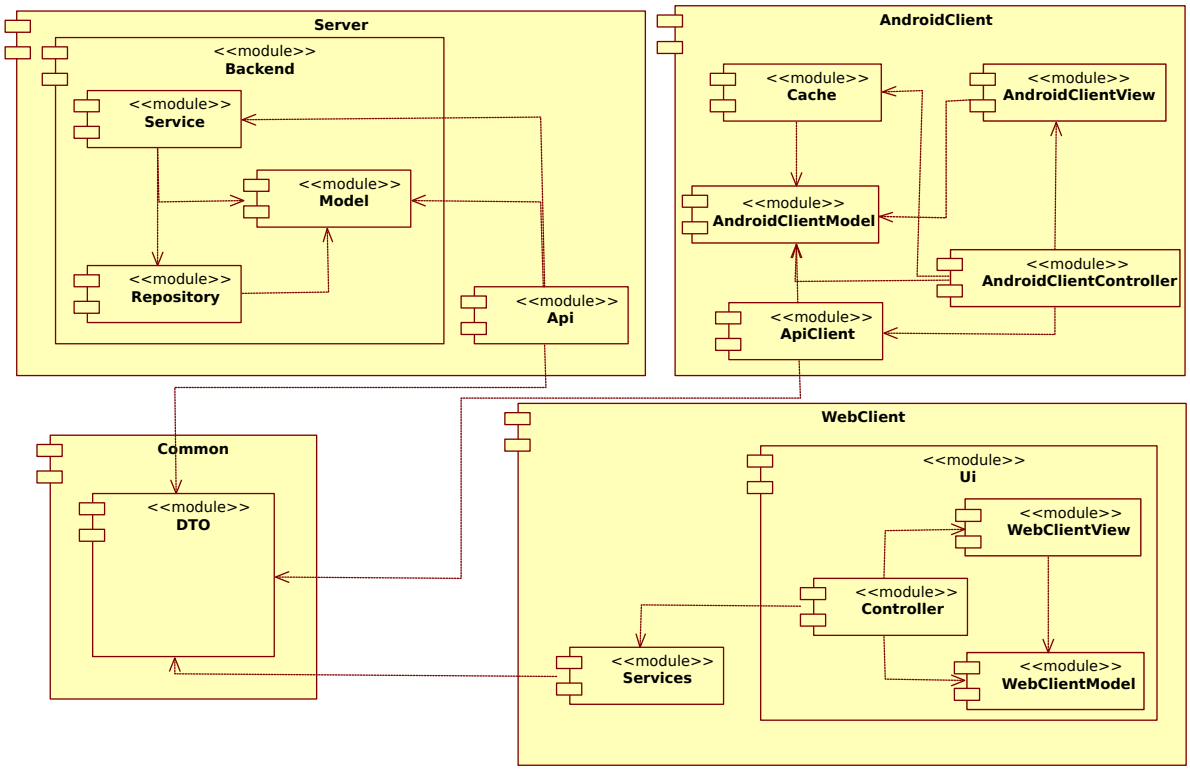
1.3. Architektúra

A rendszer fő komponensei: a szerver, az Android kliens és a webes felhasználói felület. Ezek kommunikációja a DTO tervezési minta alapján van megvalósítva; a közösen használt DTO osztályok a Common modulban helyezkednek el, így tulajdonképpen ez lesz a rendszer negyedik komponense (1. ábra).

A szerver komponens tartalmazza a backend-et és a REST API megvalósítását. A Backend felépítésében megfigyelhető a többrétegű architektúra: a Repository modul felel az adatbázissal való kapcsolattartásért és egyéb adatmanipulációval kapcsolatos műveletekért, a Service modulban található a központi alkalmazáslogika, az adatokat pedig modell osztályok reprezentálják, ezeket a Model modulban találhatjuk. Az API modul a Service modullal kommunikál, tőle kér le adatokat, hozzá továbbítja a kéréseket. A választ DTO objektumokká alakítja, majd JSON formátumban továbbítja a kliensnek HTTP protokollon keresztül.

Az Android alkalmazás felépítésében az MVC minta érvényesül: az AndroidClientModel modulban található osztályok reprezentálják az adatokat, az AndroidClientView modul osztályai, nézetei határozzák meg, hogy ezek az adatok milyen formában kerülnek a telefon képernyőjére, az AndroidClientController modulban pedig azok a kontrollerek találhatóak, melyek a vezérlésért felelnek. Az Android kliens az ApiClient modul segítségével küld kéréseket a szerver fele, ugyancsak ennek a segítségével fogadja a válaszokat és alakítja át objektumokká. A lekérdezett adatok memóriába való mentéséért a Cache modul felel.

A webes adminisztrációs felület az Android klienshez hasonlóan MVC minta alapján épül fel. Emellett még található itt egy Service modul, amely az API kérésekért és a válaszok fogadásáért felelős.



1. ábra. A rendszer komponensei

2. A szerver

A FestivApp rendszer legösszetettebb komponense a Java-ban írt szerver, melynek implementálása az új technológiák és a logikai összetettség miatt is kihívást jelentett. A fejezet architekturális rétegekre lebontva mutatja be a felhasznált technológiákat, kitér felhasználásuk módjára, leírja a komponens felépítését és megvalósítását.

2.1. Adatmodell

A rendszeren belüli központi entitásokat implementáló osztályok *Java Bean*-ek: privát adattagokkal, paraméterek nélküli publikus konstruktorral és getter-setter metódusokkal rendelkező osztályok, amelyek implementálják a `Serializable` interfészt. Mindegyik osztály az `AbstractModel` leszármazottja, amely egyetlen adattaggal, a `UUID`-val rendelkezik. Ez egy olyan azonosító, amely rendszerszinten egyedi minden objektumra nézve. A hierarchia következő lépcsőfokán a `BaseEntity` osztály áll, melyben szintén egyetlen adattag, az `id` található. Ez megegyezik az osztálynak megfelelő relációs adatbázis tábla elsődleges kulcsával.

Annak érdekében, hogy az adatforgalom csökkenjen a szerver és a kliens alkalmazások között, lehetséges csak azokat az adatokat lekérdezni, melyek adott dátum után változtak. Emiatt a szinkronizációs mechanizmus miatt fontos a `SynchronizableEntity` ősosztály:

`lastModified` mezője az utolsó módosítás dátumának megfelelő időpecsétet tárolja, `enabled` mezője pedig azt mutatja, hogy az objektum engedélyezve van-e. Az `enabled` mező `false` értékre való állítása a törlés műveletét hivatott helyettesíteni. Bevezetése azért volt szükséges, hogy az időpecsét alapján történő adatlekérés során a kliens a törlésekről is értesüljön.

A `Program` osztály egy példánya egy adott rendezvény attribútumait tárolja, például a kezdeti és végső időpontját, a rendezvény nevét, időzónáját. A Program alá sorolódik be minden más entitás, amely rendezvényre vonatkozó adatokat tárol (például események, előadók, helyszínek). A tartalmazási viszonyt azonban nem aggregáció fejezi ki, hanem a `tenant id` hordozza, mely megegyezik a Program osztály `id` mezőjével.

A legtöbb attribútumot tartalmazó osztály az `Event`, melyben egy adott esemény címe, kezdeti és végső dátuma, illetve leírása található, és aggregáció kapcsolja össze a `Location` osztállyal (1:1-hez viszony), a `Category` osztállyal (1:1-hez viszony) és a `Presenter` osztállyal (1:n-hez és 1:1-hez viszony). Az aggregált osztályok rendre az esemény helyszínére, kategóriájára, előadójára és moderátorára vonatkozó információkat tartalmaznak.

Az `Account` osztály is a fontosabb entitások közé sorolható, a felhasználók fontosabb adatait és azonosítóit tárolja. Adattagjai között szerepel a név, a jelszó, a bejelentkezés módja (a FestivApp alkalmazás szerverén vagy valamelyik közösségi hálón keresztül történt) és egy azonosító, amely a bejelentkezés módjának megfelelően vagy az e-mail cím, vagy a közösségi oldal által a felhasználónak megfeleltetett azonosító.

2.2. A komponensek menedzsmentje és a projekt konfigurálása

A szerver modul fejlesztésénél használt legfontosabb technológia a Spring. A Spring egy pehelysúlyú keretrendszer, amely megkönnyíti Java alkalmazások fejlesztését. A pehelysúlyú (lightweight) jelleg a Spring esetében a keretrendszer alapelvére utal, amely a minimális kihatást fogalmazza meg: azt célozza, hogy bármilyen alkalmazás mögé könnyen be lehessen építeni a Spring Core-t (minden hasznos funkcionalitásával együtt), és különösebb költségek nélkül el is lehessen távolítani [1].

A Spring keretrendszer magját az IoC konténer képezi. A komponensek életciklusát a keretrendszer menedzseli, és biztosítja, hogy egy adott komponens futási időben hozzáférjen függőségeihez és interakcióba lépjen velük. Ezt a Spring a Java reflection mechanizmusának segítségével éri el, a komponensek függőségeinek kezelésére pedig *dependency injection*-t (DI) használ. A Spring DI mechanizmusának középpontjában a `BeanFactory` interfész található, mely az IoC konténer által felügyelt komponensek (a *Spring bean*-ek) létrehozásáért felelős.

A függőségek injektálására vonatkozó beállításokat többféleképpen adhatjuk meg, például XML konfigurációs állományokban vagy annotációk segítségével. A FestivApp projekt szerver

komponense az utóbbi megoldást alkalmazza.

A Spring Boot keretrendszer megkönnyíti Spring alapú alkalmazások létrehozását és konfigurálását. Mindezt úgy éri el, hogy a kezdeti beállításokat nem bízza a fejlesztőre, hanem hasonló alkalmazásokra jellemző értékeket és tulajdonságokat vesz alapértelmezettnek, így nagyon gyorsan és egyszerűen létrehozható az alkalmazásnak egy kezdeti működő verziója. A Spring Boot a további fejlesztést sem hátráltatja, a konfigurációk könnyen módosíthatóak [2].

2.3. Az adathozzáférési réteg

Az adathozzáférési réteg feladata az adatbázissal való kapcsolattartás. Különböző adatbázisműveletek végrehajtását kéri, illetve fogadja és feldolgozza a lekérdezések eredményeit. A FestivApp szervere MySQL adatbázist használ, amellyel az adathozzáférési réteg a *JDBC API*-n keresztül lép kapcsolatba.

A perzisztencia megvalósításához a Spring keretrendszer kézenfekvő megoldásokat nyújt, így az adathozzáférési réteg implementálása gyorssá és egyszerűvé válik.

2.3.1. A Spring Data Access/Integration modul

A Spring olyan Data Access Object (DAO) támogatást nyújt, mely segítségével egységes módon kezelhetőek és könnyen cserélhetőek a különböző adathozzáférést biztosító technológiák és egyéb keretrendszerek.

A Spring lehetővé teszi, hogy különböző adatforrások kezelése egységesen történjen, és egy alkalmazáson belül több adatforrás is használható legyen. A FestivApp esetében egyelőre csak MySQL adatbázis van használva, de a továbbfejlesztés során indokolttá válhat MongoDB, vagy más NoSQL rendszer bevezetése. A Spring segítségével ez majd gyorsan és gördülékenyen történhet.

A relációs adatbázisokkal történő hatékony kommunikáció érdekében a Spring nem csak integrálja a *JDBC API*-t, hanem absztrakciót is biztosít hozzá, mely megkíméli a programozót az alacsony szintű részletek implementálásától: csak a kapcsolat paramétereit kell meghatározni, a lekérdezéseket megírni, és a lekérdezés paramétereit deklarálni.

A Spring keretrendszerre épülő alkalmazásokba könnyen integrálható a Java Persistence API (JPA), amely a JDBC feletti absztrakciós szintet képez és célja az Object Relational Mapping (ORM) programozási modell szabványosítása a Java alkalmazások esetében.

A Spring Data Access modul segítségével a kivételkezelés szintén egységesen, bevált recept szerint történhet. A Spring az ORM keretrendszer, illetve a JDBC által dobott specifikus kivételeket naplózza és DataAccessException objektumokba „csomagolja”. Így lehetővé válik, hogy ezeknek a rétegspecifikus kivételeknek a kezelése a szolgáltatási rétegen belül történjék [1].

2.3.2. A Spring Data keretrendszer

A Spring Data keretrendszer célja megkönnyíteni az adathozzáférési réteg implementálását. Tulajdonképpen több keretrendszert fog egybe, amelyek különböző típusú adatbáziskezelő rendszerekkel együtt alkalmazhatóak. A relációs adatbázisok esetében a Spring Data JPA használható, amely a JPA feletti absztrakciós szintként működik. A programozónak nem kell implementálnia a DAO osztályokat, csupán ezek interfészeit.

A Spring Data legfontosabb interfésze a `Repository` interfész, amelyből minden más adathozzáférési műveletet deklaráló interfész származik. Ilyen például a `CrudRepository`, mely összetettebb CRUD műveletek (create, read, update, delete) megadására nyújt lehetőséget. Így például egy lekérdezés megírásához elegendő egy metódust deklarálni, ebből a keretrendszer fel tudja építeni a megfelelő adatbázis-oldali parancsot [3].

A FestivApp adathozzáférési rétegében található `EventRepository` interfész metódusai jól példázzák, hogy a fent említett módon összetettebb lekérdezések is tömören és könnyen megadhatóak. Például a

```
List<Event> findEventsByStartDateBetweenAndLocationIdOrderByStartDateAsc(
  Date date1, Date date2, Long locationId);
```

metódus olyan eseményeket fog visszatéríteni, melyek a paraméterként megadott két dátum között kezdődnek, és azon a helyszínen zajlanak, melyet a szintén paraméterként megadott `locationId` azonosít; a visszatérített listában az események kezdeti dátumuk szerint növekvő sorrendbe lesznek rendezve.

Ha a lekérdezés túl bonyolult ahhoz, hogy a metódus deklarációjánál alkalmazott elnevezési konvenciók alapján kényelmesen létre lehessen hozni, a metódusokon alkalmazható annotációk segítségével JPQL query-k is használhatóak.

A Repository interfészek mögött a háttérben a JPA `EntityManager-e` található, amely szinkronizálja a perzisztencia környezethez tartozó beanek állapotát az adatbázisba mentett állapottal. Természetesen, specifikusabb esetekben arra is lehetőség van, hogy saját implementációkat hozzunk létre a repository interfészeknek, és így például a JPA Criteria API segítségével dinamikusan felépített, vagy akár natív lekérdezések is alkalmazhatóak.

A FestivApp szerver adathozzáférési rétegében a modell osztályoknak megfelelő DAO interfészek jelennek meg, a fent említett `EventRepository` mellett, többek között, a `CategoryRepository`, a `LocationRepository`, a `PresenterRepository` és a `ProgramRepository`. Ezek mindegyike olyan metódusokat deklarál, melyek a központi entitásoknak megfelelő adatbázistáblákon hajtanak végre műveleteket. A repository komponensek hierarchiájának köszönhetően az alapműveleteknek megfelelő metódusokat (`findAll`, `findOne`, `save`, `delete`) nem szükséges az általunk származtatott `Repository` interfészekben deklarálni.

Meghívásukhoz elegendő, ha létezik a `CrudRepository`-nak olyan példánya, mely a megfelelő sablonparaméterekkel jött létre. A sablonparaméterek az adatbázis táblának megfelelő modell osztály nevét és az elsődleges kulcs típusát jelentik.

2.3.3. EclipseLink és multitenancy

A FestivApp projekt különböző rendezvények adatait egységesen kezeli. Ez azt jelenti, hogy az adatok szerkezetének felépítése, tárolása, lekérdezése azonos módon történik. Ez a *multitenancy* mechanizmus által válik lehetségessé.

A multitenancy-t főleg abban az esetben szokták bevezetni, amikor egy adott szoftver több különböző felhasználócsoportot szolgál ki ugyanolyan szerkezetű adatokkal, de ezeket az adatokat elkülönítve, a felhasználók jogosultságainak megfelelően kell kezelni és megjeleníteni. Például, egy rendszer egy adatbázisban tárolja több cég adatait, de minden felhasználó csak saját cégének adataihoz fér hozzá. Esetünkben a multitenancy használata azért indokolt, mert az adatok hierarchiájának csúcsán elhelyezkedő entitások, vagyis a rendezvények ugyanígy szétválasztják az alájuk besorolt adatokat. A szervezők csak saját rendezvényeik adatait menedzselhetik, a felhasználók pedig rendezvényenként különválasztva láthatják az adatokat.

A multitenancy segítségével kiküszöbölhető több azonos szerkezetű adatbázis vagy adatbázis tábla használata. A hasonló szerkezetű adatok ugyanabban a táblában tárolódnak, és egy specifikus azonosító (*tenant id*) különbözteti meg őket. Multitenancy-t támogató ORM keretrendszer használata esetén a lekérdezések egyszerűek maradnak, mivel a tenant id-ra vonatkozó megszorítás nem kerül bele a keretrendszer szintjén megírt parancsba; elégséges megadni csupán egyszer, az adatbázis-műveletek elvégzése előtt.

A JPA referencia implementációjának számító EclipseLink keretrendszer nagyon jó multitenancy támogatással rendelkezik, ezért a FestivApp ezt használja a Spring Boot által alapértelmezettnek tekintett Hibernate helyett.

A multitenancy megvalósításához biztosítani kell, hogy minden adatbázis-művelet előtt be legyen állítva a tenant id értéke. A tenant id esetünkben megfelel a rendezvény azonosítójának (a `programId`-nak), ami minden rendezvényhez köthető API kérés paraméterlistájában szerepel. A programId értékét az API modulban elmentjük egy kérés hatókörű (request scope) beanbe. Mikor az API továbbítja a kérést a szolgáltatási rétegnek, a `programId` már nem szerepel a metódusok paraméterlistájában. Ezután a szolgáltatási réteg az adathozzáférési réteg metódusait hívja, és ezek előtt a metódushívások előtt már be kell állítani a tenant id értékét. Ez egy aspektus segítségével valósul meg.

2.3.4. Spring AOP

Az aspektusorientált programozás az objektumorientáltságtól eltérő gondolkodásmódot feltételez, de nem ellent mond ennek, hanem kiegészíti ezt. Az AOP legfontosabb eleme az aspektus, melynek segítségével átmetsző követelményeket reprezentálhatunk, így biztosítva a nagyobb fokú modularizációt.

A Spring AOP implementációja nem támogatja az AOP paradigma által bevezetett lehetőségek mindegyikét, csak a fontosabbakat. Az AspectJ nyelv viszont teljes mértékben integrálható a Springgel, így elérhetővé válik minden AOP funkcionalitás.

Egy aspektus írásánál azonosítani kell a join point-ot (az a programpont, ahová az aspektus kódrészletet szúr be), meg kell határozni egy pontszűrőt (pointcut, a kifejezés, mely kiválasztja, azonosítja a join point-ot), és definiálni kell egy advice-ot (a kódrészlet, mely végrehajtódik, ha a program eléri a join point-ot, illetve a kódrészlet végrehajtására vonatkozó szabályok) [1].

Ahhoz az aspektushoz, ami a tenant id értékét állítja be (`TenancyResolverAspect`), olyan pontszűrő rendelődik, amely minden adathozzáférési réteg fele érkező metódushívást azonosít. Az aspektus tehát még a DAO metódusokba való belépés előtt végrehajtódik. Az advice kiveszi a kérés hatókörű beanből a korábban beállított `programId`-t, és értékét beállítja tenant id-nak.

2.4. A szolgáltatási réteg

A szolgáltatási réteg tartalmazza az alkalmazás központi logikáját. A beérkező kérések paramétereit ellenőrzi, további feldolgozást végez rajtuk, majd, ha szükséges, meghívja az adathozzáférési réteg metódusait. Az innen érkező választ továbbítja az őt hívó komponensnek, vagy a célnak megfelelően dolgozza fel. Naplózza és kezeli a kivételeket, réteg specifikus kivételt dob tovább (`ServiceException`).

A FestivApp esetében az alkalmazás központi logikája `@Service` annotációval ellátott Spring beanekben található. A cserélhetőség érdekében Service interfészek vannak létrehozva, melyeknek implementációi ServiceBean osztályokban találhatóak. Ezáltal az is biztosított, hogy a felsőbb rétegekben injektálni tudjuk a szolgáltatási réteg komponenseit (az injektálás ugyanis nem implementációk, hanem interfészek szerint történik).

A Service osztályok egyszerű hierarchiába rendeződnek. Ennek csúcsán a `BaseService` áll, ebben azok a metódusok jelennek meg, melyeket várhatóan minden Service osztály implementálna (`findOne`, `findAll`, `save`, `delete`). A `BaseService` egy `BaseEntity` típusú sablonparaméterrel rendelkezik, a `BaseServiceBean` osztály ennek a paraméternek megfelelő `CrudRepository` példányt injektál, és definiálja a fent említett négy metódust.

A `BaseService`-ből modell osztályoknak megfelelő Service osztályok származnak, melyek már specifikusabb műveleteket tartalmaznak. A Service osztályok közül fontos funkciója van

az AccountService-nek, amely a felhasználókkal kapcsolatos műveleteket tartalmazza, például a regisztrációért felelős registerNewUserAccount metódust. Ez egy Account objektumot kap paraméterként, ellenőrzi, hogy létezik-e már ugyanazzal az azonosítóval regisztrált személy, hogyha nem, akkor a jelszót az SHA-1 algoritmus segítségével hash-eli, és meghívja az AccountRepository save metódusát, paraméterként a módosított Account objektumot adva át.

Kiemelhető a NotificationService is, mely a Google *Cloud Messaging* szolgáltatását veszi igénybe, és értesítéseket (*push notification*) küld azokra az eszközökre, melyekre telepítve van a FestivApp Android alkalmazás.

2.5. API

A FestivApp szervere RESTful webszolgáltatásokon keresztül publikálja szolgáltatásait. A REST (Representational State Transfer) olyan programozási modell, melyet HTTP protokollon keresztül történő, állapot nélküli kommunikációra terveztek. Az adatok és műveletek erőforrásokként vannak kezelve, URI-k azonosítják őket, és manipulációjuk HTTP analógia alapján néhány egyszerű művelet (PUT, GET, POST, DELETE) segítségével valósítható meg. A küldött, illetve fogadott üzenetek típusát a szabvány nem szögezi le, a leggyakoribb az XML, illetve a JSON formátum használata. A FestivApp szerver JSON formátumban fogadja és küldi üzeneteit.

A REST API implementálásához a Spring Web MVC keretrendszer nyújtott segítséget.

2.5.1. A Spring Web MVC keretrendszer

A Spring Web MVC (model-view-controller) keretrendszere MVC mintára épülő webalkalmazások fejlesztését támogatja. Középpontjában a DispatchServlet áll, mely a Front Controller tervezési minta alapján épül fel: fogadja és a megfelelő handlernek továbbítja a HTTP kéréseket. A handlerek a @Controller és a @RequestMapping annotációk segítségével állíthatóak be. Ez a mechanizmus lehetővé teszi RESTful alkalmazások fejlesztését is [4].

A FestivApp esetében a különböző erőforrásoknak Resource osztályok felelnek meg. Ezekben az osztályokban az említett módon a @RequestMapping annotáció köti össze a kérést a handlerrel, vagyis meghatározza, hogy adott endpoint-ra érkező, adott HTTP művelet esetén milyen metódus fusson le. Például az EventResource osztály findEvent metódusa az /api/programs/{programId}/events/{eventId} URI-ra érkező GET kérés esetén fog lefutni. A {programId} és az {eventId} URI paraméterek, ún. PathVariable-ök helyét jelölik. Ezek a paraméterek be kell kerüljenek a metódus paraméterlistájába is. Összegezve, a felannotált findEvent metódus fejléce az alábbi módon néz ki:

```
@RequestMapping(value = "/api/programs/{programId}/events/{eventId}",
method = RequestMethod.GET)
public EventDTO findEvent(@PathVariable final String programId,
                          @PathVariable final Long eventId);
```

A Resource osztályok olyan metódusokat tartalmaznak, melyek handlerei a REST szabványnak megfelelő URI-knak. Az `/api/programs/{programId}/events` endpointra érkező GET kérést a `getEvents` metódus szolgálja ki, és a válaszüzenetben egy lista érkezik az összes eseménnyel. Az `/api/programs/{programId}/events` URI-ra érkező POST művelet azt kéri a rendszertől, hogy a kérés törzsében megjelenő új entitást szúrja be a paraméterként megadott rendezvény programjába. Ha GET kérés érkezik az `/api/programs/{programId}/events/{eventId}` endpointra, akkor a válaszüzenet az `eventId`-val azonosított eseményt fogja tartalmazni. Ha ugyanerre az endpointra PUT kérés érkezik, akkor az `eventId`-val azonosított esemény adatai frissítve lesznek a kérés törzsében érkező adatok szerint. Ha a kérés típusa DELETE, akkor az azonosított entitás törölve lesz.

Ezeket a műveleteket a Resource osztályok úgy végzik el, hogy a kérést továbbítják a szolgáltatási réteg megfelelő Service-ei felé. A Service-ek, a Spring IoC segítségével, interfészeik szerint vannak injektálva, így biztosítva, hogy a két réteg között ne legyen szoros függőségi viszony.

A Resource osztályok közül a legtöbb az alkalmazás központi entitásaihoz köthető: `ProgramResource`, `EventResource`, `CategoryResource`, `PresenterResource`, `LocationResource`, `AccountResource`. Ezek mindegyike a fent leírt minta szerint van implementálva. Az API réteghez tartozik még a `FavouritResource`, mely a felhasználók által kedvencnek jelölt események lementésére és lekérdezésére biztosít két (POST, illetve GET parancsnak megfelelő) metódust, illetve a `SocialResource`, melynek `socialLogin` metódusa közösségi hálók segítségével történő bejelentkezésre biztosít lehetőséget.

Mivel a REST API a HTTP protokollt használja, a HTTP állapotkódok (status codes) megfelelő beállítása is követelmény. A Spring Web MVC alapértelmezetten 200-as (OK) kódot állít be, ha az adott művelet sikeres volt. Ez a FestivApp esetében annyiban változott, hogy POST kérés esetén a rendszer 201-es (Created) kódot térít vissza, és a válasz fejlécében a Location változónak beállítja az újonnan létrejött erőforrás elérési helyét. Hiba esetén (nem kezelt kivétel) a Spring Web MVC szintén gondoskodik alapértelmezett kód beállításáról, ez pedig az 500-as (Internal Server Error). A FestivApp ismeri a 400-as (Bad Request), 401-es (Unauthorized), 403-as (Forbidden), és a 404-es (Not Found) kódokat is. Az új kódok bevezetése a Spring Web MVC kivételkezelési mechanizmusának segítségével történt. Olyan kivétel osztályok deklarálására volt szükség, melyek a `@ResponseStatus` annotációval rendelkeznek, és ennek paraméterként a kivételnek megfelelő hibakódot adják át. Például a BadRequestException

a `@ResponseStatus(HttpStatus.BAD_REQUEST)` kifejezéssel van annotálva. A Resource osztályokban hiba esetén a hibának megfelelő kivétel dobódik, a Spring pedig a válasz felépítésekor a háttérben beállítja a hibakódot.

2.5.2. A Data Transfer Object tervezési minta alkalmazása

A FestivApp projekten belül a *Data Transfer Object*-eknek (DTO) [5] külön modul van létrehozva, festivapp-common néven. Ez a modul függősége a szerver modulnak és mindkét kliens modulnak (Android és web), így mindhárom ugyanazokat a DTO-kat használja.

Mivel az API által implementált műveletek nagy része az alkalmazás központi entitásaival kapcsolatos, a DTO-k is ezeknek az entitásoknak felelnek meg. A modell osztályok DTO-kká alakítását Assembler osztályok végzik. Az Assembler osztályok mindegyike három metódust definiál: `createDTO`, `createModel` és `updateModel`. A `createDTO` egy modell objektumot kap paraméterként, és egy DTO-t térít vissza. A `createModel` egy DTO-t kap paraméterként és egy modell objektumot térít vissza. Az `updateModel` két értéket kap paraméterként: egy modell objektumot és egy DTO-t. Feladata az, hogy a DTO-ban tárolt adatok szerint frissítse a modell objektumot. Az `updateModel` metódus bevezetése a PUT típusú kérések miatt volt hasznos: a PUT kérés handler metódusában első lépésként lekérdezzük a paraméterként kapott azonosítóval rendelkező entitást, ezt frissítjük a kérés törzsében érkező DTO szerint, majd lementjük az objektumot.

A POST és a PUT kérések törzsében közlekedő DTO-k különböznek a GET kérések válaszaiban megjelenő DTO-któl. Az `EventDTO` esetében például a GET kérés visszatéríti a tartalmazott `LocationDTO`-t, `CategoryDTO`-t és a `PresenterDTO`-k listáját is. Ez ebben az esetben pont a tervezési minta egyik alapvető célját szolgálja, vagyis megkíméli a klienst további metódushívásoktól. A POST és a PUT törzsében azonban nincs értelme a tartalmazott DTO-kat is elküldeni. Az entitások, amelyek ezeknek a DTO-knak felelnek meg, már jelen vannak az adatbázisban, ezért az egyszerűsített EventDTO csak azonosítójukat tartalmazza.

2.6. Biztonság

Webszolgáltatások esetében gyakran előfordul, hogy bizonyos erőforrásokhoz korlátozni kell a hozzáférést. A FestivApp esetében ilyen erőforrások például a felhasználók adatai és az ezekkel kapcsolatos műveletek, a kedvencnek jelölt események kezelése, az adminisztrátor és a moderátor jogosultságú felhasználók által végezhető műveletek stb. Az erőforrások egy RESTful API-n keresztül érhetőek el, a REST pedig egy állapotmentes kommunikációs modell, ezért olyan biztonsági megoldásokra van szükség, melyek megtartják az állapotmentességet.

2.6.1. Spring Security

A Spring Security keretrendszer átfogó biztonsági megoldásokat nyújt bármilyen Java alapú enterprise alkalmazás számára, leginkább azonban a Spring keretrendszerrel fejlesztett alkalmazásokat támogatja.

A Spring Security keretrendszerben a felhasználók hitelesítése az `AuthenticationManager` interfészen keresztül történik. Ennek az alapértelmezett implementációja a `ProviderManager` osztály, amely a kérést elküldi minden `AuthenticationProvider`-nek, melyhez a hitelesítés során a rendszernek kapcsolódnia kell. Az `AuthenticationProvider`-ek leírják a hitelesítés minden lehetséges forgatókönyvét. A FestivApp szerver komponensének esetében használt `DaoAuthenticationProvider` a legegyszerűbb implementáció. `UserDetails` objektumokban tárolja a felhasználók adatait, ezeket a `UserDetailsService` segítségével kérdezi le az adatbázistól, és összehasonlítja a felhasználó által megadott adatokkal. Ha a hitelesítés sikeres, akkor a `SecurityContextHolder` objektumba egy teljesen inicializált `Authentication` objektum kerül [6].

A FestivApp esetében az autorizáció (engedélyezés) a Spring Security keretrendszer által támogatott *OAuth2* standard szerint történik.

2.6.2. OAuth2

Az OAuth2 célja HTTP szolgáltatásokhoz való hozzáférés korlátozása. Működésében négy fontos szerep különíthető el: az erőforrás birtokosa, az erőforrás szerver, a felhasználó és az autorizációs szerver. Az erőforrás birtokosa egy olyan entitás (nem feltétlenül személy), amely képes engedélyezni a hozzáférést egy védett erőforráshoz. Az erőforrás szerver tárolja és szolgálja ki a védett erőforrásokat. A felhasználó egy olyan alkalmazás, amely az erőforrás birtokosának jóváhagyásával kérést küld az erőforrás szerver fele. Az autorizációs szerver feladata azoknak az access tokeneknek a kibocsátása, melyekkel a felhasználó igazolni tudja jogosultságát [7, 8].

Az OAuth2 standard négyféle engedélyezési típust kínál, ezek közül a FestivApp a jelszó alapú engedélyezést használja, amely lehetővé teszi, hogy a kérésben érkezett felhasználónevet és jelszót (a hitelesítés után) az autorizációs szerver access tokenre váltsa, és a válaszban ezt elküldje a felhasználónak. Ezután a felhasználó minden kérésének fejlécében beállítja ezt az access tokent, így igazolni tudja jogosultságát. Az access token viszont csak adott ideig érvényes, és miután lejár, a felhasználónak egy külön kérést kell intéznie a szerverhez, hogy új tokent kapjon. Ezt egy refresh token segítségével teheti meg, melyet a szerver az első válaszüzenetében küld az első access tokennel együtt. A token minden kérés fejlécében megjelenik, ez teszi lehetővé az állapotmentes kommunikációt.

2.6.3. JWT

A jogosultság igazolására több típusú token is használható, a FestivApp által generált tokenek típusa az IETF standardként is elfogadott JWT (*JSON Web Token*). A JWT-ben egy JSON struktúrájú objektum van kódolva, mely a felhasználóra vonatkozó adatokat tartalmazza (felhasználónév, jogosultság). A JWT használata biztonságos, mivel a benne foglalt információ digitális aláírással van ellátva. A digitális aláírást esetünkben a HMAC algoritmus végzi egy titkos kulcs alapján, melyet csak a feladó (az autorizációs szerver) ismer.

A JWT a kompakt forma és a bennfoglaltság (*self-containment*) miatt válik hatékony adatátviteli módszerré. A kompakt forma a JWT kis méretére vonatkozik, mely lehetővé teszi, hogy a token könnyen továbbítható legyen URL-en keresztül, HTTP üzenet POST paramétereként vagy a HTTP fejléc részeként. A bennfoglaltság arra utal, hogy a JWT-ben tárolt adatok minden szükséges információt tartalmaznak a felhasználó beazonosítására, így egy kérés érkezésekor az adatbázist legfeljebb egy alkalommal szükséges lekérdezni [9, 10].

2.7. Közösségi hálók integrációja

A közösségi hálók korában elvárásnak minősül, hogy egy alkalmazás integrálja a népszerű közösségi oldalak szolgáltatásait. A FestivApp mobil alkalmazás Facebook, illetve Google alapú bejelentkezést támogat. A mechanizmus szerver oldali implementálásában a Spring Social projekt segített.

A Spring Social projekt célja megkönnyíteni a Spring keretrendszerre épülő alkalmazások számára a *Software as a Service* (SaaS) rendszerekhez való kapcsolódást úgy, hogy adott felhasználót megtestesítve intéz kéréseket a szolgáltató API-jához. A kapcsolatot egy `Connection` objektum reprezentálja, amelyet a szolgáltatóhoz (Facebook, Google) társított `ConnectionFactory` hoz létre a felhasználó access token-jének ismeretében. Ezt az access tokent a felhasználónak a szolgáltatótól kell kérnie [11].

A FestivApp rendszer esetében az alternatív bejelentkezési módot (Facebookon, Google-en keresztül) választó felhasználó a szolgáltatóhoz jelentkezik be először, a szolgáltató által biztosított felületen. Sikeres bejelentkezés után egy access tokent kap vissza, melyet a FestivApp szerveréhez továbbít. Itt ellenőrzésre kerül az access token hitelessége, és a felhasználóhoz a FestivApp autorizációs szervere által generált access token érkezik a válasz üzenetben. Ezt a továbbiakban ugyanúgy használhatja, mint a hagyományos bejelentkezés útján szerzett tokent.

3. Az Android kliens alkalmazás

Az Android napjainkban a legelterjedtebb mobil eszközre telepíthető operációs rendszer, ezért született az a döntés, hogy a FestivApp mobil kliens alkalmazás fejlesztése először erre a platformra történjen. Az Android operációs rendszer nyílt forráskódú, ennek is köszönhető, hogy nagyon sokan járulnak hozzá fejlesztéséhez, például rendszerhibák kijavításával, különböző könyvtárak létrehozásával, vagy Android platformra történő fejlesztést segítő technológiák fejlesztésével.

Az Android platform mobil eszközökre optimalizált Linux kernelre épülő operációs rendszer. Az Androidon futó alkalmazások fejlesztése az *Android Software Development Kit* (SDK) segítségével történik [12]. A visszafele kompatibilitás biztosítása az *Android Support Library* segítségével valósul meg. Ez a könyvtár lehetővé teszi, hogy az újabb Android API-k által nyújtott funkcionalitások elérhetőek legyenek régebbi verziókat támogató eszközökön is. A felhasználói réteg szélesebb körű lefedettségének biztosítása mellett az Android Support Library a kinézetre és a teljesítmény javítására vonatkozó megoldásokat is nyújt [13].

3.1. Az adatmodell

Az Android kliensen belül alkalmazott adatmodell hasonlít a szerver komponens adatmodelljéhez. Mindegyik osztály az `AbstractModel`-ből származik, emellett az ősosztály mellett pedig megjelenik a `SynchronizableEntity`, melynek `enabled` mezője lehetővé teszi, hogy az időpecsét alapján történő adatlekérés során a kliens a szerveren történt törlésekről is értesüljön. Ha értéke `false`, akkor törölni kell az objektumnak megfelelő rekordot az adatbázisból. Az `id` mezőt minden osztály a `BaseEntity`-től örökli. A központi entitások a szerver komponenshez hasonlóan a `Program`, az `Event`, a `Category`, a `Presenter` és a `Location`. Az `Event` osztályban a programra vonatkozó adatok mellett találunk még egy logikai típusú `favourite` mezőt, melynek értéke igaz, ha az esemény kedvencnek van jelölve, illetve egy `asciiTitle` nevű, String típusú mező is megjelenik. Ebben a mezőben a cím ékezetek nélküli formája található. Erre a keresési funkció miatt volt szükség: azt biztosítja, hogy a keresés ne függjön a keresett szövegben megjelenő ékezetektől. A `Notification` osztály értesítésekre (push notification) vonatkozó információkat tartalmaz (tartalom és dátum), a `Style` osztály pedig egy adott rendezvény brandjének megfelelő színeket tárol.

3.2. Az adathozzáférési réteg

Az adathozzáférési réteg implementálása a cserélhetőséget szem előtt tartva történt: az adatok manipulációjával kapcsolatos metódusok deklarációja Repository interfészekben található,

ezeknek implementációi technológia-specifikusak (*OrmLite*).

Az OrmLite egy pehelysúlyú keretrendszer Java objektumok relációs adatbázisba történő perzisztálásához. Többfajta adatbázist támogat, többek között az Android eszközökön futó SQLite-ot is, ehhez azonban szükség van arra, hogy natív kéréseket is küldjön az Android operációs rendszer API-jaihoz. A perzisztenciára vonatkozó információkat más ORM keretrendszerekhez hasonlóan itt is annotációk segítségével adhatjuk meg: a `@DatabaseField` annotáció jelzi, hogy az adott mező egy adatbázisoszlopnak felel meg, az annotáció paramétereivel pedig megadhatjuk az oszlop nevét, típusát és különböző megszorításokat [14].

Az adathozzáférési rétegen belül a legfontosabb feladatot a `DatabaseHelper` osztály látja el. Factory-nak is tekinthető, mivel fő feladata a központi entitásoknak megfelelő DAO objektumok létrehozása és visszatérítése úgy, hogy közben azt is ellenőrzi, hogy mindegyikből egy példány létezzen. A Repository osztályok konstruktorukban elkérnek a `DatabaseHelper`-től egy adott entitásra vonatkozó DAO példányt, a metódusaikban pedig ezt felhasználva végzik el az adatbázis-műveleteket. A DAO osztályok `SQLException`-t dobhatnak. Ezek naplózása után az érintett Repository egy réteg-specifikus `RepositoryException`-t dob tovább a felsőbb rétegek fele.

3.3. Az API kliens

Az Android alkalmazás a REST API-n keresztül lép kapcsolatba a szerverrel, ezért a hálózati kommunikációért felelős az API kliens, amely a központi entitásoknak megfelelő API interfészeket, és az ezeket implementáló osztályokat tartalmazza. Az API hívásokat, valamint az üzenetek parse-olását a *Retrofit* könyvtár végzi az *OkHttp* nevű HTTP klienssel együtt.

A Retrofit könyvtár megkönnyíti az API hívások felépítését, küldését és fogadását. Ezenkívül arról is gondoskodik, hogy az üzenet törzsében küldött objektumokat a megfelelő (előre meghatározott) formátumúra alakítsa (a FestivApp esetében ez a JSON), illetve a válaszok esetében fordított irányba is elvégezze az átalakítást [15]. Ahhoz, hogy mindez megvalósuljon elégséges csupán egy interfészt deklarálni, melyben annotációk segítségével lehet megadni a szükséges információkat: az erőforrások elérési helyét és a kérés típusát. A metódus deklarálásánál a fejléc szerkezetével lehet jelezni, hogy a kérés szinkron vagy aszinkron. Utóbbi esetben a paraméterek között egy `Callback` típusú objektumnak is jelen kell lennie. Például, a

```
@GET("/api/programs/{programId}/locations")
void getLocations(@Path("programId") Long programId,
                  @Query("last_checked") Long timestamp,
                  Callback<List<LocationDTO>> cb);
```

metódusdeklaráció egy aszinkron kérést jelöl.

A *Retrofit*-os interfészek központi entitásoknak felelnek meg: `RetrofitProgramApi`, `RetrofitEventApi`, `RetrofitLocationApi` stb. A `RetrofitAuthenticationApi` a bejelentkezéssel kapcsolatos metódusokat tartalmazza. Az API kliens legfontosabb osztálya az `ApiManager`, melynek fő feladata a RetrofitApi interfészeknek megfelelő objektumok létrehozása. Az `ApiManager` a Factory tervezési minta szerint épül fel, statikus `createApi` metódusai a Retrofit `RestAdapter` osztályának segítségével a paraméterként megadott típusból hoznak létre egy példányt. Az erőforrások védettségének különbözősége miatt szükséges a `createApi` metódus háromszoros túlterhelése: az első változat olyan erőforrásokat feltételez, melyekhez nincs korlátozva a hozzáférés; a második abban az esetben használható, ha access token-re van szükség (az `AuthenticationApiBean` használja); a harmadik garantálja, hogy az access token minden kérés esetében be lesz állítva a fejlécben. Ezek a beállítások egy `RequestInterceptor`-ban vannak meghatározva, amely minden kérés elküldése előtt lefut. Ha az access token érvényessége lejárt *Unauthorized* hibaüzenet érkezik vissza. Ekkor használható fel a refresh token, melynek segítségével új access token szerezhető. Az erre vonatkozó kérést a `RestAdapter`-nek beállított `Authenticator` küldi el. Az `Authenticator` osztály az OkHttp könyvtár eleme, `authenticate` metódusa minden olyan esetben lefut, mikor a rendszer az Unauthorized hibaüzenetet kapja vissza.

3.4. Szinkronizációs mechanizmus

A rendezvények programja több száz eseményt is tartalmazhat, ezért a fejlesztés során fontos szempont volt az, hogy ezt a nagyobb adatmennyiséget az Android kliens csak egyszer töltse le a szerverről, mégpedig a rendezvény kiválasztásakor. Ezután csak a frissítéseket kérdezi le, ami jelentősen csökkenti az adatforgalmat.

A szinkronizációs mechanizmus a `SyncService` osztály segítségével valósul meg. Ez egy háttérben futó service-ként működik, `onHandleIntent` metódusa mindig lefut, ha az alkalmazás elindul, a felhasználó a pull-to-refresh szolgáltatáson keresztül kéri az adatok frissítését, vagy ha programváltás történik. Mikor a `SyncService` az API kliens metódusait hívja, paraméterként az utolsó frissítés dátumát tartalmazó időpecsétet is átadja. Mivel a szerver komponens támogatja az adatok ilyen módon történő lekérdezését, a visszatérített lista csak a legutóbbi frissítés óta módosult adatokat fogja tartalmazni. Ezután a `SyncService` az adathozzáférési réteg megfelelő metódusait hívja az adatok lementéséhez. Ha programváltás történt, az utolsó frissítés dátuma 0, így a `SyncService` minden adatot visszakap, törli a megfelelő táblák tartalmát, és lementi az adatokat. A műveletek sorrendje biztosítja azt, hogy ha valamilyen hiba történik az adatok lekérdezésénél, az alkalmazás visszaálljon az előzőleg megjelenített rendezvény programjára.

A felhasználó által megjelölt kedvenc események szintén programváltáskor szinkronizálódnak a szerverrel: a `SyncService` lekérdezi az újonnan kiválasztott rendezvénynek megfelelő kedvenc események azonosítóját (ha már léteznek a szerver adatbázisában), a memóriában lévő kedvenceket feltölti a megfelelő programhoz, majd az eseményekkel együtt törli a kedvencekre vonatkozó információkat (az event tábla favourite oszlopának tartalmát). Miután lementi az új eseményeket az adatbázisba, a kedvenc események azonosítóit tartalmazó lista alapján frissíti az `Event` objektumok `favourite` mezőjét.

3.5. A felhasználói felület

Az Android alkalmazás felhasználói felületének felépítése követi az MVC tervezési mintát. A legfőbb alkotóelem az `Activity` osztály, amely a kontroller szerepét látja el, és jól meghatározott funkcionalitásokat foglal magába. Minden `Activity` osztályhoz egy XML állományban leírt nézet tartozik, amely tartalmazza a megjelenítendő elemeket (gombok, listák, szövegdobozok stb.), illetve az elrendezésre és megjelenítésre vonatkozó információkat. Ezekhez az elemekhez az `Activity` osztályban rendelhető érték (itt kapcsolható össze a modell a nézettel), illetve itt határozható meg az elemek viselkedése is. Egy activity több fragmentet tartalmazhat. A fragment az activity-hez hasonló elven működő kisebb logikai egység, általában egyetlen funkcionalitást foglal magába, a felhasználói felület egy elkülöníthető részéért felelős [16].

A FestivApp alkalmazás központi activity-je a `MainActivity` osztály. Ez több fragmentet is tartalmaz, melyek között az oldalsó menü, a NavigationDrawer segítségével lehet váltani. Innen érhető el a `PagerFragment`, mely tartalmazza a napi bontásban megjelenített programot. Minden napnak egy `DayFragment` felel meg, ezek között egy ViewPager segítségével lehet váltogatni. A `ProgramsActivity` szintén a `MainActivity`-ben található menüből érhető el, és a rendezvények közötti váltást teszi lehetővé. Ez azonban csak akkor lehetséges, ha az eszköz kapcsolódva van az internetre, és a szerver elérhető. Szintén a menüből érhető el a `LoginActivity` és a `RegistrationActivity`, amelyek a bejelentkezést, illetve a regisztrációt teszik lehetővé. Az alkalmazás használatához nem föltétlenül szükséges a bejelentkezés, viszont vannak olyan funkcionalitások, melyek csak bejelentkezett felhasználók számára érhetőek el. Például, egy eseményt kedvencnek jelölni, vagy a kedvencek listáját megtekinteni csak bejelentkezett felhasználók számára lehetséges.

Az alkalmazás fontos funkcionalitása a keresés (események címére vagy előadók nevére lehet keresni). Az ezzel kapcsolatos logikát a `SearchActivity` tartalmazza, amelynek legfontosabb eleme az Android Support Library által biztosított SearchView. A SearchView teszi lehetővé például javaslatok megjelenítését két karakter beírása után. Ennek megvalósításához egy `ContentProvider` osztály létrehozására volt szükség, amely az adathozzáférési réteghez

fordul az aktuális találatok lekérdezéséhez. Egy javaslat kiválasztásakor, vagy a keresés gombra kattintva a találatok egy dátum szerint rendezett listában jelennek meg.

A fent említettek activity-k mellett létezik még az `EventActivity` és a `LocationDetailsActivity`. Az `EventActivity` egy esemény részletes adatainak megjelenítéséért felelős. A `LocationDetailsActivity` tartalmazza egy helyszín leírását, valamint a rendezvény térképén feltüntve a helyszín koordinátáit. A rendezvény térképe egy statikus kép, amely offline módban is elérhető. A helyszínhez navigációt lehet kérni a Google Maps-re mutató hivatkozás segítségével.

Az alkalmazás fejlesztése során gyakran biztosítani kellett azt, hogy két activity kommunikálni tudjon egymással, például az egyik el tudjon indítani egy másikat, és különböző információkat tudjon átadni neki. Például, a `DayFragment` listaelemére kattintva el kell induljon az `EventActivity`, melynek meg kell jelenítenie a megfelelő eseményre vonatkozó információkat. Ez Intent-ek segítségével valósul meg. Ha a kommunikáció nem azonnali, akkor `BroadCastReceiver` osztályokra van szükség. A `BroadCastReceiver` `onReceive` metódusa adott BroadCast hatására hívódik meg. A FestivApp esetében például a háttérben futó `SyncService` egy BroadCast formájában értesíti a felhasználói felület megfelelő elemeit a műveletek sikerességéről vagy sikertelenségéről. Ha a kommunikáció nem irányított, vagyis nem meghatározott, hogy pontosan melyik osztálynak lesz szüksége az adatokra, akkor használható az Android SharedPreferences osztálya. A SharedPreferences primitív típusokat képes kezelni, az adatokat kulcs-érték párok formájában menti le a telefon memóriájába. Ezek az értékek az alkalmazás újraindítása után is elérhetőek lesznek. A FestivApp esetében a SharedPreferences osztály például abban segít, hogy a kiválasztott rendezvény azonosítóját mindegyik activity le tudja kérdezni, és ez az érték újraindítás esetén is megmaradjon a memóriában.

A FestivApp alkalmazás egyik legfontosabb eleme az események listája. Fontos tehát, hogy feltöltése, görgetése, újrarajzolása gyorsan és gördülékenyen történjen. A gyorsítás érdekében a lista nézet implementálásához nem a hagyományosnak számító ListView, hanem a sokkal jobb teljesítményre képes RecyclerView került alkalmazásra.

A RecyclerView „egy flexibilis nézet, amely adott méretű ablakon keresztül nyújt betekintést egy nagy adathalmazba” [17]. Ahogy a neve is jelzi, az újrahasznosítás elvén működik: azok az elemek, amelyek lekerültek a képernyőről a görgetés következtében, nem szűnnek meg létezni; a rendszer úgy hasznosítja őket újra, hogy a tartalmukat lecseréli azokra az adatokra, amelyeknek a következőkben kell megjelenniük a képernyőn.

A FestivApp alkalmazás esetében az adathalmaz annyira nagyméretű lehet, hogy a memóriában való tárolása már a teljesítmény rovására menne. Ezért a RecyclerView-hoz adatbázis kurzorra épülő adapter van társítva, amely az adatokat (például egy eseményt) csak akkor olvassa be a memóriába, ha azokra valóban szükség lesz (megjelennek a képernyőn).

3.5.1. Material Design irányelvek

A Google által bevezetett *Material Design* irányelvek célja egy olyan „vizuális nyelv kialakítása, amely egyesíti a bevált design princípiumokat a technológia és a tudomány találmányaival, lehetőségeivel” [18]. A Material Design princípiumai olyan kinézet- és viselkedésbeli tulajdonságokat határoznak meg, amelyek segítenek érthetőbbé és átláthatóbbá tenni egy felhasználói felületet. Annak érdekében, hogy a FestivApp felhasználóbarát, könnyen használható alkalmazás legyen, célszerű volt követni ezeket az elveket. A FestivApp olyan elemeket, színeket és animációkat tartalmaz, amelyek megfelelnek a Material Design követelményeknek.

A felhasznált elemek közül kiemelendő a *Floating Action Button*. Az irányelvek szerint erre a gombra kattintva a felhasználónak az oldalhoz kapcsolódó legfontosabb funkcionalitást kell elérnie. A FestivApp főoldalán például a Floating Action Button-on keresztül érhető el a szűrés lehetősége, illetve az `EventActivity`-ben ennek a gombnak a segítségével lehet kedvencnek nyilvánítani egy adott eseményt. A Material Design elveket követi ezenkívül az oldalsó menü (*Navigation Drawer*), a listaelemek, valamint ezek animációja, a főoldalon található lapozó (*Tab Layout*), a térkép animációja, illetve mindegyik fragment és activity kinézete.

A színek beállítása szintén a Material Design szerint történik: az alkalmazáson belül két szín és ezek árnyalatai jelennek meg: a fő szín (*primary color*) és a kiemelő szín (*accent color*). Annak érdekében, hogy ez a két szín a kiválasztott fesztivál színvilágát tükrözze, a színekre vonatkozó beállításokat dinamikus módon kellett elvégezni, ami több felületi elem esetén is kihívást jelentett, hisz az Android alapértelmezetten az XML alapú stílusdefiníciót támogatja.

A FestivApp-ban használt Material Design elemek jelentős része az Android Support Design könyvtár része, de más külső library-k is használva voltak azoknak az elemeknek az implementálásához, amelyek nem jelennek meg az előbb említett könyvtárban.

3.6. Dependency injection

Android alkalmazások fejlesztése során nem áll rendelkezésre IoC konténer, így alapértelmezetten a DI sem támogatott. Léteznek viszont alternatív megoldások, melyek - bár nem hasonlíthatóak össze az IoC konténerek által nyújtott lehetőségekkel - támogatják a DI minta alkalmazását.

3.6.1. Dagger

A *Dagger* könyvtár a `javax.inject` csomag standard annotációira épít. A DI megvalósítása érdekében elegendő `@Module` annotációval ellátott osztályokat létrehozni, melyekben megtörténik az injektálásra szánt osztályok példányosítása. A példányosítás módját is szabályozni lehet, használható például a `@Singleton` annotáció, amely jelzi a *Dagger*-nek, hogy az érintett

osztálynak csak egy példányát hozza létre [19].

Például a `RepositoryModule` osztály az adathozzáférési rétegen található entitások példányosításáért felelős. A `LocationRepository` objektumot példányosító metódusa a következőképpen néz ki:

```
@Provides
@Singleton
LocationRepository provideLocationRepository() {
    return new OrmLiteLocationRepository(context);
}
```

A `FestivApplication` osztályban található az `ApplicationComponent` interfész, amely a Dagger `@Component` annotációjával van ellátva. Ez a komponens felel a tulajdonképpeni injektálásért. Egy osztály, ha injektálni szeretne, az `ApplicationComponent` megfelelő `inject` metódusának saját magára mutató referenciát ad át.

3.6.2. Butterknife

A *Butterknife* könyvtár segítségével lehetségessé válik a felhasználói felület elemeinek injektálása. Ehhez az activity-ben a megfelelő adattagot a `@Bind` annotációval kell ellátni. Szükséges ezenkívül az elem azonosítójának megadása is. A Butterknife a háttérben ezekből az információkból kódot generál (a másik megoldás a reflection használata lenne, de ebben az esetben ez túl lassúnak bizonyulhatna) [20]. Például:

```
@Bind(R.id.activity_event_txt_location)
TextView location;
```

4. A webes kliens alkalmazás

A FestivApp webes komponensének felhasználói felülete HTML nyelvben íródott, a dinamikus tartalmak kezelésére JavaScriptet alkalmaz és RESTful API-n keresztül kommunikál a szerverrel.

4.1. Felhasznált technológiák

A kliens alkalmazás esetében az *Angular JS* keretrendszer felelős a webes felület komponenseinek kezeléséért, a navigációs logika meghatározásáért, illetve a szerverrel való aszinkron kommunikáció biztosításáért. Az objektumorientált megközelítés alkalmazhatósága a

TypeScript nyelv használata által biztosított. A felhasználói felület a *Bootstrap* keretrendszer segítségével van felépítve.

4.1.1. Angular JS

Az Angular JS egy pehelysúlyú modulokra épülő JavaScript keretrendszer, amely olyan programozási mintákat alkalmaz, mint az MVC, a DI, illetve a *two way binding*.

Mivel az Angular JS támogatja az MVC tervezési mintát, könnyen elkülöníthetőek a megjelenítendő adatok, a felhasználói felület komponensei, illetve a vezérlés. A nézeteket Angular direktívákkal felruházott HTML oldalak képviselik, melyeket sablonnak nevezünk. A keretrendszer gyorsasága annak köszönhető, hogy első indításnál betöltődik minden vezérléshez szükséges kód, illetve a külső függőségek, majd futás közben tölti le az említett sablonokat, amikor szüksége van ezekre. A sablonok által flexibilis felületet biztosít a fejlesztő számára a DOM (Document Object Model) elemek manipulálásához. A DI-nek köszönhetően modularizált, lehetőséget biztosítva a fejlesztő közösség számára új modulok megírására.

Számos modulja közül érdemes megemlíteni a *Restangular*-t, amely a hálózati kommunikációért felel, a *ui-router*-t, amely a nézetek egymásba ágyazhatóságát teszi lehetővé, illetve az *ngTable*-t, amely a szerverről visszatérő adatok táblázatokban történő megjelenítsét és szerkesztését támogatja.

Az Angular JS támogatja a *two way binding* módszert, az adatmodellt két oldalról köti le: egyrészt a felhasználói felületen, másrészt a JavaScript vezérlő részen. A programozási mintának köszönhetően a kód lerövidül, mert ha a felhasználó módosítja a felületen az adatot, a módosítás automatikusan érvényesül a háttérben, és ugyanez fordítva is igaz [21].

4.1.2. TypeScript

A TypeScript a JavaScriptet kibővítő objektumorientált nyelv, melyet a Microsoft fejlesztett ki. A TypeScript kódból egy fordító generál a böngésző által futtaható JavaScriptet.

A nyelv bevezeti a típusok és osztályok fogalmát, így szigorúbban meghatározhatóak az adatmodellek, illetve a hivatkozások biztonságosabbak lesznek, mert típus inkompatibilitás esetén fordítási hibát kapunk. Ezek miatt a tulajdonságai miatt esett erre a technológiára a választás a FestivApp esetében.

Ahhoz, hogy az Angular JS modulokat optimálisan használjuk és kihasználjuk a fordító előnyeit, létezik a GitHub-on egy DefinitelyTyped nevű repository, amelyből a compiler letölti az adott modulok TypeScript definícióját, majd ellenőrzi a megírt kód helyességét [22].

4.1.3. Twitter Bootstrap

Mivel több webes böngésző és ezekből számos verzió létezik, nehéz olyan felületet írni, amely ugyanúgy néz ki minden böngészőben. Erre ad megoldást a Twitter Bootstrap keretrendszer. Amellett, hogy egységes beállítást biztosít a különböző böngészők renderelő egységeinek, egy alap stílust is meghatároz a weboldalnak. Lehetőséget ad arra is, hogy különböző képernyőméretek esetében másképp jelenjen meg a weboldal (például elemek kerüljenek máshová, vagy tűnjenek el), így a mobil készülékekről történő böngészést is kényelmesebbé teszi.

4.2. A webes kliens megvalósítása

A webes kliens adatmodellje TypeScript interfészek segítségével van felépítve és néhány specifikus mező kivételével megfelel a szerver komponens adtmodelljének. A szerver komponenshez hasonlóan itt is megjelennek az `IProgram`, `IEvent`, `IPresenter`, `ILocation` és `ICategory` központi entitások, melyek segítségével egy esemény programja írható le. Ezek mellett megjelenik az `INotification` entitás, amely egy telefonos értesítést (push notification) ábrázol, illetve az `IStyle`, amely egy adott rendezvény brandjének grafikus elemeit reprezentálja.

A szerverrel való kommunikációt és az erőforrások feldolgozását a Restangular modul végzi. A Restangular keretrendszer felelős azért, hogy aszinkron kérést indítson a szerverhez, megvárja és feldolgozza a választ, majd visszajelezzen a Controller-nek. A visszajelzést `Promise`-okon keresztül végzi, callback függvények segítségével. A keretrendszer előnyös tulajdonsága a beágyazott erőforrások automatikus feldolgozása is [23]. A FestivApp esetében ilyen például az `IProgram`-on belüli `IStyle` entitás, amelyet bármilyen plusz művelet nélkül feldolgoz a Restangular.

Mivel a REST API autentikációt igényel, szükség volt egy egyszerű megoldásra a HTTP kérések hitelesítésére. A megoldást a HTTP interceptorok szolgáltatták. Minden HTTP kérést felfog egy interceptor és a kérés fejlécébe illeszti a bejelentkezésnél kapott access token-t, ezáltal azonosítva magát a szerveren. Ugyanez a szolgáltatás felelős azért, hogy ha a szerver Unauthorized (401) hibakódot ad, akkor a felhasználót a bejelentkezés oldalra irányítsa [24].

Mivel a HTTP kérések állapot nélküliek, szükség van egy helyre, ahol a felhasználóspecifikus információk tárolhatóak. A HTML5-ben bevezetett *Local Storage* jó megoldást ad erre. A bejelentkezés után kapott access token ide mentődik, majd egy HTTP interceptor minden szerverhez indított kérés esetén validálja a tokent és beleírja a kérés fejlécébe. A tokenen kívül a felhasználó által kiválasztott program azonosítója és az oldalsó menü állapota is a Local Storage-be mentődik, hogy minden bejelentkezés után ne kelljen újra megadni a legutóbb kiválasztott programot, illetve ne kelljen mindig újra kinyitni vagy bezárni az oldalsó menüt.

5. Eszközök és módszerek

A FestivApp fejlesztését több olyan népszerű eszköz segítette, melyeket a programozók és fejlesztő csapatok előszeretettel használnak szerte a világban. Az alábbiakban ezek közül a legfontosabbak kerülnek bemutatásra. Ezek közül a legtöbb a Scrum módszer által kitűzött célok elérésében segített.

A Scrum egy agilis módszer, amely inkrementáló és iteratív fejlesztést feltételez. A csapat igyekezett a módszer által ajánlott tevékenységek szerint dolgozni. A fejlesztés kéthetes sprintekben zajlott, a cél pedig az volt, hogy minden sprint eredménye a FestivApp egy új, működő verziója legyen. A sprintek elején sor került a planning („futamtervező") megbeszélésre, amely során a csapat a FestivApp backlogjából kiválasztotta az elvégzendő feladatokat, és megbecsülte az implementáláshoz szükséges időt. Projektmenedzsmentben, a feladatok nyilvántartásában és követésében a Trello segített.

Verziókövetéshez a csapat Mercurial rendszert használt. Az új funkcionalitások fejlesztése külön ágakon valósult meg. Ha a funkcionalitás elkészült és jóvá lett hagyva, az ág lezáródott, és a stabilnak tekintett default ágba merge-ölődött.

A buildelésben és a függőségek menedzselésében a Gradle segített. A Gradle Groovy alapú build szkripteket használ, és nagyon hasznos eszköz multi-project buildek esetében. Mivel a FestivApp web modulja nem egy Java projekt, a Gradle ezt nem tudja közvetlenül buildelni, csupán elindítja a Gulp megfelelő utasításait, és az eredményt a szerver modul webapp mappájába másolja. Ebből a mappából szolgáltatja a szerver a webes felület statikus erőforrásait.

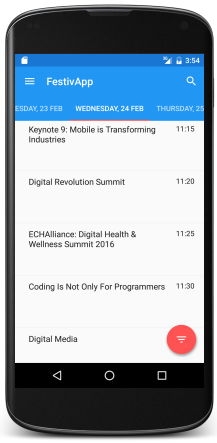
A webes kliens függőségeit a Bower menedzseli, és a Gulp automatikusan injektálja a weboldal forráskódjába. Ugyancsak a Gulp felelős a web modul fejlesztési verziójának a buildeléséért, amely kevésbé tömörített és több lehetőséget ad a debuggolásra, illetve a kiadott (release) verzióknál használt optimalizált kód legenerálásáért is ő felel. Ezek mellett a Browsersync figyeli a forráskód változását, és biztosítja a live reload támogatást. A webes fejlesztésben résztvevő eszközök az npm csomagkezelővel vannak felügyelve, melynek segítségével nagyon egyszerűen konfigurálható a környezet. A webes kliens kezdeti vázának legenerálásában a Yeoman projektgeneráló nyújtott segítséget.

A folytonos integráció (Continuous Integration - CI) módszerének alkalmazásával a hibákat egyszerűen és gyorsan ki lehetett szűrni, ugyanis, ha új kód került a központi tárolóba, automatikusan buildelődött a teljes projekt, az automatizált tesztek futtatását is beleértve. A FestivApp CI rendszere a Jenkins, amely több száz plugin segítségével teszi könnyebbé az automatizált build folyamatát. A teljes projektet érintő build minden default ágat érintő push művelet után lefutott. Az automatikus build után a SonarQube statikus kódelemző ellenőrzései is lefutottak, így a forráskód minősége is folyamatosan monitorizálva volt.

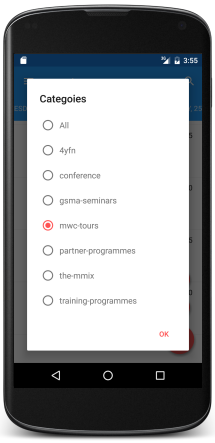
6. A FestivApp működése

Az Android alkalmazás első indításakor ki kell választani azt a rendezvényt, amelynek programját az applikáció megjeleníti. Miután ez megtörtént, megjelenik a főoldal a napokra lebontott programmal (2. ábra). Ezt a programot kategóriák és helyszínek szerint lehet szűrni, ez a funkció az oldal alján található gomb lenyomásával érhető el (3. ábra).

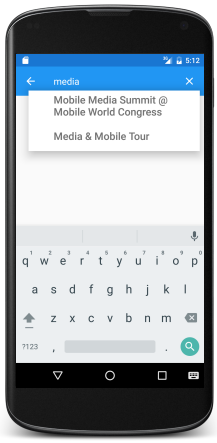
Keresni a jobb sarokban megjelenő nagyító jelre kattintva lehet. A szövegmezőbe legalább két karaktert beírva egy lenyíló listában láthatóak az aktuális találatok (4. ábra). A keresés eredményei időrendi sorrendben jelennek meg egy listában. Egy esemény részletes leírását a megfelelő listaelemre kattintva lehet elolvasni. Ilyenkor egy új oldal jelenik meg, amely egy olyan gombot is tartalmaz, melynek lenyomásával az esemény kedvencnek nyilvánítható (5. ábra).



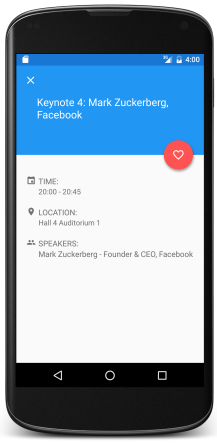
2. ábra. MainActivity



3. ábra. Szűrő

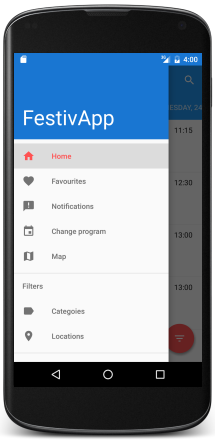


4. ábra. SearchActivity

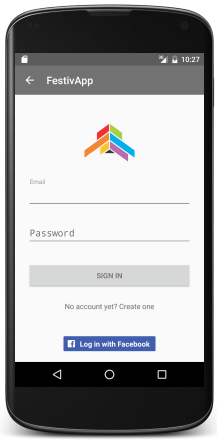


5. ábra. EventActivity

Az alkalmazás fontos eleme a menü (6. ábra). Innen további funkcionalitásokat és oldalakat lehet elérni. Külön nézeten tekinthetőek meg például a beérkezett értesítések és a kedvencnek jelölt események, a rendezvény térképe, további információk a kategóriákról és a helyszínekről, és innen lehetséges a rendezvények közötti váltás is. A bejelentkezést (7. ábra) és a regisztrációt lehetővé tevő oldalakat szintén a menüből lehet elérni.

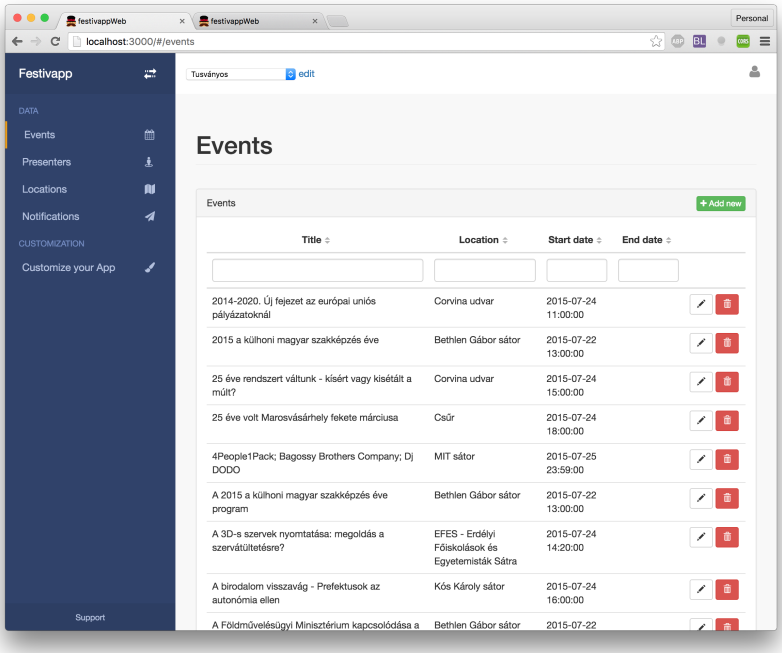


6. ábra. NavigationDrawer



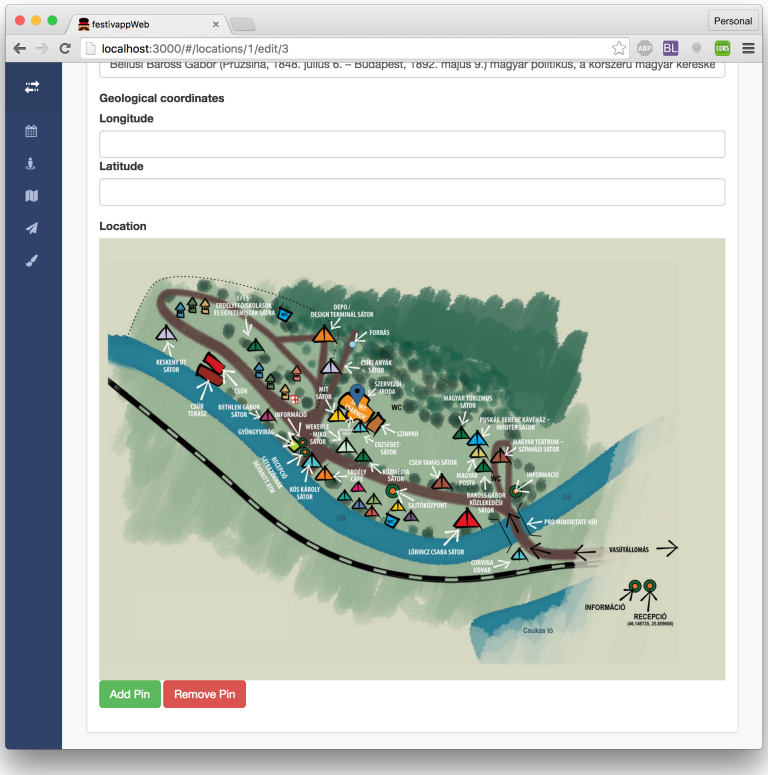
7. ábra. LoginActivity

A webes alkalmazás esetében, bejelentkezés után a felhasználó a kiválasztott rendezvény eseményeit láthatja (8. ábra). A webes felület három részre bontható: az oldalsó menü, amelyben a rendezvénnyel és alkalmazással kapcsolatos műveleteket lehet elérni, a felső műveleti sáv, amely a rendezvényváltás lehetőségét biztosítja, illetve a tartalom rész, amely a kiválasztott oldal tartalmát mutatja. Az események, előadók és helyszínek egy táblázatban jelennek meg, ahol lehetőség van szűrni, rendezni, törölni, módosítani, illetve újat hozzáadni.



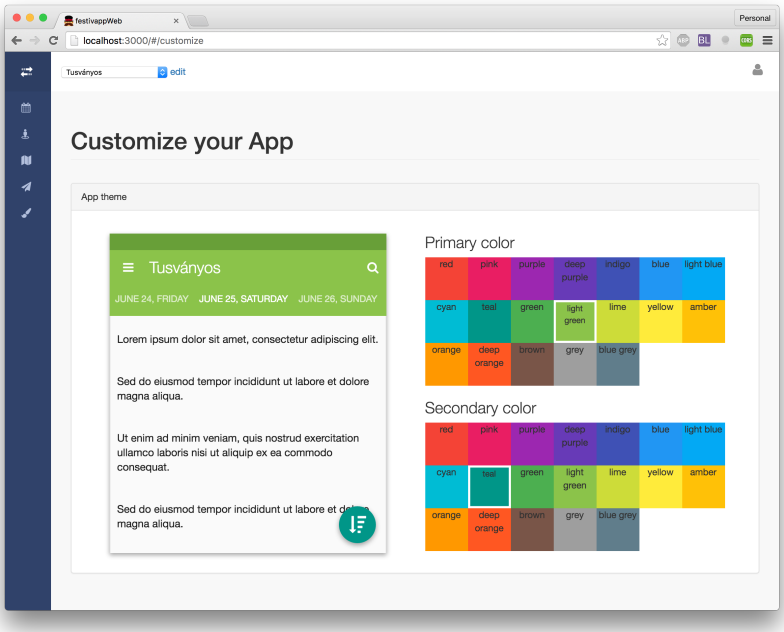
8. ábra. Események

Minden eseménynek meghatározható a neve, beállítható az időzónája, a kezdeti és végső dátuma. Az eseményhez hozzárendelhető egy egyedi térkép, amelyre markerek helyezhetőek el az egyes helyszínek szerkesztésénél. Emellett a tényleges földrajzi koordináták is megadhatóak, így az applikáció a Google Maps segítségével navigációt is biztosíthat (9. ábra).



9. ábra. Helyszín adatainak meghatározása

A webes felület a kiválasztott rendezvény applikáción belüli kinézetének módosítására is lehetőséget ad (10. ábra).



10. ábra. Testreszabás

Következtetések és továbbfejlesztési lehetőségek

A céloknak megfelelően sikerült egy olyan általános rendszert felépíteni, amely lehetőséget biztosít rendezvények programjának menedzsmentjére és böngészésére (akár internetkapcsolat nélkül is). Létrejött egy hatékony szinkronizációs mechanizmus, sikerült elérni, hogy az alkalmazás a megjelenített rendezvény brandjének megfelelően jelenjen meg, és kifejlődött egy olyan API, amelyet akár külső rendszerek is használhatnak, ha a FestivApp adatbázisában tárolt adatokhoz akarnak hozzáférni.

Az alkalmazás már több rendezvény résztvevőit segítette a programban eligazodni. A felhasználók visszajelzései alapján és a fesztiválszervezőkkel való beszélgetések során rengeteg új ötlet született, melyek mind a továbbfejlesztési lehetőségek között szerepelnek. Ezek közül a fontosabbak:

- hírfolyam implementálása, amelyben a rendezvény hírei, közleményei jelennének meg;
- időjárás előrejelzés a szabadtéren megrendezésre kerülő eseményekhez;
- a rendezvények adatainak nemzetköziesítése, lehetőséget biztosítva a szervezőknek, hogy a programot több nyelven is feltölthessék;
- ismerősök tartózkodási helyének behatárolása;
- helyek megosztása a résztvevők személygépkocsijaiban;
- régiókezelés: a rendezvények szűrése régió/ország szerint.

A mobil alkalmazás fejlesztése során fontos szempont volt az offline működés garantálása. Ez azért szükséges, mert egy nagyobb fesztivál esetében az internetkapcsolat nem mindig biztosított. Az is nagyon gyakran előfordul, hogy a mobil hálózat annyira leterhelődik, hogy a telefonálás és az üzenetküldés is lehetetlenné válik. Emiatt merült fel az ötlet, hogy meg lehetne próbálni kifejleszteni egy belső chatet, amely alternatív csatornát is tudna használni az üzenetek továbbítására, a kommunikáció történhetne például BlueTooth vagy WiFi direct segítségével.

A webes felület továbbfejlesztésére is vannak tervek, elsősorban a program felvitelére vonatkozóan. A kézi feltöltés célszerű abban az esetben, ha az adatok először kerülnek fel az internetre. Ha azonban egy másik adatbázis vagy statikus weboldal már tartalmazza ezeket, célszerű lenne, ha importálásuk automatizált módszerrel is megtörténhetne.

Hivatkozások

[1] C. Ho, R. Harrop, and C. Schaefer, *Pro Spring 3*. Apress, 2012.

[2] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, and S. Deleuze, *Spring Boot Reference Guide*. [Online] http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ [Utolsó megtekintés dátuma: 2016-04-20]

[3] O. Gierke, T. Darimont, C. Strobl, and M. Paluch, *Spring Data Reference Documentation*. [Online] http://docs.spring.io/spring-data/jpa/docs/current/reference/html/ [Utolsó megtekintés dátuma: 2016-04-20]

[4] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B. Hale, A. Colyer, J. Lewis, C. Leau, M. Fisher, S. Brannen, R. Laddad, A. Poutsma, C. Beams, T. Abedrabbo, A. Clement, D. Syer, O. Gierke, R. Stoyanchev, P. Webb, R. Winch, B. Clozel, S. Nicoll, and S. Deleuze, *Spring Framework Reference Documentation*. [Online] http://docs.spring.io/autorepo/docs/spring/current/spring-framework-reference/htmlsingle/ [Utolsó megtekintés dátuma: 2016-04-20]

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

[6] B. Alex, L. Taylor, R. Winch, and G. Hillert, *Spring Security Reference*. [Online] http://docs.spring.io/autorepo/docs/spring-security/4.1.0.RC1/reference/htmlsingle/ [Utolsó megtekintés dátuma: 2016-04-20]

[7] E. D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749. [Online] http://tools.ietf.org/html/rfc6749 [Utolsó megtekintés dátuma: 2016-04-24]

[8] A. Parecki. (2012) OAuth 2 Simplified. [Online] https://aaronparecki.com/2012/07/29/2/oauth2-simplified [Utolsó megtekintés dátuma: 2016-04-24]

[9] M. Jones, J. Bradley, and N. Sakimura. JSON Web Token (JWT). RFC 7519. [Online] https://tools.ietf.org/html/rfc7519 [Utolsó megtekintés dátuma: 2016-04-24]

[10] Introduction to JSON Web Tokens. Auth0. [Online] https://jwt.io/introduction/ [Utolsó megtekintés dátuma: 2016-04-24]

[11] C. Walls and K. Donald, *Spring Social Reference Manual*. [Online] http://docs.spring.io/spring-social/docs/1.0.x/reference/html/ [Utolsó megtekintés dátuma: 2016-04-20]

[12] Android Platform. Techopedia. [Online] https://www.techopedia.com/definition/4219/android-platform [Utolsó megtekintés dátuma: 2016-04-24]

[13] Support library. Android. [Online] http://developer.android.com/tools/support-library [Utolsó megtekintés dátuma: 2016-04-24]

[14] G. Watson, *OrmLite - Lightweight Object Relational Mapping (ORM) Java Package*. [Online] http://ormlite.com/ [Utolsó megtekintés dátuma: 2016-04-24]

[15] Retrofit: A type-safe HTTP client for Android and Java. Square. [Online] http://square.github.io/retrofit/ [Utolsó megtekintés dátuma: 2016-04-24]

[16] Z. Mednieks, L. Dornin, G. B. Meike, and M. Nakamura, *Programming Android*, 2nd ed. O'Reilly Media, 2012.

[17] RecyclerView. Android API Reference. [Online] http://developer.android.com/reference/android/support/v7/widget/RecyclerView.html [Utolsó megtekintés dátuma: 2016-04-24]

[18] *Material Design Guidelines*, Google. [Online] https://www.google.com/design/spec/material-design [Utolsó megtekintés dátuma: 2016-04-24]

[19] Dagger: A fast dependency injector for Android and Java. Square. [Online] http://square.github.io/dagger/ [Utolsó megtekintés dátuma: 2016-04-24]

[20] J. Wharton. Butterknife. [Online] http://jakewharton.github.io/butterknife/ [Utolsó megtekintés dátuma: 2016-04-24]

[21] *Angular JS API Reference*, Google. [Online] https://docs.angularjs.org/api [Utolsó megtekintés dátuma: 2016-04-24]

[22] J. Preece. (2016) Writing AngularJS 1.x with TypeScript. [Online] http://www.developerhandbook.com/typescript/writing-angularjs-1-x-with-typescript/ [Utolsó megtekintés dátuma: 2016-04-24]

[23] M. Gontovnikas. Restangular. [Online] https://github.com/mgonto/restangular [Utolsó megtekintés dátuma: 2016-04-24]

[24] N. Ye. (2014) Interceptors in AngularJS and Useful Examples. WebDevEssay. [Online] http://www.webdeveasy.com/interceptors-in-angularjs-and-useful-examples/ [Utolsó megtekintés dátuma: 2016-04-24]