STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informační technologie

Mobilní aplikace pro Olympiádu dětí a mládeže

Filip Čacký
Liberecký kraj

Liberec 2019

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informační technologie

Mobilní aplikace pro Olympiádu dětí a mládeže

Mobile application for the Youth Olympic Games

Autoři: Filip Čacký
Škola: Střední průmyslová škola strojní a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3, příspěvková organizace, 460 01, Liberec
Kraj: Liberecký kraj
Konzultant: Mgr. Michal Stehlík

Liberec 2019

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Liberci dne 15. 3. 2019 ……
Filip Čacký

Poděkování

Tímto bych chtěl poděkovat magistru Michalu Stehlíkovi za odborné vedení práce, svým kolegům z firmy Skeleton Software za odborné rady a Jirkovi Vrbovi za dobrovolnou pomoc ve formě vývoje REST API generujícího smyšlená data pro testování webové služby a mobilní aplikace.

Anotace

Práce se zabývá vytvořením multiplatformní mobilní aplikace a webové služby na objednávku pro koncového zákazníka. Aplikace bude sloužit jako hlavní průvodce Olympiádou dětí a mládeže 2019 v Libereckém kraji. Webová služba bude sloužit k distribuci dat do mobilních klientů a k administraci. Účastníci olympiády tak získají prostřednictvím svého mobilního zařízení snadný přístup ke všem informacím, které je mohou v průběhu akce zajímat.

Klíčová slova

Mobilní aplikace; webová služba; Olympiáda

Annotation

This work is concerned with the development of a multi-platform mobile application and a web service commisioned by a customer. The application will be used as the main guide for the upcoming Summer Youth Olympic Games 2019 in the Liberec region. The web service is used for distributing data to mobile clients and for administration. The attendants thus get an easy access to all information that might concern them during the event through their mobile devices.

Keywords

Mobile application; web service; Olympics

Obsah

Úvod

Myšlenka na vytvoření mobilní aplikace pro olympiádu vzešla od bývalého náměstka hejtmana Libereckého kraje, pana Leoše Křečka, který řídil rezort školství, mládeže, tělovýchovy a sportu. Záměrem pana Křečka bylo do nastávající letní Olympiády dětí a mládeže zapojit studenty Libereckých škol a umožnit jim tak získat praktické zkušenosti, které by mohli zužitkovat ve svém oboru. Obrátil se tedy na SPŠSE a VOŠ v Liberci, zda by se našel student, který by měl zájem tuto aplikaci vytvořit. Práce mi byla nabídnuta panem magistrem Michalem Stehlíkem a jelikož mi připadala velice atraktivní – viděl jsem v ní unikátní příležitost se v průběhu studia zapojit do příprav celorepublikové akce s letitou tradicí, nabídku jsem rád přijal a zanedlouho začal s vývojem.

1 Komunikace se zákazníkem

Kapitola se zabývá schůzkami mezi Libereckým krajem a mnou a koncipováním potřebných a požadovaných funkcí mobilní aplikace a webové služby.

V průběhu vývoje probíhala mezi zástupci Libereckého kraje a mnou e-mailová a telefonická komunikace. Při vývoji bylo též potřebné komunikovat se třetími stranami. Mezi třetí strany patří IT firma eSport, která vyvíjí informační systém pro Český olympijský výbor, a dopravní společnost JezdiBusem, která pro Olympiádu dětí a mládeže zajišťuje dopravu.

1.1 Jednání o mobilní aplikaci

Za Liberecký kraj jsem nejčastěji komunikoval s panem bakalářem Miroslavem Šourkem, hlavním koordinátorem letní Olympiády dětí a mládeže 2019, se kterým jsem řešil požadavky na funkcionalitu a design aplikace. Pan Šourek navrhnul funkcionalitu aplikace a dodal potřebné podklady k vytvoření datové struktury a vzhledu.

Kromě telefonické a e-mailové komunikace proběhlo celkem sedm schůzek, které se týkaly vývoje. Předmět těchto schůzek se různil dle aktuálního stavu projektu a momentálně vyvíjených funkcí.

1.1.1 První schůzka 13. 6. 2018

První schůzka proběhla v červnu roku 2018 na půdě školy. Při této schůzce došlo ke slovní domluvě mezi zástupci Libereckého kraje a mnou, že budu aplikaci vyvíjet. Zazněl zde též první námět na moduly a využití aplikace. Prvotním nápadem byla aplikace, která uživatelům zobrazí neměnný seznam sportovních, ubytovacích a jídelních lokací a seznam autobusových spojů, kterými se na ně mohou dopravit. V tuto chvíli se ještě ani neuvažovalo o potřebné implementaci serverového backendu.

Původnímu návrhu aktuální stav aplikace moc podobný není – postupem času požadavky řídícího výboru Olympiády dětí a mládeže narostly. S každou schůzkou bylo nutné mnohé části aplikace pozměnit, aby vyhovovala nejnovějším požadavkům a představám.

1.1.2 Schůzky do 15. 1. 2019

Do 15. 1. 2019 se konaly celkem čtyři schůzky, první z nich se konala koncem září. Hlavním předmětem těchto schůzek bylo vytvoření návrhů pro všechny funkce aplikace.

Hned při první schůzce bylo jasné, že se nároky od červnové schůzky velmi zvýšily. Kvůli požadavku na zobrazení kompletního programu uživatelům bylo již nutné implementovat i serverovou část, která by aplikacím dodávala dynamická data.

1.1.3 Požadavky mobilní aplikace

V průběhu těchto schůzek postupně vznikly požadavky na následující moduly:

1. *Aktuality* – Aktuální informace z prostředí objednatele, případně dalších organizací. Možnost zasílání informačních sdělení uživatelům aplikace.
2. *Navigace* – Uživatel bude mít možnost navigace ze své aktuální pozice na sportoviště, ubytování a na pozici, kterou si uloží jako domovskou.
3. *Sportovní program* – Aplikace poskytne uživateli kompletní přehled o všech pořádaných sportovních soutěžích. Program si bude uživatel moci filtrovat dle svých priorit a zájmů.
4. *Výsledková část* – Uživatel bude mít k dispozici průběžně aktualizované výsledky sportovních soutěží a celkového pořadí krajů. Výsledkovou část bude moci uživatel filtrovat dle svých preferencí.
5. *Doprovodný program* – Aplikace poskytne uživateli přehled doprovodných akcí pořádaných během Her.
6. *Jídelníček* – Uživatel bude mít k dispozici jídelníček Her.
7. *Kontakty* – Uživatel bude mít k dispozici kontakty na pracovníky informačního centra zřízeného během akce.
8. *Doprava* – V aplikaci bude pro uživatele k dispozici jízdní řád přepravy na sportoviště a na doprovodné programy.
9. *Sekce s partnery* – Aplikace bude obsahovat loga partnerů dodaná objednatelem s proklikem na příslušné weby.

Hlavní problém, který v tuto chvíli nastal, byl přístup k oficiálním datům z informačního systému Českého olympijského výboru, jelikož bez nich by v aplikaci nebylo možné uživatelům zobrazit program. Jedním z možných řešení tohoto problému bylo vytvoření webové služby, která by sloužila k vytvoření dat ve stejném formátu, avšak kvůli obrovskému množství záznamů, které do oficiálního systému vkládá tým lidí o desítkách členů, by bylo použití takové služby nepřijatelné.

V této části vývoje proběhlo ze strany Libereckého kraje několik pokusů o navázání spolupráce mezi Českým olympijským výborem a mnou, bohužel se každý z těchto pokusů setkal s neúspěchem. Abych aplikaci mohl nadále vyvíjet, vytvořil jsem si prozatimní webovou službu, která mi dovolila připravit si testovací data.

I když byly veškeré požadavky na moduly aplikace dohodnuté, nebulo v tuto chvíli možné je všechny vytvořit. Stále nebyl koncipovaný formát autobusové dopravy pro modul *Doprava*. Vytvoření modulu *Kontakty* nebylo v tuto chvíli důležité, jelikož by se jednalo pouze o statickou stránku s informacemi. Kvůli nejistotě přístupu k datům od Českého olympijského výboru také nebyla implementována stránka s celkovým pořadím krajů. Důležitá byla implementace všech ostatních modulů kvůli příští schůzce.

1.1.4 Prezentace před partnery a řídícím výborem ODM

16. 1. 2019 se konalo zasedání řídícího výboru Olympiády dětí a mládeže, kterého se též účastnili hlavní partneři, pan magistr Petr Tulpa, náměstek hejtmana za rezort školství, mládeže, tělovýchovy, a sportu, zástupci Českého olympijského výboru a zástupce Technické univerzity v Liberci.

Den předtím proběhla generální zkouška. Součástí tohoto zasedání byla prezentace funkcí aplikace. Po této prezentaci přislíbil Český olympijský výbor spolupráci na vývoji aplikace.

1.1.5 Poslední dosavadní schůzka

Poslední dosavadní schůzka se konala 7. 3. 2019. Hlavním předmětem této schůzky bylo koncipování modulu *Doprava***.** Této schůzky se kromě pana Šourka zúčastnil také zástupce firmy JezdiBusem.

1.2 Požadavky na webovou službu

Vývoj ostré verze webové služby započal 30. 1. 2019. V tuto chvíli již přislíbil svou součinnost Český olympijský výbor.

1.2.1 Požadavky webové služby

Požadavky na webovou služby byly následující, vyplynuly z požadavků na mobilní aplikaci.

1. *Příprava dat* – Webová služba bude schopná vytvářet data, která se nezískávají ze systémů třetích stran – zprávy, jídelníčky, autobusové linky, ceremoniály, doprovodný program.
2. *Intergrace se systémy ČOV* – Webová služba bude napojena na systémy Českého olympijského výboru a bude z nich automaticky průběžně získávat potřebná data – sporty, soutěže, výsledky, sportovní lokace, ubytování, jídelny.
3. *Publikace dat* – Webová služba bude sloužit k publikaci dat do mobilních klientů.

1.2.2 RESTful API od firmy eSport

Po prezentaci mobilní aplikace před partnery a zástupci Českého olympijského výboru mi byl dán kontakt na projektového manažera firmy eSport. S ním jsem řešil specifikace RESTful API, které pro mě má firma vytvořit.

API bude sloužit k integraci informačního systému Českého olympijského výboru s mou webovou službou, webová služba si z něj bude stahovat data specifikovaná v požadavcích na webovou službu v bodě 2.

2 Výběr technologií

Tato kapitola se zabývá technologiemi zvolenými pro vývoj mobilní aplikace a webové služby.

Mobilní aplikace i webová služba jsou psány v jazyce C#. Hlavními důvody pro volbu tohoto jazyka jsou mé předchozí zkušenosti s ním, kvalitní dokumentace dostupná na webových stránkách *docs.microsoft.com/dotnet/* a aktivní komunita vývojářů na webových Q&A stránkách Stack Overflow.

2.1 Webová služba

Webová služba je vytvořená pomocí frameworku ASP.NET Core. Pro vytvoření uživatelského rozhraní byl využit návrhový vzor MVC, uživatelské rozhraní je vytvořené pomocí technologie Razor, design je vytvořen použit framework Twitter Bootstrap. Pro webové API, jež slouží k distribuci dat do mobilních klientů, je využit návrhový vzor REST.

Hlavním důvodem pro zvolení ASP.NET Core místo klasického ASP.NET frameworku je web server Kestrel. Aplikace hostované na IIS serveru využívající Kestrel běží v odděleném procesu od IIS (tzv. „out-of-process hosting model“). Kestrel je poměrně nový web server a neobsahuje tolik funkcí jako starý IIS HTTP Server, který musí udržovat zpětnou kompatibilitu s jeho předchozími verzemi. Výsledkem je efektivnější request pipeline, díky které dosahují ASP.NET Core aplikace využívající Kestrel mnohem vyššího výkonu.

2.1.1 Databáze

Webová služba využívá MSSQL databázi. K propojení databáze a webové služby je použita ADO.NET ORM knihovna Entity Framework Core a lazy-loading proxies.

Databázový model je vytvořen pomocí code-first přístupu. Všechny dotazy do databáze jsou tvořeny pomocí LINQ (Language INtegrated Query).

2.1.2 Mapování entit na modely

K mapování databázových entit na modely je využita knihovna AutoMapper. Tato knihovna mapuje datové typy dle jmenných konvencí. Je nutné vytvořit tzv. „mapy“ objektů, které udávají, jaké objekty se mají mezi sebou mapovat. Velkou výhodou je nastavení „CreateMissingTypeMaps“, díky kterému si AutoMapper mapy automaticky a průběžně vytváří sám, statické vytváření map je tak nutné pouze v případě jistých vyjímek.

2.1.3 Logování

Pro logování chyb a dění v aplikaci je využita knihovna Serilog. Tato knihovna umožňuje snadné vytváření strukturovaných logů a jejich zápis do velkého množství volně dostupných providerů (tzv. sink).

V této aplikaci jsou využity následující sinky:

1. Console – Tento sink je využit pouze při vývoji a debugování. Jakmile je aplikace deploynutá na hosting, není k němu umožněn přístup. Zapisuje logy do konzolového terminálu.
2. File – Tento sink umožňuje zápis logů do textového souboru ve specifikovaném adresáři.
3. Async – Jelikož se v aplikaci loguje z různých vláken, je využitý asynchronní sink, který slouží jako wrapper pro předchozí sinky. Díky tomuto sinku je logování v aplikaci thread-safe.

2.1.4 Provádění opakovaných úkolů

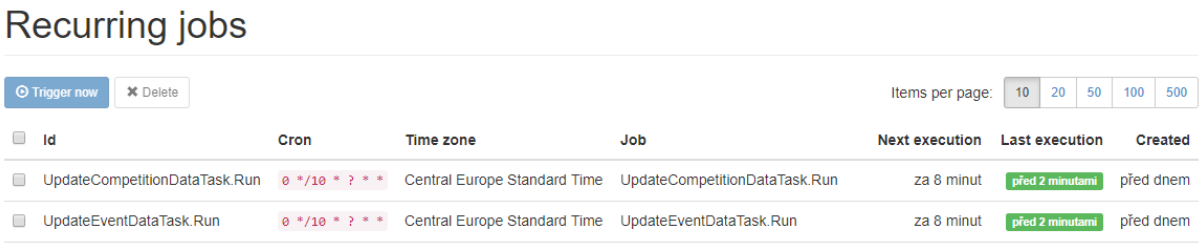
K provádění opakovaných úkolů nebo úkolů s prodlevou (průběžné stahování dat z REST API, automatická publikace datových souborů), je v aplikaci využita služba Hangfire.

Hangfire je služba pro plánování opakovaných, zpožděných, okamžitých a řetězených operací, které se provádí v samostatném vláknu. Samotný Hangfire server běží nezávisle na hlavím vláknu aplikace a pro ukládání svého stavu využívá MSSQL databázi.

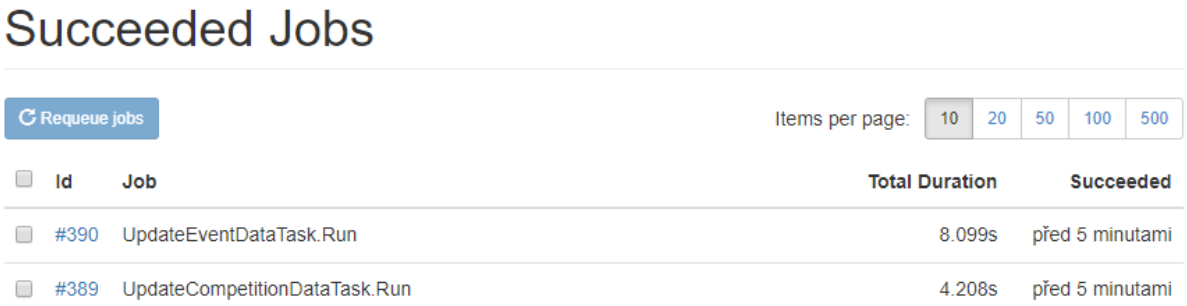
Velkou výhodou Hangfire serveru je automatické a nastavitelné opakování selhaných úkolů a velmi snadné ovládání chování úkolů při chybách. Možné je i jejich manuální spuštění.

Další součástí Hangfire služby je Hangfire dashboard, který umožňuje sledování stavu úkolů (momentálně probíhající, naplánovaný, selhaný, úspěšný...). O těchto úkolech dále automatický sbírá různé informace jako doba běhu, vypsané chyby a parametry metod.

Díky Hangfire tato aplikace pro svůj chod nevyžaduje žádný zásah správce, ten je nutný pouze v případě potencionálních neošetřených chyb, díky kterým není možné úspěšné dokončení úkolů.



Obrázek 2 – Příklad opakujících se úkolů v prostředí Hangfire dashboard



Obrázek 1 – Příklad úspěšně dokončených úkolů v prostředí Hangfire dashboard

2.2 Mobilní aplikace

Aplikace je vytvořená pomocí nástrojů Xamarin, které umožňují vývoj plně nativních mobilních aplikací. Velkou výhodou Xamarinu je sdílení kódu napříč platformami.

Aplikace vytvořené pomocí Xamarinu využívají společný codebase pro většinu své logiky a uživatelského rozhraní. Pomocí dependency-injections je možné do společného kódu implementovat platformně specifické funkce.

Aplikace je postavena nad proprietárním frameworkem firmy Skeleton Software – SkeletonFrameworkMobile. Tento framework integruje mnohé platformně specifické funkce, logování dění, chybové hlášení a další nástroje do společné části kódu. Hlavním důvodem použití tohoto frameworku je vývoj pro iOS. Jelikož nevlastním potřebný hardware nutný k vývoji aplikací na iOS zařízení (iPhone na spuštění a Mac na build a deploy aplikace) a nemám moc zkušeností s nativním iOS vývojem, využití tohoto frameworku je pro mě jediný způsob, jak aplikaci zpřístupnit uživatelům zařízení od firmy Apple.

2.2.1 Popup stránky

Jelikož části aplikace vyžadovaly navigaci přes velké množství úrovní, rozhodl jsem se kvůli uživatelské přívětivosti několik stránek vytvořit jako popup.

Jelikož žádná oficiální verze Xamarinu momentálně nenabízí tvorbu platformně nezávislých popup stránek, využil jsem proto knihovnu Rg.Plugins.Popup.

Tato knihovna slouží k vytvoření popup stránek ve společné části kódu a stará se o jejich správné vykreslení na všech platformách. Nabízí také mnohé přechodové animace.

3 Datové modely

Projekt ODM.Backend.Models je sdílený mezi mobilní aplikací a webovou službou z důvodu uniformity dat, která mobilní aplikace žádá od webové služby. Díky tomu není nutné psát pravidla pro JSON mapování.

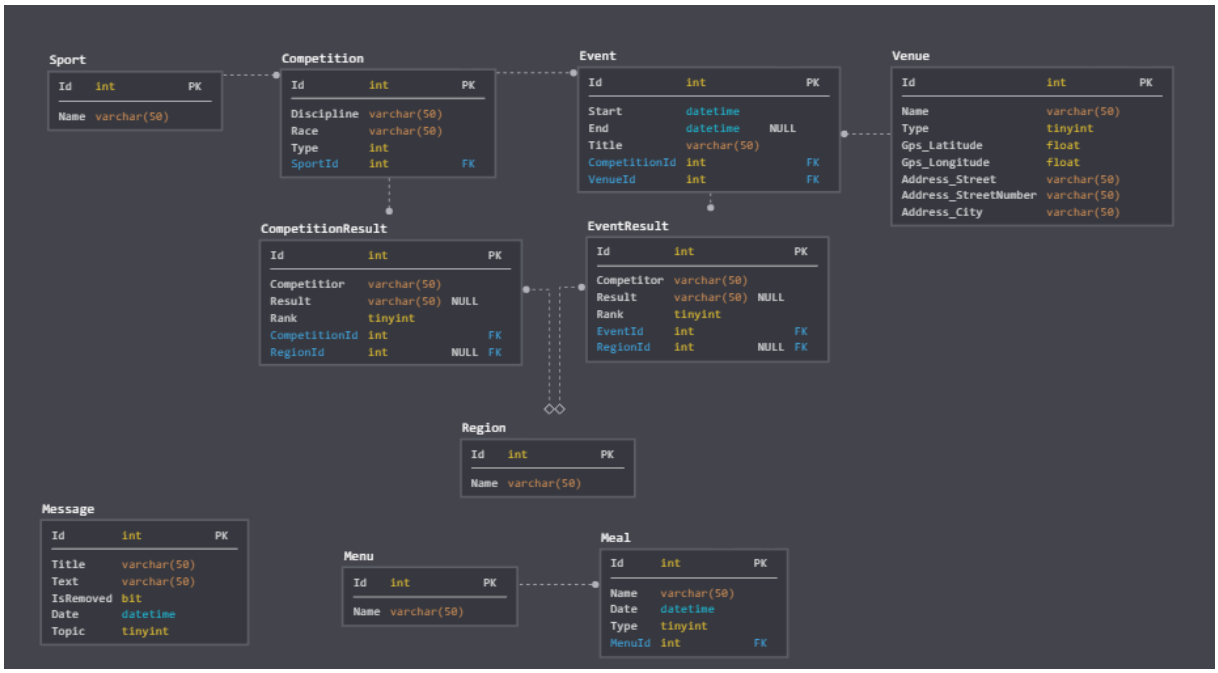
Obsahuje definice tříd, enumů a structů, jež se používají napříč webovou službou a mobilní aplikací.

3.1 Důležité části

1. Models.DataGroups – Třídy definované v této složce slouží k přenosu dat mezi webovou službou a mobilní aplikací. Serializují se do formátu XML/JSON.
2. Models – Všechny datové třídy obsahují implementaci interface IEqualityComparer. Ten slouží k zjištění rozdílů mezi různými verzemi datových souborů DataGroups, využívá se pro extension metodu Enumerable.Except.

3.2 Schéma datových tříd

Schéma využitých datových tříd vypadá následovně:



Obrázek 3 – Schéma datových tříd

Popis datových tříd

1. Sport – Sport konající se na ODM (tenis)
2. Competition – Soutěž ve sportu (dvouhra – mladší dívky)
3. CompetitionResult – Výsledek soutěže
4. Event – Zápas spadající pod soutěž (Novák – Čech)
5. EventResult – Výsledek zápasu
6. Venue – Ubytování / místo konání sportu
7. Region – Kraj účastnící se olympiády
8. Message – Zprává odeslaná uživateli
9. Menu – Jídelníček
10. Meal – Obsah jídelníčku

4 Webová služba

Kapitola popisuje strukturu webové služby.

Webová služba primárně slouží jako zdroj dat pro mobilní klienty, její další funkcí je tvorba a správa dat.

Je rozdělena na následující vrstvy.

1. ODM.Backend.Models – Tento projekt obsahuje definice tříd, které webová služba sdílí s mobilní aplikací.
2. ODMWeb.DL – Datová vrstva aplikace, obsahuje definice entit a DbContext pro EntityFramework Core. Je vytvořená pomocí návrhového vzoru Repository a UnitOfWork.
3. ODMWeb – Tento projekt obsahuje logiku a uživatelské rozhraní webové služby.

4.1 Datová vrstva

Datová vrstva webové služby implementuje návrhové vzory Repository a UnitOfWork. Databázový model je vytvořen pomocí Entity Framework code-first modelování. DbContext také obsahuje záznamy o datových souborech serializovaných v pracovním adresáři webové služby.

4.1.1 Repository

Jednotlivé repository zapouzdřují metody určené k práci se specifickými entitami v EF DbContextu.

Repository patřící entitám, které se stahují z cizího API implementují interface IForeign. Ten slouží k aktualizaci modelů v databázi pomocí modelů stažených z cizího API. Jedná se o tyto repository:

1. CompetitionRepository
2. CompetitionResultRepository
3. EventRepository
4. EventResultRepository
5. RegionRepository
6. SportRepository
7. VenueRepository

Metody definované v interface IForeign využívají vlastní vyjímky, které jsou vyhozeny v případě, že není možné vyřešit databázové závislosti – závislost entity v databázi chybí. Tyto vyjímky jsou potřebné ke správnému logování a ovládání chování Hangfire úkolů.

```
internal class Repository<TEntity> : IRepository<TEntity> where TEntity : BaseEntity
{
    protected readonly OdmDbContext dbContext;

    public Repository(OdmDbContext dbContext)...

    public Task<TEntity> GetById(int id)
    {
        return this.dbContext.Set<TEntity>().FindAsync(id);
    }

    public void Add(TEntity entity)...

    public void AddRange(IEnumerable<TEntity> entities)...
```

Obrázek 4 – Generické repository

```
internal class CompetitionRepository : Repository<CompetitionEntity>, ICompetitionRepository<CompetitionDependencies>
{
    public CompetitionRepository(OdmDbContext dbContext) : base(dbContext)
    {
    }

    public async Task<CompetitionEntity> GetCeremonial()...

    public async Task<CompetitionEntity> GetEntertainment()...

    public async Task<int> UpdateFromForeign(List<ForeignCompetition> foreignCompetitions)
    {

        foreach (var comp in foreignCompetitions)
        {
            var dependencies = await UpdateSingleFromForeign(comp);
            if (!dependencies.Result)
            {
                throw new SportDependencyNotResolvedException(dependencies.ForeignSportId.ToString());
            }
        }

        return
            await dbContext.SaveChangesAsync();
    }

    public async Task<CompetitionDependencies> UpdateSingleFromForeign(ForeignCompetition foreignItem)...
}
```

Obrázek 5 – Specifické repository

4.1.2 UnitOfWork

Všechny repository jsou zapouzdřeny ve třídě UnitOfWork, aby nebyl možný přístup k samotným repository z ostatních částí projektu, jsou nastavené jako internal. Jejich funkcionalita je do zbytku projektu dostupná pouze skrz třídu UnitOfWork, tím je ošetřena práce s DbContextem a databázová vrstva je zapouzdřena do jedné třídy, která se sama stará o správné likvidování připojení k databázi.

```
public class UnitOfWork : IUnitOfWork, IDisposable
{
    private readonly OdmDbContext context;

    public IVenueRepository Venues { get; }
    public IRepository<MealEntity> Meals { get; }
    public IRepository<MenuEntity> Menus { get; }
    public IRegionRepository Regions { get; }
    public ISportRepository Sports { get; }
    public IRepository<MessageEntity> Messages { get; }
    public ICompetitionResultRepository CompetitionResults { get; }
    public IEventResultRepository EventResults { get; }
    public SerializationRecordRepository SerializedFiles { get; set; }
    public ICompetitionRepository<CompetitionDependencies> Competitions { get; set; }
    public IEventRepository<EventDependencies> Events { get; set; }

    public UnitOfWork(OdmDbContext context)...

    public async Task<int> Save()
    {
        var result = await this.context.SaveChangesAsync();
        return result;
    }

    public void Dispose()...
}
```

Obrázek 6 – Příklad UnitOfWork

4.2 ODMWeb

Webová služba je psaná podle návrhového vzoru MVC a implementuje RESTful API. Umožňuje tvorbu a úpravu dat, automaticky plánované stahování dat z cizího API a automatickou publikaci datových souborů pro mobilní klienty. Veškeré důležité dění je v aplikaci logováno, tyto logy se dále dají zobrazit přímo v aplikaci.

Přístup do služby je ošetřen pomocí technologie Identity. Běžným uživatelům mobilní aplikace tak webová služba není přístupná. REST API endpointy jsou nezabezpečené, jelikož neodesílají citlivá data.

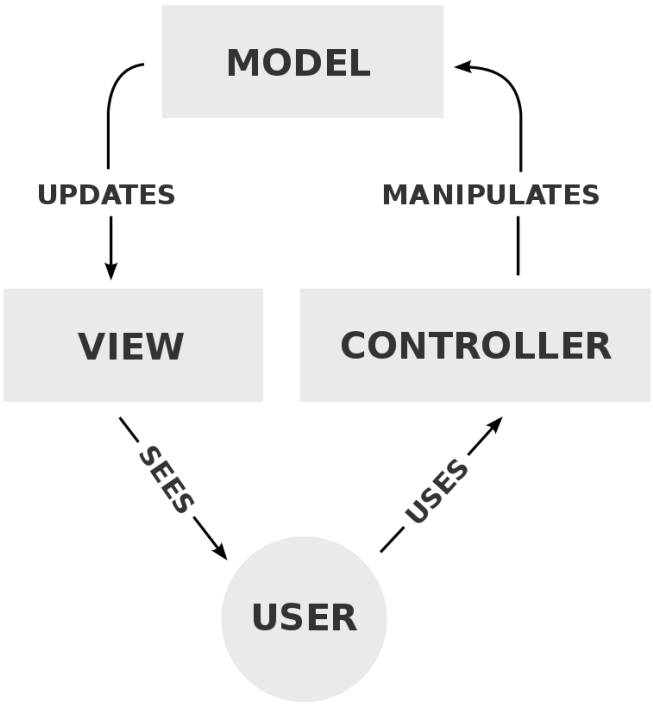
4.2.1 Administrační část

Aplikace umožňuje tvorbu dat, jež nejsou přístupná z cizího API. Jedná se o následující data.

1. Menu – jídelníčky
2. Message – zprávy uživatelům
3. Ceremonial – oficiální ceremoniály
4. Entertainment – doprovodný program

Ceremoniály a doprovodný program využívají modely Event, která mají relaci k instancím Competition, jelikož je struktura dat ceremoniálů a doprovodného programu stejná, jako sportovního programu.

Uživatelské rozhraní administrační části je vytvořeno pomocí návrhového vzoru MVC.



Obrázek 7 – Návrhový vzor MVC (dostupné z https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/1024px-MVC-Process.svg.png)

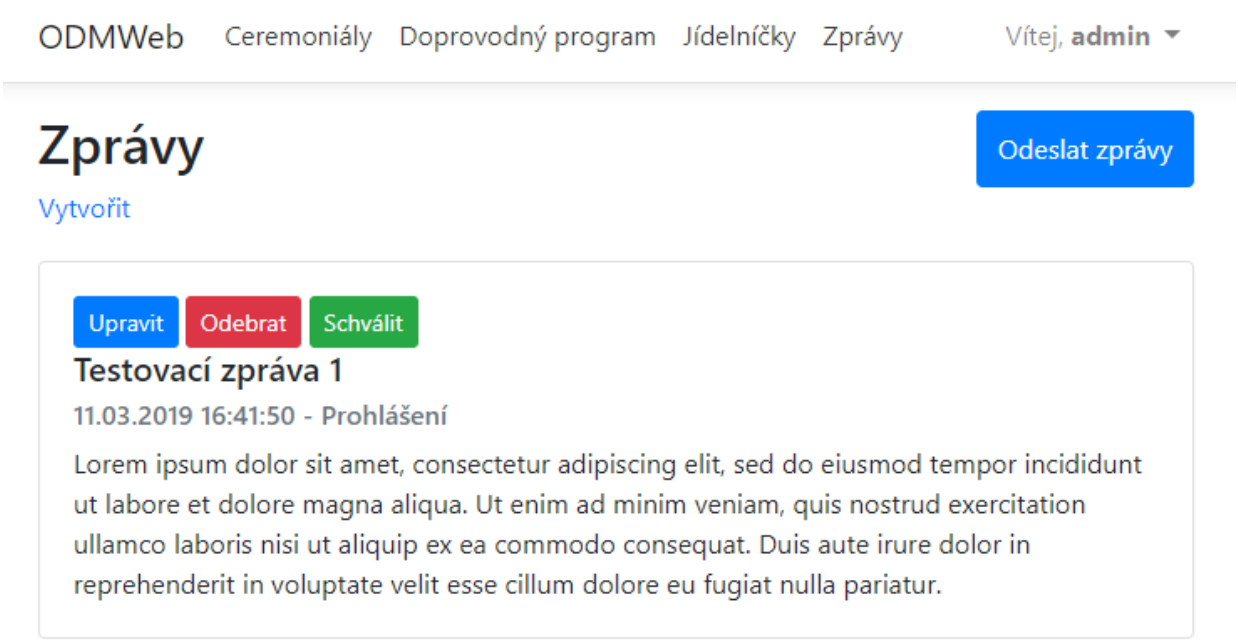
Jelikož bude mít k webové službě přístup více lidí, je nutné zabezpečit vytváření dat před nechtěnými chybami. Byl proto implementován následující lifecycle.

Obrázek 8 – Lifecycle modelů ve webové službě

1. Created – Data byla vytvořena, v tomto stavu je lze schválit k publikaci, smazat a upravit.
2. Approved – Data byla schválena k publikaci, tuto akci může provést pouze správce s dostatečným oprávněním, v tomto stavu je lze publikovat uživatelům, nebo vrátit do stavu created.
3. Published – Data byla publikována uživatelům, v tomtu stavu s nimi již nelze manipulovat.

Tato funkcionalita je implementována v následujících controllerech.

1. MenuController
2. MessageController
3. CeremonialsController
4. EntertainmentController



Obrázek 9 – Příklad uživatelského rozhraní webové služby

4.2.2 Publikační část

Publikační část zahrnuje REST API a třídy obsluhující serializaci dat do verzovaných XML souborů, pro lokální uložení byl zvolen formát XML kvůli vyšší rychlosti serializace a deserializace než u formátu JSON.

4.2.2.1 REST API

Webová služba dodává data do mobilních klientů pomocí open endpoint RESTful API. Data jsou přenášena v JSON řetězcích, které jsou vytvořeny serializací následujících tříd:

1. MessageAppData – Jedná se o zprávy odeslané uživatelům. (model Message)
2. ProgramAppData – Jedná se o sportovní program, doplňkový program a ceremoniály. (modely Competition a Event)
3. ResultAppData – Jedná se o výsledky soutěží a zápasů.
   (modely CompetitionResult a EventResult)
4. StaticAppData - Jedná se o data, která by měla být v průběhu ODM neměnná
   (modely Sport, Venue, Menu, Meal, Region)

Jednotlivé datové třídy jsou serializovány v XML souborech uložených v pracovním adresáři webové služby ve složce App_Data. Každý soubor obsahuje publikovaná data v čas serializace, na serveru tak vzniká mnoho různých verzí jednotlivých souborů.

Aby se uživatelům šetřil datový plán, REST API odesílá pouze rozdíly dvou souborů (změny). V každém GET requestu pro určitý soubor se specifikuje verze dat, kterou uživatel má lokálně uloženou a webová služba vrátí rozdíl pouze provedené změny. Aby se předešlo odesílání požadavků, na které by se odpovědělo prázdným souborem (žádné změny nebyly provedeny), byl implementován endpoint, který vrátí poslední dostupné verze souborů na serveru.

Samozřejmě je možné, že by serializovaný XML soubor mohl být smazán nebo poškozen z různých důvodů (chyba, nedostatek místa na hostingu). Vyhledávání posledního dostupného souboru je tedy rekurzivní. Získá se poslední dostupný záznam o souboru z databáze a zjistí se, zda cesta odkazuje na platný soubor, pokud ne, tento záznam se označí jako neplatný a opět se vybere nejnovější dostupný soubor.

```
[HttpGet("static/{remoteVersion}")]
public async Task<StaticAppData> Static(int remoteVersion)
{
    StaticAppData response = new StaticAppData();

    var type = SerializationRecordType.Static;

    var clientVersionRecord =
        await unitOfWork.SerializedFiles.GetByVersionAndType(remoteVersion, type);
    var lastServerVersionRecord =
        await unitOfWork.SerializedFiles.GetLastAvailableByType(type);

    if (lastServerVersionRecord?.Version == clientVersionRecord?.Version)
    {
        return response;
    }

    StaticApiProvider provider = new StaticApiProvider(this.unitOfWork);
    var clientAppData = await provider.GetData(clientVersionRecord) ?? new StaticAppData();
    var serverAppData = await provider.GetLastAvailableData(lastServerVersionRecord) ?? new StaticAppData();

    if (clientAppData?.Version == serverAppData?.Version)
    {
        return response;
    }

    response = provider.GetChanges(clientAppData, serverAppData);

    return response;
}
```

Obrázek 10 – API endpoint pro aktualizaci statických dat

Endpointy REST API jsou routovány následovně:

1. /api/data/{soubor}/{verze} – Tento endpoint vrátí JSON, který obsahuje rozdíl mezi poslední verzí serializovanou na serveru a verzí specifikovanou parametrem „verze“. Parametr „soubor“ určuje na jaká data se uživatel dotazuje (message/static/program/result).
2. /api/data/state – Tento endpoint vrátí JSON, ve kterém jsou uvedené čísla posledních verzí souborů.

Tato funkcionalita je zapouzdřená ve třídách nacházejících se v namespace ODMWeb.DataProviding.ApiProviders.

4.2.2.2 Tvorba verzovaných souborů

Každý datový soubor je serializovaný ze záznamů v databázi, které jsou schválené k publikaci, nebo jsou získané prostřednictvím cizího API.

Datové soubory jsou uloženy do pracovního adresáře aplikace do složky App_Data, o každém souboru je veden záznam v databázi, který zahrnuje cestu k němu, verzi, jaký správce ho vytvořil a čas serializace.

Logika této části je implementovaná v namespace ODMWeb.DataProviding.Publishers. Veškerá publikování datových souborů je implementováno pomocí úkolů pro Hangfire službu, důvodem je sběr statistik běhu úkolů, kterou Hangfire server automaticky provádí.

Serializace nových verzí souborů Program a Result se v aktuální verzi provádí automaticky každou hodinu, pro produkční verzi aplikace se tato funkce nejspíš provede vždy po stažení nových dat.

Serializaci souborů lze spustit i manuálně, tato funkce je dostupná přes controller PublishOdmDataController.

```
public class MessageDataPublisher : AbsDataPublisher
{
    public MessageDataPublisher(IHostingEnvironment environment, IMapper mapper, IUnitOfWork unitOfWork) ...

    public override async Task<bool> Run(string userName)
    {
        var serializer = new MessageDataSerializer();
        var entityData = await serializer.CreateEntityGroup(this.UnitOfWork);
        var modelData = serializer.CreateDataGroup(this.Mapper, entityData);
        var version = await UnitOfWork.SerializedFiles.GetLastVersionByType(SerializationRecordType.Message) + 1;
        var path = Path.Combine(FolderPath, Config.GetMessageVersionFileName(version));

        var record = await UnitOfWork.SerializedFiles.Add(new SerializationRecordEntity(version, SerializationRecordType.Message, path, userName));

        bool result = serializer.SerializeData(modelData, path, version);
        if (result)
        {
            await UnitOfWork.SerializedFiles.SetIsAvailable(record, true);
            await serializer.SetEntitiesPublished(entityData, UnitOfWork);
            Logger.Warning("PublishMessageData successful by user: {Name}. Record: {@Record}", userName, record);
        }
        else
        {
            Logger.Error("PublishMessageData failed by user: {Name}. Record: {@Record}", userName, record);
        }
        return result;
    }
}
```

Obrázek 11 – Příklad data publisheru

4.2.2.3 Automatická aktualizace dat

Automatická aktualizace dat z cizího API je prováděná pomocí opakujících se úkolů pro Hangfire server, časový harmonogram aktualizace je definován pomocí CRON výrazu. Přístup k Hangfire dashboardu, kde se nachází informace o úkolech a jejich manuální ovládání je umožněno pouze hlavnímu administrátorovi.

Firma eSport k datu odevzdání této práce API stále vyvíjí a o průběhu vývoje mě průběžně informuje. Webová služba je aktuálně napojena na testovací API, jež generuje data Competition a Event, data CompetitionResult a EventResult generovaná nejsou, jelikož nejsou důležité pro aktuální stádium vývoje, zbytek prozatimních dat je staticky vytvořen v daných třídách aplikace.

DTO modely a připojení k databázi jsou definované v namespace ODMWeb.ForeignApiManager. Třída WebService zapouzdřuje stahování všech dat z cizího api.

Opakované úkoly pro Hangfire server jsou definované v namespace ODMWeb.Hangfire.RecurringJobs.

U úkolů, které aktualizují data s relací k jiným záznamům (UpdateCompetitionDataTask, UpdateEventDataTask) je ošetřená chyba vyřešení závislostí. Pokud metody definované v repository vyhodí vyjímku o nemožnosti vyřešení závislostí, úkol se terminuje a dle vyhozené vyjímky se pokusí s cizího API nejdříve aktualizovat záznamy potřebné k vyřešení chybících závislostí. Po minutě se terminovaný úkol pustí znovu.

V aktuální verzi projektu se jedná o následující vyjímky:

CompetitionDependencyNotResolvedException – Pro danou akci (Event) chybí soutěž (Competition).

SportDependencyNotResolvedException – Pro danou soutěž (Competition) chybí sport.

VenueDependencyNotResolvedException – Pro danou akci (Event) chybí místo konání (Venue).

Tyto vyjímky jsou ošetřené a logované, aby došlo k terminaci úkolu, je nutné po jejich vyskutnutí vyhodit jakoukoliv neošetřenou vyjímku.

```
public class UpdateCompetitionDataTask : BaseHangfireTask<UpdateCompetitionDataTask>
{
    [AutomaticRetry(DelaysInSeconds = new[] { 60 })]
    public override async Task Run()
    {
        logger.Information("UpdateCompetitionDataTask start.");
        var service = new WebService();
        var data = await service.GetCompetitions();
        try
        {
            await this.unitOfWork.Competitions.UpdateFromForeign(data);
            logger.Information("UpdateCompetitionDataTask successful.");
        }
        catch (SportDependencyNotResolvedException ex)
        {
            logger.Error(ex, "UpdateCompetitionDataTask failed. Sport dependency not resolved.");
            BackgroundJob.Enqueue<UpdateSportDataTask>(t => t.Run());
            logger.Error("Stopping UpdateCompetitionDataTask execution.");
            throw new InvalidOperationException();
        }
    }

    public UpdateCompetitionDataTask(IUnitOfWork unitOfWork) ...
}
```

Obrázek 12 – Příklad Hangfire úkolu

```
RecurringJob.AddOrUpdate<UpdateCompetitionDataTask>(t => t.Run(), "0 */10 * ? * *", TimeZoneInfo.Local);
RecurringJob.AddOrUpdate<UpdateEventDataTask>(t => t.Run(), "0 */10 * ? * *", TimeZoneInfo.Local);
RecurringJob.AddOrUpdate<ProgramDataPublisher>(t => t.Run("hangfire"), "0 * * * *", TimeZoneInfo.Local);
RecurringJob.AddOrUpdate<ResultDataPublisher>(t => t.Run("hangfire"), "0 * * * *", TimeZoneInfo.Local);
RecurringJob.AddOrUpdate<CheckFileAvailabilityTask>(t => t.Run(), "0 */30 * ? * *", TimeZoneInfo.Local);
```

Obrázek 13 – Registrace opakovaných Hangfire úkolů

4.2.2.4 Logování

Veškeré důležité dění je v aplikaci asynchronně logováno. Pro každý den je vytvářen nový Log soubor v pracovním adresáři webové služby ve složce Logs.

Do logů se zapisují veškeré akce, které vykonal administrátor (tvorba dat, schválení, publikace) a průběh veškerých automaticky vykonaných akcí (tvorba datových souborů, aktualizace databáze, kontrola dostupnost souborů)

Tyto logy si může administrátor zobrazit přímo ve webové službě, nebo stáhnout. Slouží k tomu LogController.

```
2019-03-11 15:40:43.170 +01:00 [INF] UpdateCompetitionDataTask start.
2019-03-11 15:40:43.174 +01:00 [INF] Attempting GET request from: http://fake-api-pro-ceckyho.herokuapp.com/api/competitions.
2019-03-11 15:40:43.608 +01:00 [INF] GET request from: http://fake-api-pro-ceckyho.herokuapp.com/api/competitions successful.
2019-03-11 15:40:43.648 +01:00 [ERR] UpdateCompetitionDataTask failed. Sport dependency not resolved.
ODMWeb.DL.Exceptions.SportDependencyNotResolvedException: Sport dependency not available in current context. Id:3
   at ODMWeb.DL.Repositories.CompetitionRepository.UpdateFromForeign(List`1 foreignCompetitions) in D:\Repositories\ODMWeb\ODMWeb.DL\Repositor
   at ODMWeb.Hangfire.RecurringJobs.UpdateCompetitionDataTask.Run() in D:\Repositories\ODMWeb\ODMWeb\Hangfire\RecurringJobs\UpdateCompetitionDa
2019-03-11 15:40:43.654 +01:00 [ERR] Stopping UpdateCompetitionDataTask execution.
2019-03-11 15:40:43.657 +01:00 [INF] UpdateSportDataTask start.
2019-03-11 15:40:43.769 +01:00 [INF] Pulling fake Sport data.
2019-03-11 15:40:43.769 +01:00 [INF] UpdateSportDataTask successful.
2019-03-11 15:40:59.807 +01:00 [INF] UpdateCompetitionDataTask start.
2019-03-11 15:40:59.807 +01:00 [INF] Attempting GET request from: http://fake-api-pro-ceckyho.herokuapp.com/api/competitions.
2019-03-11 15:41:00.204 +01:00 [INF] GET request from: http://fake-api-pro-ceckyho.herokuapp.com/api/competitions successful.
2019-03-11 15:41:00.247 +01:00 [INF] UpdateCompetitionDataTask successful.
```

Obrázek 14 - Příklad strukturovaného logu

V této části logu lze vidět selhání automatické služby pro stažení dat soutěží. Stažená data nemají v databázi potřebné relace, proto se služba přeruší a nejprve se aktualizují potřebné relace.

4.2.2.5 Automatická kontrola dostupnosti souborů

Ikdyž při získávání serializovaných datových souborů jsou ApiProvider třídy schopné samy zjistit, zda je soubor na serveru dostupný, pro zrychlení dotazů běží každých 30 minut automatická kontrola dostupnosti všech záznamů o aktuálně dostupných souborech skrze službu Hangfire.

Tento úkol je definován ve třídě CheckFileAvailabilityTask.

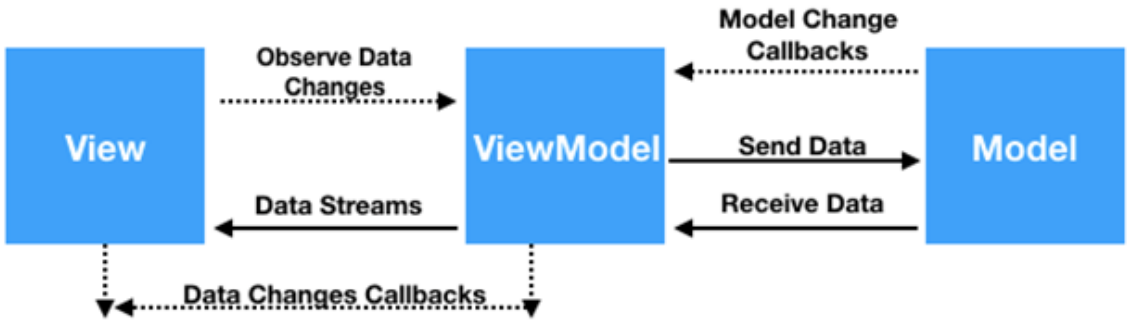
5 Mobilní aplikace

Kapitola popisuje strukturu mobilní aplikace.

Mobilní aplikace je multiplatformní a je vytvořena pomocí návrhového vzoru MVVM. Spustitelná je na zařízeních iOS a Android.

Plně podporuje verzi iOS 9 a výše a verzi Android 4.2.2 a výše.

Návrhový vzor MVVM



Obrázek 15 – Návrhový vzor MVVM (dostupné z https://cdn.journaldev.com/wp-content/uploads/2018/04/android-mvvm-pattern.png)

Veškeré dění v aplikaci je logováno, v případě fatální chyby existuje možnost odeslání chybového hlášení na server. To je potřebné pro opravu chyb, které se dějí pouze na určitých verzích operačního systému a určitých zařízení.

Aplikace je rozdělena na následující vrstvy.

1. ODM.Backend.Models – Tento projekt obsahuje definice tříd, které mobilní aplikace sdílí s webovou službou.
2. ODM.Backend – Tento projekt obsahuje třídy určené pro práci s webovou službou a interním uložištěm aplikace.
3. ODM.Droid – Tento projekt je vstupní bod pro zařízení Android, zde je implementována platformní funkcionalita.
4. ODM.iOS – Tento projekt je vstupní bod pro zařízení iOS, zde je implementována platformní funkcionalita.
5. ODM – V tomto projektu je implementována hlavní logika aplikace, uživatelské rozhraní a je zde sjednocena multiplatformní funkcionalita.

5.1 Backend mobilní aplikace

Backend mobilní aplikace je definovaný v projektu ODM.Backend. Slouží ke komunikaci s cizím API, práci s interním uložištěm a k nastavení aplikace. Veškerá práce s daty je implementována v „manager“ třídách, každá manager třída se stará o specifickou část backendu aplikace.

5.1.1 SettingsManager

Třída se stará o práci s platformním Key-Value uložištěm (známé jako SharedPreferences v platformě Android). Využívá se k ukládání a čtení důležitých informací jako verze datových souborů, serializovaných notifikací a lokace bydliště.

5.1.2 DataUpdateManager

Třída se stará o komunikaci s webovou službou.

```
public async Task CheckOrUpdateData()
{
    var state = await this.GetFileState();
    if (state.LastStaticFileVersion != 0)
    {
        await this.UpdateStaticData(state.LastStaticFileVersion);
    }
    if (state.LastMessageFileVersion != 0)
    {
        await this.UpdateMessageData(state.LastMessageFileVersion);
    }
    if (state.LastProgramFileVersion != 0)
    {
        await this.UpdateProgramData(state.LastProgramFileVersion);
    }
    if (state.LastResultFileVersion != 0)
    {
        await this.UpdateResultsData(state.LastResultFileVersion);
    }
}

internal async Task<bool> UpdateProgramData(int version)
{
    var repository = new ODMWebRepository();
    int localVersion = SettingsManager.Instance.ProgramVersion;
    var needsUpdate = localVersion < version;
    if (needsUpdate)
    {
        var data = await repository.GetProgramData(localVersion);

        if (data != null)
        {
            var dataVersion = await DataManager.UpdateProgramData(data);
            if (version != 0)
            {
                SettingsManager.Instance.ProgramVersion = dataVersion;
                return true;
            }
            return false;
        }
    }
    return false;
}
```

Obrázek 16 – Hlavní funkce DataUpdateManageru

Nejprve zjistí poslední verze souborů serializovaných na serveru. Pokud se na serveru nachází nová verze nějakého souboru, odešle se požadavek o změny mezi lokálně uloženými daty a novými daty na serveru. Odpověď serveru pak DataUpdateManager předá třídě DataManager, která je uloží do lokálního uložiště aplikace.

5.1.3 DataManager

Třída se stará o ukládání a čtení dat z interního uložiště aplikace. Po zapnutí tato třída načte veškerá data do statické paměti, to je nutné z důvodu rezponzivnosti aplikace.

Obsahuje tři druhy metod.

1. Update metody – Starají se o aktualizaci interního uložiště.
2. Save metody – Uloží data do interního uložiště ve formátu JSON.
3. Load metody – Deserializují JSON soubory z interního uložiště do statických objektů.

Tato třídá se chová jako data provider pro celou aplikaci, je nad ní postaven repository pattern podobný tomu na serveru.

5.1.4 Manager třídy pro datové modely

Tyto třídy jsou velmi podobné návrhovému vzoru Repository. Pracují se statickými objekty ve třídě DataManager a objekty v nich načtené filtrují dle potřeby.

```
public class EventResultManager : AbsManager
{
    public async Task<List<EventResult>> GetAll()
    {
        try
        {
            var result = (await DataManager.GetResultData()).EventResults;
            return result;
        }
        catch (Exception exception)
        {
            await this.HandleException(exception);
            return null;
        }
    }

    public async Task<List<EventResult>> GetByEvent(Event eventModel)
    {
        var all = await GetAll();
        var results = all.Where(r => r.EventId == eventModel.Id).ToList();
        return results;
    }
}
```

Obrázek 17 – Manager třída pro datové modely

5.2 Android část aplikace

Projekt ODM.Droid je hlavním vstupním bodem pro Android zařízení. Je zde složka Resources, která obsahuje veškerou grafiku v aplikaci pro platformu Android a implementace platformních funkcí.

Spuštění Xamarin.Forms Android aplikace lze znázornit takto.

Obrázek 18 – Spuštění Xamarin.Forms Android aplikace

Při startu aplikace se nejprve zobrazí splashscreen stránka, poté se spustí hlavní aktivita aplikace, která po instancování všech potřebných funkcí spustí Xamarin.Forms aplikaci.

5.2.1 Android Dependency Injection

Pro využití platformě specifických funckí je v Xamarin.Forms nutné využít tzv. Dependency Injection. Následující kód se stará o správu notifikací.

```
[assembly: Dependency(typeof(Skeleton.Mobile.ODM.Droid.DependencyServices.Notifications))]
namespace Skeleton.Mobile.ODM.Droid.DependencyServices
{
    public class Notifications : INotifications
    {
        readonly AlarmManager alarmManager;
        public static int AppIconResourceId { get; set; }

        static Notifications()...

        public Notifications()...

        public async Task Send(Backend.Models.Notification notification)
        {
            await this.alarmManager.ScheduleNotification(notification);
        }

        public Task CancelByNotificationId(int notificationId)
        {
            this.CancelByNotificationIdInternal(notificationId);
            return Task.FromResult(true);
        }

        public Task<bool> RequestPermission() => Task.FromResult(true);

        protected virtual void CancelByNotificationIdInternal(int notificationId)...
    }
}
```

Obrázek 19 – Android Dependency Injection

Task RequestPermission je nutné implementovat kvůli iOS části aplikaci, která ji vyžaduje.

5.3 iOS část aplikace

Projekt ODM.iOS je hlavním vstupním bodem pro zařízení s operačním systémem iOS. Struktura je podobná Android části, funkcionalita je stejná.

5.3.1 iOS Dependency Injection

```
[assembly: Dependency(typeof(Skeleton.Mobile.ODM.iOS.DependencyServices.Notifications))]
namespace Skeleton.Mobile.ODM.iOS.DependencyServices
{
    public class Notifications : INotifications
    {
        readonly INotifications impl;

        public Notifications()
        {
            if (UIDevice.CurrentDevice.CheckSystemVersion(10, 0))
                this.impl = new UNNotificationsImpl();
            else
                this.impl = new UILocalNotificationsImpl();
        }

        public Task CancelByNotificationId(int notificationId) => this.impl.CancelByNotificationId(notificationId);
        public Task Send(Notification notification) => this.impl.Send(notification);
        public Task<bool> RequestPermission() => this.impl.RequestPermission();
    }
}
```

Obrázek 20 – iOS Dependency Injection

5.4 Společná část aplikace

Projekt ODM obsahuje sdílený kód pro obě platformy vytvořený pomocí frameworku Xamarin.Forms. Je zde implementovaná navigace aplikací, uživatelské rozhraní a logika.

Tato část mobilní aplikace je vytvořena podle návrhového vzoru MVVM, uživatelské rozhraní je vytvořené pomocí technologie XAML, struktura je tedy velmi podobná aplikacím vytvořeným pomocí technologie WPF (Windows Presentation Foundation).

5.4.1 Dependency Injection

Dependency se nejprve musí definovat pomocí interface ve společné části projektu.

```
public interface INotifications
{
    /// <summary>
    /// Potřebné pro iOS na notification permission request
    /// </summary>
    Task<bool> RequestPermission();

    /// <summary>
    /// Zrušení specifické notifikace podle jejího id
    /// </summary>
    /// <param name="notificationId">Jednoznačné id notifikace</param>
    Task CancelByNotificationId(int notificationId);

    /// <summary>
    /// Odeslání okamžité / naplánované notifikace
    /// </summary>
    /// <param name="notification"></param>
    Task Send(Notification notification);
}
```

Obrázek 21 – Interface definující Xamarin.Forms platformní dependency

Platformní funkcionalita definovaná v iOS nebo Android projektu se poté získá pomocí metody Xamarin.Forms.DependencyService.Get<INTERFACE>()

```
public static class CrossNotifications
{
    private static INotifications current;
    public static INotifications Current
    {
        get
        {
            if (current == null)
            {
                current = Xamarin.Forms.DependencyService.Get<INotifications>();
            }
            return current;
        }
    }
}
```

Obrázek 22 – Příklad získání dependency jako singleton

5.4.2 Navigace aplikací

Navigaci Xamarin.Forms aplikací je vyřešena pomocí tzv. navigačního stacku.

Ve spod navigačního stacku je vždy kořenová stránka, na tu se vykreslují stránky, na které se uživatel naviguje.

Obrázek 24 – Navigace na nové stránky (dostupné z https://docs.microsoft.com/en-gb/xamarin/xamarin-forms/app-fundamentals/navigation/hierarchical-images/pushing.png)

Obrázek 23 – Navigace na předcházející stránky v navigačním stacku (dostupné z https://docs.microsoft.com/en-gb/xamarin/xamarin-forms/app-fundamentals/navigation/hierarchical-images/popping.png)

Ve většině případů není vhodné na navigační stack pushovat více než 4 stránky kvůli uživatelské přívětivosti, z tohoto důvodu jsem v aplikaci využil tzv. popup stránky, které kompletně nepřekryjí poslední stránku v navigačním stacku, popup stránky využívají vlastní navigační stack.

V platformách Android a iOS není ošetřený „multi-click“, je proto nutné vícenásobné přesměrování ošetřit sledováním poslední stránky navigačním stacku. Tuto chybu je třeba ošetřit i v Xamarin.Forms aplikacích.

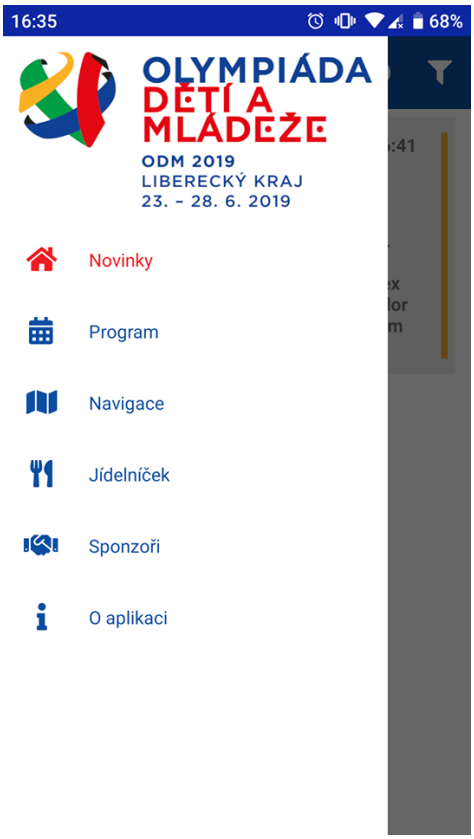
```
/// <summary>
/// Slouzi k bezpecnemu presmerovani na stranku, napriklad z listview, kdy se muze vyvolat dvakrat za sebou
/// </summary>
/// <param name="page"></param>
public static void SafeGoToPage(Page page)
{
    var stack = RootPage.RootNavigationPage.Navigation.NavigationStack;
    if (stack[stack.Count - 1].GetType() != page.GetType())
    {
        RootPage.RootNavigationPage.PushAsync(page);
    }
}
/// <summary>
/// viz. SafeGoToPage, jen pro Popup stránky
/// </summary>
/// <param name="page"></param>
public static void SafeGoToPopupPage(PopupPage page)
{
    var stack = PopupNavigation.Instance.PopupStack;

    if (stack.Count != 0)
    {
        if (stack[stack.Count - 1].GetType() != page.GetType())
        {
            PopupNavigation.Instance.PushAsync(page);
        }
    }
    else
    {
        PopupNavigation.Instance.PushAsync(page);
    }
}
```

Obrázek 25 – Ošetření multi-click navigace pro standartní a popup stránky

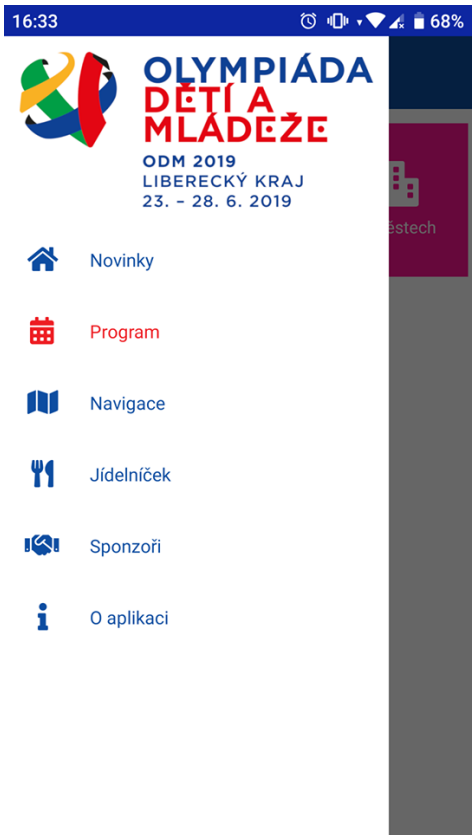
5.4.3 Postranní menu

Pro vytvoření postranního menu se v Xamarin.Forms využívá tzv. Master-Detail Page, definovaná je ve třídě Xamarin.Forms.MasterDetailPage. Stránka, která se v navigačním stacku využívá jako root page (nultá stránka) tuto třídu dědí.

Master-Detail page má dvě hlavní vlastnosti – Master page a Detail page. Jako Master page se nastaví stránka s tlačítky pro navigaci po aplikaci. Tato stránka dále slouží jako vysouvací menu. Tlačítka pro navigaci dále nastavují vlastnost Detail na stránku, která odpovídá zvolenému tlačítku. Tato stránka se poté vykreslí.



Obrázek 27 – Android master-detail page se zvolenou stránkou Novinky



Obrázek 26 – Android master-detail page se zvolenou stránkou Program

5.5 Ukázka aplikace

Kapitola se zabývá ukázkou funkcí aktuální verze aplikace.

Stávající design uživatelského rozhraní není konečný a aktivně se pracuje na novém.

Ikonky využité v aplikace jsou buď z ikonkového fontu FontAwesome, z balíčku MaterialDesign.

Mnohá data nahraná v aplikaci jsou vytvořena pomocí náhodného generovaní textových řetězců, proto nedávají v některých částech aplikace žádný smysl.

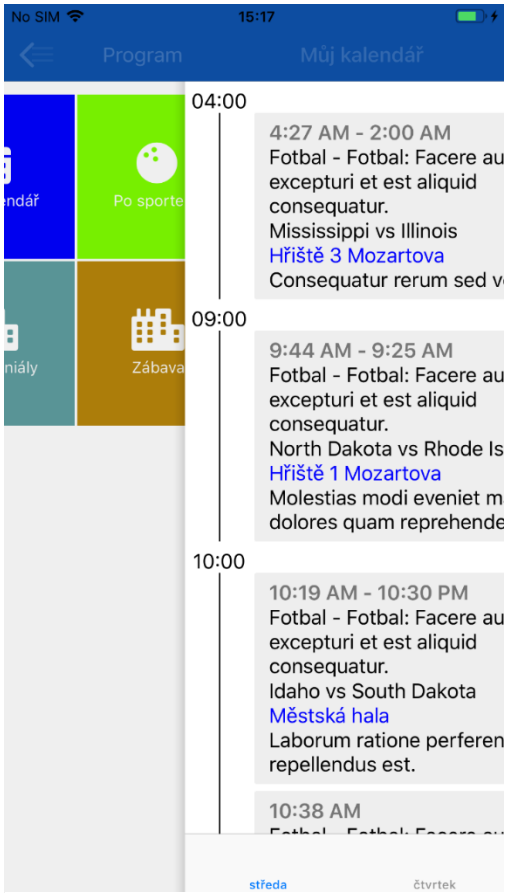
5.5.1 Splashscreen

Obrázek 28 – Android splashscreen

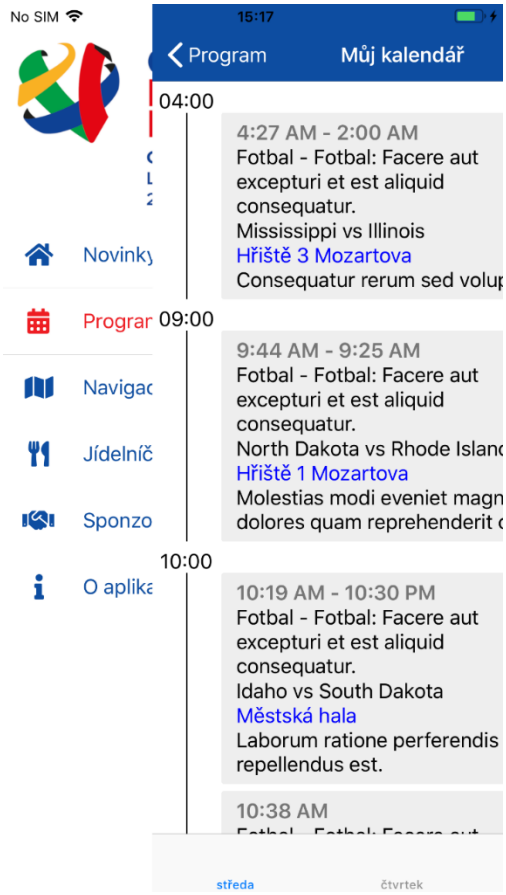
Obrázek 29 – iOS splashscreen

5.5.2 Nativní navigace

Navigace aplikací je plně nativní, uživatelé Android a iOS mají k dispozici ovládací prvky, na které jsou zvyklí. Android navigace je ukázána v obrázcích 26 a 27.



Obrázek 30 – iOS nativní navigace na stránku zpět

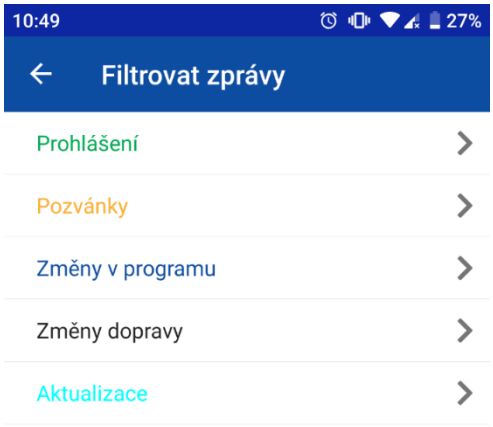


Obrázek 31 – iOS nativní navigace na postranní menu

5.5.3 Novinky

Tato stránka uživateli dovoluje zobrazit si zprávy odeslané pořadateli Her. Ikonka vykřičníku zobrazí pouze důležité zprávy (červený nadpis), ikonka trychtýře zobrazí filtry zpráv dle předmětu.

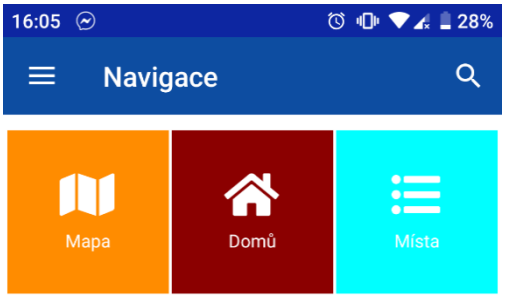
Obrázek 32 – Seznam zpráv



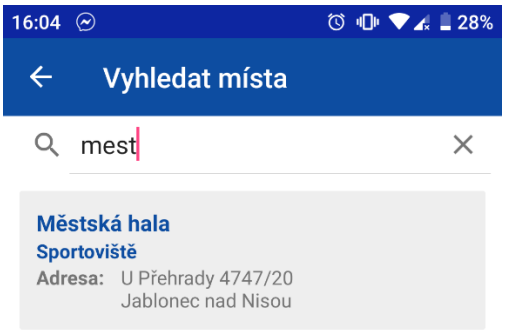
Obrázek 33 – Filtry zpráv dle předmětu

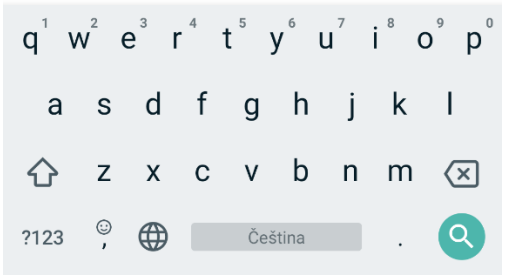
5.5.4 Navigace a místa

Tyto části aplikaci dovolují uživateli zobrazení a navigaci na daná místa konání.

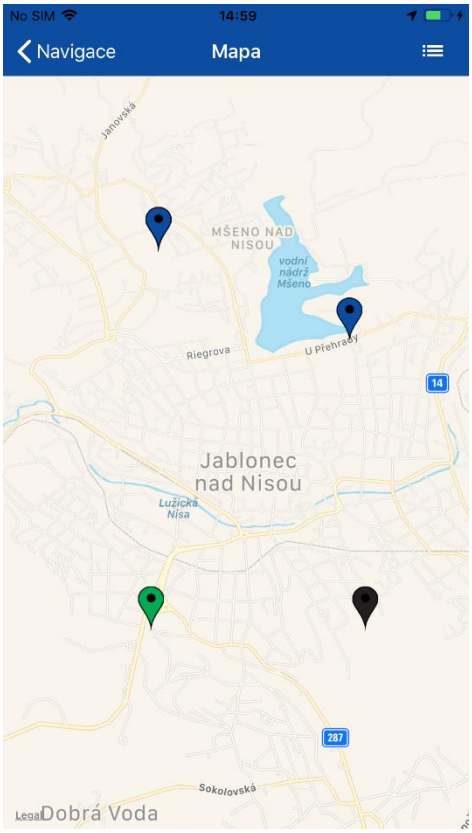


Obrázek 35 – Root page navigačních funkcí

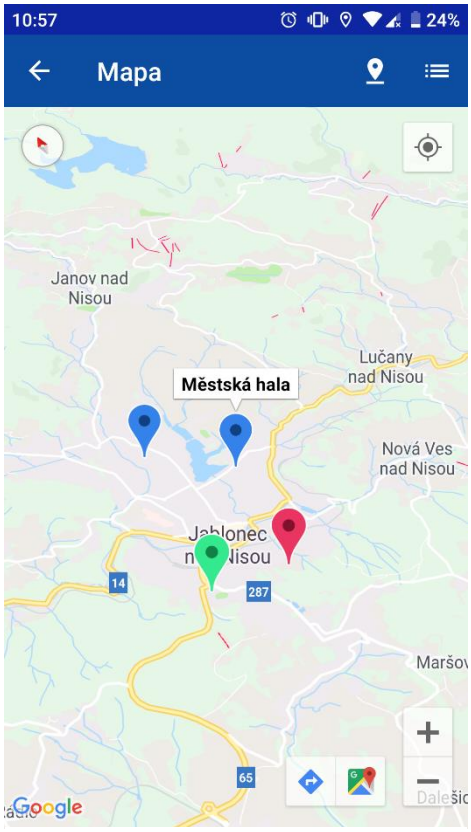




Obrázek 34 – Hledání míst dle klíčových slov



Obrázek 39 – iOS nativní mapa míst



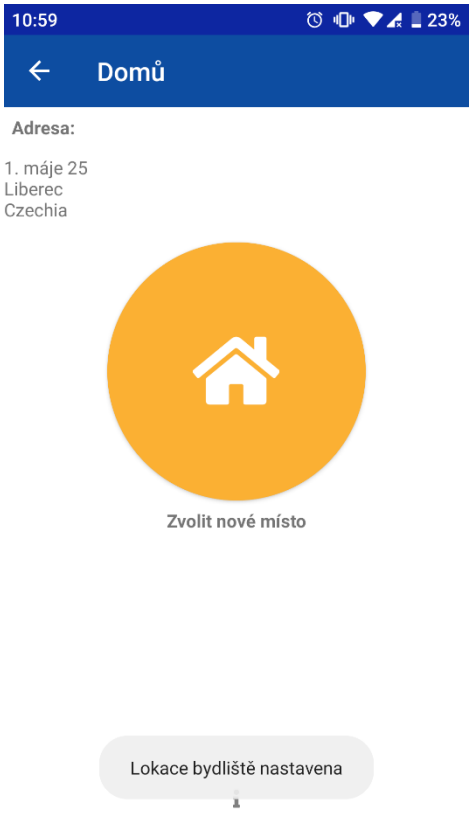
Obrázek 38 – Android nativní mapa míst

Obrázek 36 – Detail místa

Obrázek 37 - Detail soutěže se zápasy konaných na místě

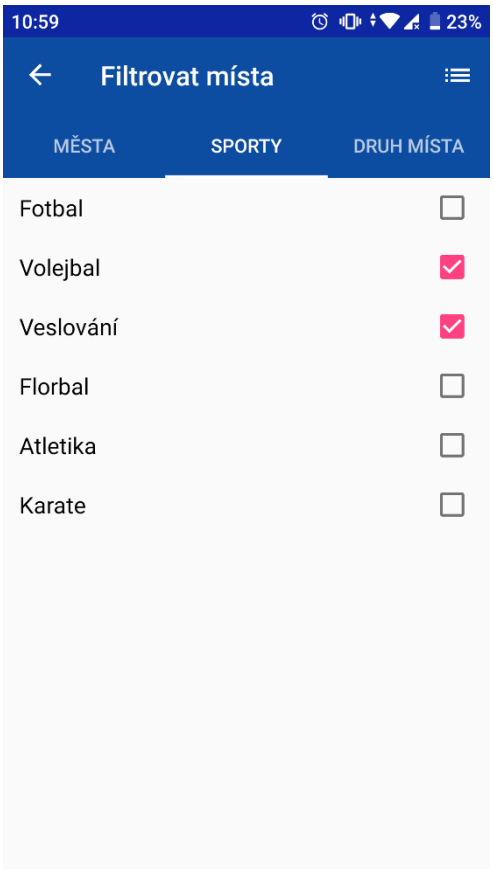
Na místa ve všech částech aplikace je možná navigace pomocí platformních aplikací.

Obrázek 40 – Stránka pro nastavení bydliště

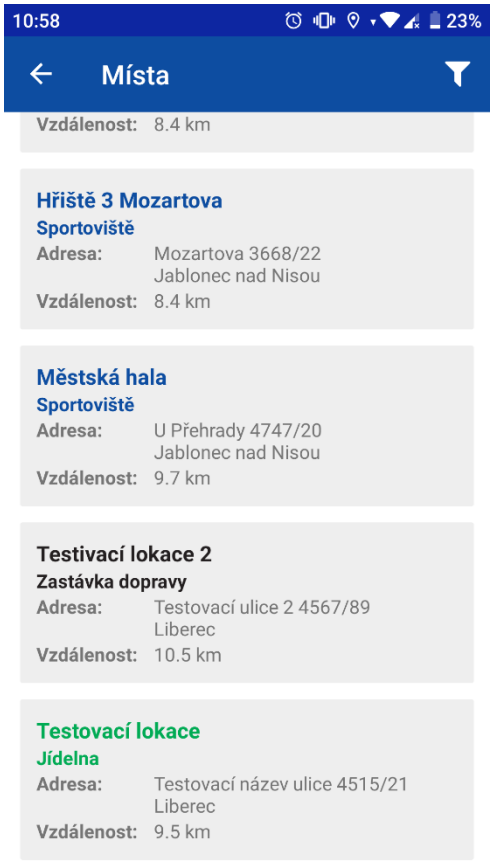


Obrázek 41 – Stránka s nastaveným bydlištěm

Tyto stránky slouží pro nastavení bydliště a pozdější navigaci na něj pomocí platformních aplikací (Google maps pro Android).

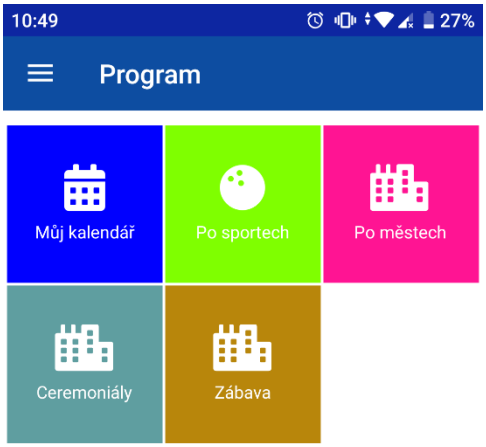


Obrázek 43 – Filtrování míst dle parametrů (města, konané sporty, druh místa)



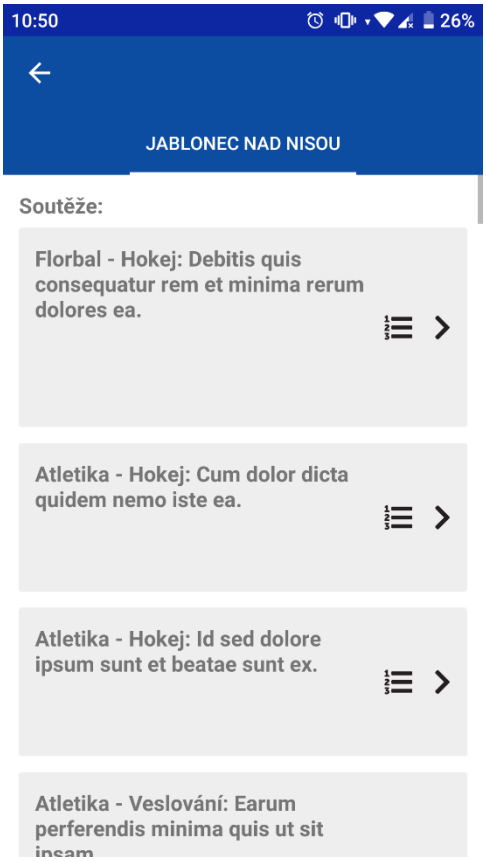
Obrázek 42 – Stránka se seznamem míst

5.5.5 Program

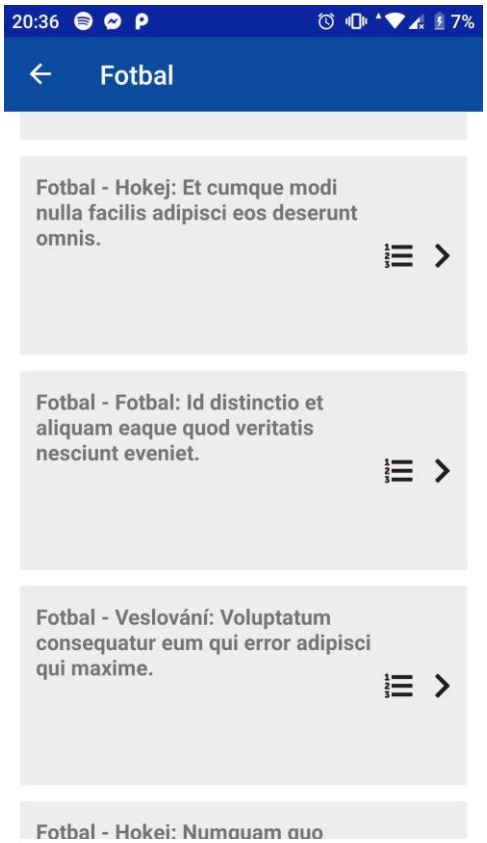


Obrázek 46 – Root stránka programu

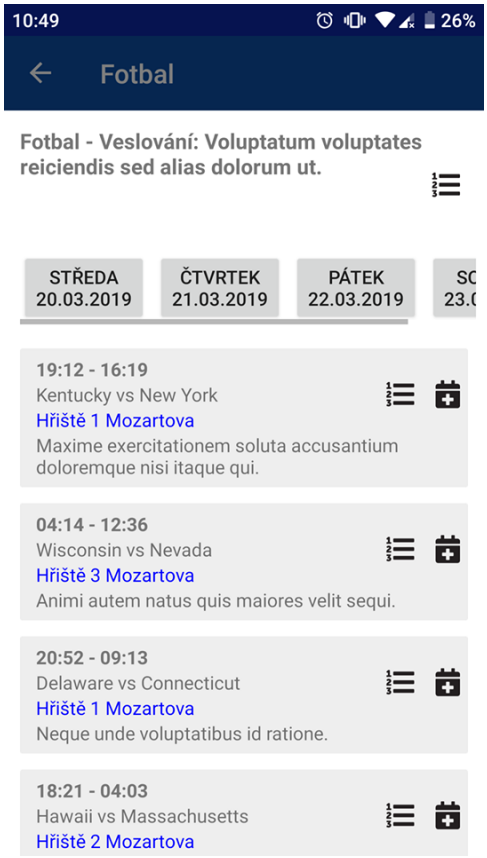
Obrázek 45 – Stránka se sporty (placeholder ikonky)



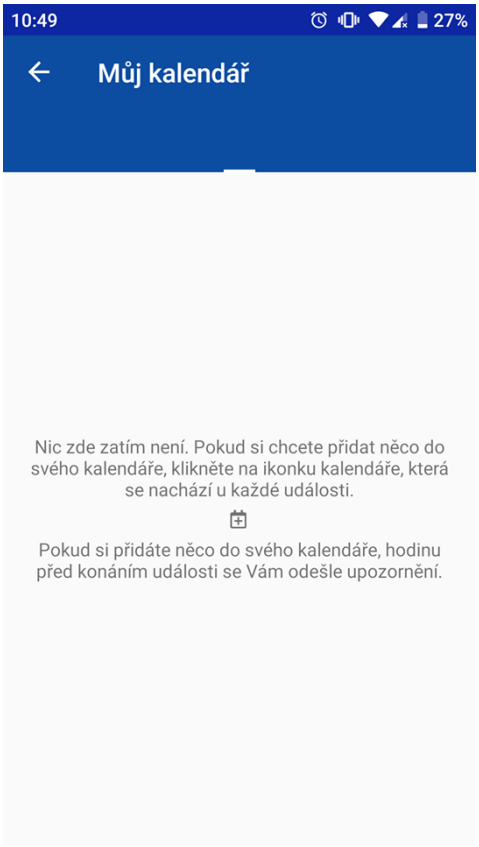
Obrázek 47 – Tabbed page stránka pro filtrování dle měst, ve kterých se soutěže pořádají



Obrázek 44 – Detail sportu, seznam soutěží v daném sportu



Obrázek 50 – Detail soutěže, seznam akcí (zápasů), filtrování podle dne



Obrázek 49 - Prázdný kalendář

Obrázek 48 - Zaplněný kalendář

5.5.6 Jídelníček

Obrázek 51 - Jídelníček

Závěr

Webová i mobilní aplikace jsou stále ve stádiu aktivního vývoje, nereprezentují tedy finální produkt.

V mobilní aplikaci zbývá implementovat pár funkcí jako modul doprava a modul kontakty. Stávající design uživatelského rozhraní není konečný a aktivně se pracuje na novém.

Webová služba je aktuálně napojena na prozatimní zdroj dat, který generuje několik desítek soutěží a několik tisíc zápasů, data sportů a míst jsou staticky vytvořená seedováním databáze, s pomocí těchto dat byly provedeny zátěžové testy webové služby a mobilní aplikace, které neodhalily žádné nedostatky ohledně výkonu. V budoucnu se napojí na oficiální API dodané firmou eSport.

Až bude hotový nový design a aplikace bude napojena na oficiální zdroj dat, mám v plánu uspořádát veřejný beta test. V tuto chvíli byla mobilní aplikace vyzkoušena na několika Android a iOS zařízení, veřejný beta test by však mohl odhalit skryté chyby (často vykreslování uživatelského rozhraní), způsobené různými verzemi OS a upravenými launchery.

Seznam obrázků

Použitá literatura

1. **Microsoft.Inc.** ASP.NET Documentation. *docs.microsoft.com.* [Online] https://docs.microsoft.com/en-gb/aspnet/.

2. —. Entity Framework Documentation. *docs.microsoft.com.* [Online] https://docs.microsoft.com/en-us/ef/.

3. **Chugh, Anupam.** Android MVVM Design Pattern. *Journaldev.* [Online] [Citace: 2019. 03 14.] https://cdn.journaldev.com/wp-content/uploads/2018/04/android-mvvm-pattern.png.

4. **Dunn, Craig, Britch, David a Petzold, Charles.** Hierarchical Navigation. *docs.microsoft.com.* [Online] [Citace: 12. 03 2019.] https://docs.microsoft.com/en-gb/xamarin/xamarin-forms/app-fundamentals/navigation/hierarchical.

5. **Microsoft.Inc.** Xamarin.Forms. *docs.microsoft.com.* [Online] https://docs.microsoft.com/en-gb/xamarin/xamarin-forms/.

6. **Wikipedia.** Model-view-controller. *wikipedia.org.* [Online] [Citace: 10. 03 2019.] https://en.wikipedia.org/wiki/Model-view-controller.

7. **Hangfire.** Documentation. *Hangfire.io.* [Online] [Citace: 12. 03 2019.] http://docs.hangfire.io/en/latest/.

8. **Serilog.** Home. *github.com/serilog.* [Online] https://github.com/serilog/serilog/wiki.

A. Seznam přiložených souborů

1. Složka ODMWeb – obsahuje solution webové služby
2. Složka ODM – obsahuje solution mobilní aplikace

Je důležité zachovat strukturu těchto adresářů kvůli sdílenému projektu s datovými třídami.

3. cz.skeleton.odm.apk – sestavený Android balíček mobilní aplikace