Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Aplikace pro mobilní telefony, tablety a chytré hodinky na platformě Apple

Bc. Jan Mísař

Vedoucí práce: Ing. Josef Gattermayer

4. května 2015

Poděkování

V první řadě bych rád poděkoval Ing. Josefu Gattermayerovi za vedení této práce, dále také dalším členům firmy Ackee za cenné rady, informace a pomoc v průběhu práce a v neposlední řadě také rodičům za podporu v průběhu celého mého studia a všem dalším, kteří jakkoli přispěli ke vzniku této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. května 2015 .....

Odkaz na tuto práci

Mísař, Jan. *Aplikace pro mobilní telefony, tablety a chytré hodinky na platformě Apple.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Obsahem této práce je návrh a implementace univerzální mobilní aplikace zaměřené na společensko kulturní akce pro mobilní zařízení na platformě Apple iOS. Součástí je i návrh serverového aplikačního rozhraní.

Klíčová slova Apple, iOS, mobilní aplikace, kulturní akce, API

Abstract

This thesis contains design and implementation of universal mobile appliaction, which is focused on social and cultural events, for Apple iOS devices. Thesis also describes design of server application interface.

Keywords Apple, iOS, mobile applications, cultural event, API

Obsah

Seznam obrázků

Úvod

Díky velkému rozšíření chytrých telefonů a bohaté nabídce aplikací si lidé navykli používat telefony k mnoha různým účelům a začali tím postupně nahrazovat zaběhnuté technologie a postupy. Jedním z těchto případů jsou i kulturní akce (hudební a filmové festivaly) nebo konference či výstavy, kde papírové programy a letáky začínají být postupně nahrazovány právě mobilními aplikacemi. To s sebou nese mnoho výhod, jako je například ušetření nákladů na tisk, možnost udržovat informace aktuální nebo lepší uživatelský zážitek pro návštěvníky, kteří tak mají pohodlný a rychlý přístup k programu, aktualitám, odkazům apod. Uživatelům se nabízí i další výhoda v podobě určité personalizace jako například plánování svého osobního programu a upozornění na vybrané události. Obecně jsou tyto možnosti díky dnešním technologiím takřka nekonečné.

Dnes již mnoho festivalů a dalších akcí svou aplikaci má, tyto aplikace jsou ale zpravidla vyvíjeny úplně od začátku pouze pro konkrétní akci a navíc různými dodavateli. Kompletní vývoj pak zbytečně zatěžuje často nízké rozpočty akcí, přestože všechny tyto aplikace jsou velmi podobné, ne-li stejné. Tohoto faktu lze využít k vytvoření univerzální aplikace, která bude pouze konfigurována pro konkrétní akce (barevné schéma, logo apod.). Tím se jednak zásadně sníží zmíněné pořizovací náklady pro organizátory akcí a jednak se takový produkt může stále vyvíjet, vylepšovat a pružně reagovat na aktuální trendy.

Kapitola 1

Cíle a požadavky

Jak bylo nastíněno v úvodu, cílem této práce je univerzální aplikace pro hudební a filmové festivaly a konference. Aplikace by měla být kompletní funkční kostrou, která bude snadno přizpůsobitelná potřebám konkrétního zákazníka.

Další částí je pak definice API (aplikačního rozhraní), skrz které bude aplikace získávat data ze serveru. Toto API bude navrženo tak, aby na něm mohly být postaveny i aplikace pro jiné mobilní platformy (Android, Windows Phone a případně další)

1.1 Omezení a potenciální problémy

Cílovou skupinou aplikace jsou návštěvníci velkých společenských akcí. Hlavním úskalím těchto akcí (v kontextu této práce) bývá koncentrace velmi vysokého počtu lidí na malém prostoru. Festivaly se často konají na odlehlých místech, která jsou jinak velmi řídce osídlená a místní infrastruktura nebývá dimenzována na tak silný jednorázový nápor. To se týká především mobilní sítě, kdy musí jeden nebo dva vysílače najednou obsluhovat tisíce až desetitisíce klientů. To vede k velmi výraznému zpomalení a častým výpadkům až úplné nedostupnosti mobilní datové sítě. Zařízení tak pracují prakticky úplně offline.

Dalším omezením bývá nedostupnost elektřiny v kempech, návštěvníci tak často nemají možnost si v průběhu několikadenní akce zařízení dobít, což je při běžné výdrži těchto zařízení problém, který musí být při návrhu aplikace patřičně zohledněn.

Aplikace tedy musí být navržena tak, aby se v průběhu akce pokud možno úplně obešla bez internetového připojení a co nejvíce šetřila energii zařízení.

Ani jedna z těchto skutečností není zásadním problémem u konferencí a kongresů, které se konají většinou ve městech a obecně na místech, kde bývá bez problému dostupná jak dostatečně výkonná datová (Wi-Fi nebo mobilní LTE či 3G), tak i elektrická síť. Ač zmíněná opatření nejsou pro tyto akce

nutná, tak samozřejmě nebudou na škodu a přispějí k lepšímu uživatelskému zážitku.

1.2 Požadavky na aplikaci

Všechny požadavky - jak funkční, tak nefunkční – souvisí především se samotným zadáním práce, ale mnoho jich také vychází ze standardů, zvyklostí nebo omezení platformy iOS, což se týká hlavně uživatelského rozhraní, které by mělo odpovídat příslušným Human Interface Guidelines definovaným firmou Apple [21]. Tyto požadavky zde nebudou blíže popisovány. Co ovšem musí být při definici požadavků bráno v potaz, jsou omezení diskutovaná v předchozí kapitole.

1.2.1 Funkční požadavky

Funkční požadavky jsou obecně požadavky na to, co všechno má produkt obsahovat, co má umět a jak se má chovat [60].

Pro tuto práci jsou funkční požadavky definovány následovně:

- Aplikace bude obsahovat:
  - program ve formě časové osy
  - seznam vystupujících
  - seznam událostí včetně míst jejich konání
  - seznam novinek včetně obrázků
  - seznam informací roztříděných do sekcí
  - mapu ve formě schématického obrázku
- Uživatel bude mít možnost si jednotlivé události zařazovat do svého osobního programu a spravovat tento program.
- Uživatel bude upozorňován na začátky svých událostí pomocí systémových notifikací.
- Místa konání bude možné slučovat do skupin.
- Aplikace bude přijímat push notifikace ze serveru.

1.2.2 Nefunkční požadavky

Nefunkční požadavky obecně nesouvisí přímo s funkcemi nebo obsahem aplikace, ale definují vlastnosti či omezení produktu jako celku. Může se jednat například o požadavky na výkonnost, přenositelnost nebo proces distribuce [60].

Pro tuto práci jsou nefunkční požadavky definovány následovně:

- Aplikace bude určena pro zařízení s operačním systémem iOS, konkrétně iPhone, iPad a Apple Watch.
- Aplikace bude univerzální pro více druhů akcí, konkrétně pro hudební nebo filmové festivaly a konference či kongresy.
- Aplikace bude navržena tak, aby byla co nejsnadněji přizpůsobitelná pro konkrétního klienta a náklady na vydání finální upravené verze byly minimální.
- Aplikace bude podporovat více jazykových mutací.
- Veškerý obsah bude stahován a pravidelně aktualizován ze vzdáleného serveru.
- Aplikaci bude možné plnohodnotně využívat i offline.
- Aplikace musí být velmi „citlivá“ z hlediska životnosti baterie a datových přenosů.

1.3 Požadavky na API

Vzhledem k tomu, že implementace serverové části není součástí této práce, tak zde není třeba řešit problémy jako dostupnost, škálovatelnost, uživatelské rozhraní pro obsluhu serverové části a další, ale můžeme se zaměřit pouze na definici API.

V případě definice požadavků na API je potřeba se kromě samozřejmého dodržování osvědčených postupů a standardů zaměřit hlavně na specifické podmínky, při kterých se akce konají (viz kapitola 1.1).

Požadavky mohou být shrnuty do následujících bodů:

- API bude maximálně datově úsporné.
- API bude nezávislé na klientské platformě.
- API bude respektovat určený standard, který bude snadno implementovatelný na co nejširším spektru klientských platforem i na serverové straně.

Kapitola 2

Analýza

Tato kapitola se zabývá především detailní analýzou cílové platformy jak z hlediska hardwaru, tak z hlediska softwaru. Dále jsou zde popsány některé důležité technologie, které byly na základě analýzy zvoleny a použity při následné implementaci.

2.1 Analýza cílové platformy

Tato kapitola popisuje platformu iOS z pohledu softwaru i z pohledu hardwaru, na kterém běží. Každé zařízení, pro které má být aplikace přizpůsobena má svá specifika a omezení. Z popsaných faktů následně vychází následný výběr technologií a knihoven, wireframes i výsledná implementace.

2.1.1 Operační systém iOS

iOS je mobilní operační systém, který vyvíjí firma Apple exkluzivně pro svá zařízení, konkrétně iPhone, iPod Touch a iPad. Je to systém vycházející z operačního systému pro stolní počítače OS X [5], sdílí s ním jednak UNIXové jádro [54] a jednak některé hlavní knihovny a frameworky.

2.1.1.1 Historie

Systém iOS se poprvé objevil v roce 2007 v první generaci iPhone [4]. Ve své první verzi neměl tehdy ještě bezejmenný systém veřejně dostupné SDK a nebylo tedy možné pro něj programovat aplikace třetích stran. To bylo zveřejněno v roce 2008 společně se změnou názvu na iPhone OS [6]. Následně byl iPhone OS kromě iPhonu nasazen i na nové zařízení iPad a v důsledku toho přejmenován na dnešní iOS [7].

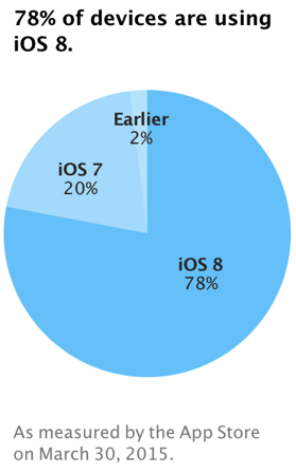
2.1.1.2 Verze a jejich rozšiřování

iOS prošel od své první verze dlouhým vývojem a po osmi letech je dnes dostupná již jeho osmá verze [11]. Ve svých začátcích byl systém velmi uzavřený a omezený, ale s postupem času a s přibývajícími hardwarovými možnostmi zařízení Apple začal vývojářům zpřístupňovat více a více možností.

Co se týká vydávání nových verzí, je iOS známý svým rychlým rozšiřováním mezi uživatele. Tomu napomáhá především oproti konkurenci dlouhá podpora starších zařízení. Například na iPhonu 4 byla plně podporována i sedmá verze iOS, přestože v době uvedení telefonu na trh byl iOS teprve ve verzi 4 [8]. Takto dlouhá kompatibilita se staršími zařízeními ve spojení s agresivní politikou distribuce nových verzí vede k jejich velmi rychlému rozšiřování.

Na obrázku 2.1 je vidět aktuální rozšíření nejnovější osmé verze iOS. Po zhruba půl roce od vydání je nejnovější verze na bezmála 80% zařízení [14].

Tato skutečnost přispívá k velmi progresivnímu vývoji celé platformy. iOS vývojáři se díky tomu nemusí zabývat zpětnou kompatibilitou se staršími verzemi systému a mohou již po krátké době využívat všechny aktuální možnosti a postupy.



Obrázek 2.1: Aktuální zastoupení verzí iOS [14]

2.1.1.3 Vývojářské nástroje

Jediné vývojové prostředí, ve kterém je možno vyvíjet aplikace pro iOS, je Xcode, které Apple poskytuje výhradně pro svůj operační systém OS X. Spolu s prostředím Xcode je dostupná celá řada podpůrných vývojářských nástrojů včetně simulátorů všech současných iOS zařízení [26]. Pro vývoj iOS aplikací byl původně určen pouze programovací jazyk Objective C. Tento jazyk, který je nadstavbou jazyka C, byl využíván již v roce 1989 v operačním systému

NEXTSTEP [50], ze kterého vychází jak OS X, tak následně i iOS. Vzhledem ke značnému stáří Objective C a přicházejícím novým trendům v oblasti programovacích jazyků, Apple v roce 2014 představil pro své produkty nový programovací jazyk Swift [10]. Swift je možné používat v kombinaci s Objective C i v rámci jednoho projektu, přechod na tuto novou technologii tedy může být postupný a není nutné staré kódy v Objective C přepisovat. V současné době lze pro vývoj využít oba zmíněné jazyky a případně je i kombinovat.

2.1.2 Hardware

Všechny iOS zařízení pohání procesory založené na architektuře ARM, které si kromě procesorů v prvních třech generacích iPhonu, které jsou osazeny procesory firmy Samsung, navrhuje Apple sám [33]. Díky tomu má Apple dokonalou kontrolu nad svým hardwarem a na základě toho může velmi dobře optimalizovat software. Právě z těchto důvodů jsou i starší podporovaná zařízení dostatečně výkonná a programátoři se tak mohou soustředit na samotný vývoj aplikací a nemusí se příliš zabývat rychlostí procesoru, velikostí operační paměti a dalšími výkonnostními parametry.

2.1.2.1 iPhone

iPhone je smartphone od firmy Apple, jehož první generace byla představena v roce 2007 [4], a dnes je na trhu jeho již osmá generace. Aktuálně jsou firmou Apple podporována zařízení 5. generace a novější. Jedná se konkrétně o iPhone 4S, 5, 5C, 5S, 6 a 6 Plus [11]. Pro všechna tato zařízení je dostupná nejnovější verze iOS a to verze 8.3.

Tyto modely iPhonu se od sebe kromě výkonnostních parametrů liší hlavně osazenými displeji. Všechny zmíněné modely již sice mají displeje s vysokým rozlišením (retina displeje [38]), ale ve čtyřech různých velikostech. Pro vývojáře je důležité především rozlišení a to z důvodu přizpůsobení uživatelského rozhraní pro všechny dostupné varianty. Všechna rozlišení všech aktuálně podporovaných zařízení jsou uvedena v následující tabulce [45].

Model	Velikost	Rozlišení
iPhone 4S	3,5”	640 × 960 px
iPhone 5, 5C, 5S	4”	640 × 1136 px
iPhone 6	4,7”	750 × 1334 px
iPhone 6 Plus	5,5”	1080 × 1920 px

Z hlediska vývoje aplikací můžeme k iPhonu zařadit i zařízení iPod Touch, které ač je prezentováno primárně jako hudební přehrávač, tak se jedná prakticky o iPhone bez GSM modulu. iPody Touch jsou osazeny stejným hardwarem i operačním systémem jako iPhony a kromě zmíněného telefonování nabízí i naprosto identickou funkčnost. V současné době je firmou Apple plně

podporována už pouze poslední pátá generace tohoto zařízení, která hardwarovou výbavou odpovídá iPhonu 4S [22].

2.1.2.2 iPad

iPad je tablet od firmy Apple, který byl poprvé představen v roce 2010, a od té doby Apple vydal několik nových generací, včetně nejnovějšího modelu iPad Air. V roce 2012 přidal do svého tabletového portfolia zmenšenou verzi nazvanou iPad Mini, kterou taktéž i nadále vyvíjí a aktualizuje. Z řady iPad jsou kromě prvního modelu stále podporovány všechny generace a u řady iPad Mini dokonce všechny dosud vydané modely [49].

Stejně jako u iPhonu je z hlediska vývoje nejdůležitější rozlišení displeje. Nejstarší podporovaný iPad ještě nevyužívá retina displej a rozlišení jeho displeje činí 1024 × 768 obr. bodů. Stejné rozlišení používá i první generace iPadu Mini. Všechny ostatní modely iPadu i iPadu Mini jsou již osazeny retina displeji s dvojnásobným rozlišením 2048 × 1536 obr. bodů. Situace je tedy jednodušší než v případě iPhonu. Oproti čtyřem různým variantám rozlišení na iPhonech se vývojáři na iPadu potýkají pouze se dvěma variantami [49].

Jestliže o iPodu Touch můžeme tvrdit, že je to iPhone bez GSM modulu, tak o iPadu by se dalo říct, že se jedná o zvětšený iPhone bez možnosti telefonování. Systém iOS je na obou zařízeních víceméně stejný, verze pro iPad je pouze přizpůsobena pro větší displeje a i hardwarové vybavení je velmi podobné. Vývoj aplikací je tak až na nutnost přizpůsobení se většímu displeji identický jako vývoj pro iPhone.

2.1.3 Apple Watch

Apple Watch jsou chytré hodinky, které byly představeny v září roku 2014. V době vzniku této práce, ještě hodinky nebyly uvedeny na trh, a proto všechny zde uvedené informace vycházejí pouze z marketingových materiálů firmy Apple, případně z dokumentace a poznatků zjištěných při vývoji. Hodinky se vyrábí ve dvou velikostech lišících se uhlopříčkou displeje. Menší verze má displej o uhlopříčce 38 mm a větší verze 42 mm. Tyto dvě velikosti jsou pak nabízeny ve třech edicích, které se ovšem liší pouze vzhledem nebo použitými materiály. Z hlediska hardwaru jsou tak až na zmíněné dvě velikosti displeje všechny modely naprosto identické.

Apple Watch jsou hodinky, které kromě zobrazování času nabízí i propojení s telefonem pomocí technologie bluetooth a otevírají tak nové možnosti tohoto segmentu zařízení. Hodinky umí například zobrazovat notifikace z telefonu, měřit pomocí senzorů tepovou frekvenci, přehrávat hudbu do bluetooth sluchátek a další. Nejzajímavější je ale z hlediska této práce možnost vývoje aplikací.

Přestože hodinky ještě nebyly v době vzniku této práce fyzicky dostupné, Apple při jejich představení vydal veřejné SDK a v rámci Xcode i vývojářské

nástroje včetně simulátoru hodinek. I bez fyzického zařízení je tedy možné plnohodnotně vyvíjet aplikace.

Zařízení Apple Watch se od iPhonu nebo iPadu zásadně liší, nejedná se totiž o samostatné zařízení, na kterém by běžel plnohodnotný iOS, ale jde pouze o doplněk, rozšíření k iPhonu. Z toho také vychází architektura aplikací a způsob, jakým hodinky fungují [16].

2.1.3.1 Architektura spojení

Hodinky jsou oproti telefonům nebo tabletům výrazně menší, tudíž mají i výrazně menší baterii. To výrobcům neumožňuje osadit hodinky příliš výkonným hardwarem. Ač by to teoreticky bylo možné, překážkou je s výkonem se zvyšující energetická náročnost. Tyto problémy s malou baterií a tedy nízkým výkonem Apple vyřešil delegací veškerých výpočetních operací do připojeného telefonu.

Aplikace pro hodinky je pouze rozšířením běžné iOS aplikace, ve skutečnosti se jedná spíše o iOS aplikaci, která se umí zobrazit na hodinkách, než vyloženě o „aplikaci pro hodinky“. Takováto iOS aplikace s rozšířením pro hodinky má tři hlavní části. První nepostradatelnou částí je samotná iOS aplikace neboli mateřská aplikace. Druhou je WatchKit Extension, což je kód, který obsluhuje zmíněné delegované výpočetní operace z hodinek. Tato část je uložena v telefonu společně s mateřskou aplikací. Poslední částí je WatchKit App, která je jediná fyzicky uložena v hodinkách. Ta neobsahuje žádný spustitelný kód, ale pouze definici uživatelského rozhraní a návaznosti uživatelských akcí na kód ve WatchKit Extension. Veškerou komunikaci v rámci těchto třech částí zajišťuje knihovna WatchKit. Tato architektura je znázorněna na obrázku 2.2.

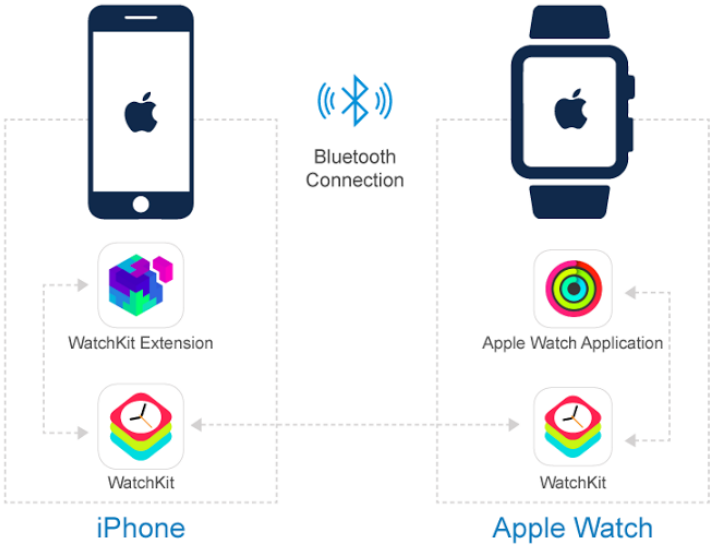
Z popsané architektury vychází i způsob distribuce těchto aplikací. Aplikace pro hodinky nemůže bez mateřské aplikace existovat a je tedy distribuována společně s ní jako její součást [17].

2.1.3.2 Možnosti a omezení WatchKitu

Kompletní logika a veškerý kód je vykonáván v iPhonu, proto nejsou na tento kód kladena žádná větší omezení. Je možno využívat obvyklé knihovny, databázovou vrstvu i všechny další postupy známé z programování na iOS.

Velký rozdíl je ovšem v návrhu a práci s uživatelským rozhraním. Zatímco na iOS umisťujeme objekty ručně přímo na konkrétní souřadnice a rozložení objektů je pouze na programátorovi, tak na hodinkách je rozložení objektů z velké části v režii systému. Objekty jsou skládány ve výchozím režimu pod sebe a programátor má v tomto směru velmi omezené možnosti.

Z popsané architektury komunikace také vychází, že není možné vytvářet objekty uživatelského rozhraní dynamicky v kódu, objekty je možné pouze předem definovat a poté je v kódu plnit obsahem (např. textem nebo obráz-



Obrázek 2.2: Komunikace Apple Watch s iPhonem [17]

kem), případně je skrývat nebo odkrývat. Naproti tomu v iOS je možné z kódu za běhu vytvořit nový objekt a vložit ho do existující hierarchie objektů na obrazovce.

Popsané skutečnosti se mohou jevit jako velmi omezující, ale v případě úplně nového produktu, kterým Apple Watch jsou, jsou tato omezení pochopitelná a z pohledu výrobce dobře nastavená. Programátoři jsou těmito omezeními poměrně striktně vedeni k jednotně vypadajícím aplikacím, které zapadají do vzhledu a vzorců ovládání celého systému.

Podobný trend bylo možné pozorovat i u starších verzí iOS. Zpočátku velmi omezený systém se postupně vývojářům víc a víc otevíral a vývojáři mají tak dnes mnohem více volnosti než v raných verzích systému. Tento vývoj je možné očekávat i u hodinek. [18]

2.2 Použité technologie

Na základě vytyčených požadavků a předchozí analýzy byly vybrány některé knihovny a technologie, které jsou následně využity při implementaci. Jedná se jak o využití vlastností a možností systému iOS, tak o knihovny či frameworky třetích stran, které usnadňují a urychlují vývojový proces. Všechny vybrané technologie jsou popsány v následujících kapitolách.

2.2.1 Objective C

Jak je popsáno v kapitole 2.1.1.3, firma Apple nabízí vývojářům dva programovací jazyky, ve kterých je možno nativní aplikace pro iOS vyvíjet. Jsou to původní Objective C a nový jazyk Swift. Přestože aktuálně (duben 2015) je Swift již na velmi dobré úrovni a existuje pro něj velké množství knihoven i podpora vývojářské komunity, tak pro vývoj aplikace byl zvolen jazyk Objective C. V době začátku práce na aplikaci byl Swift velmi čerstvou novinkou a vývoj v tomto jazyce by tedy kromě využití modernějšího jazyka nebyl žádným přínosem, ale naopak přítěží. V průběhu vývoje aplikace se situace okolo Swiftu velmi zlepšila, ale přepis do něj by byl zbytečnou prací a případná kombinace obou jazyků by byla nepřehledná, proto je pro vývoj použit právě jazyk Objective C.

2.2.2 Cocoapods

CocoaPods je manažer závislostí pro Objective C a Swift. Umožňuje snadno přidávat do projektu knihovny třetích stran, spravovat jejich verze a vzájemné závislosti a aktualizovat je. Všechny knihovny potřebné pro projekt se definují v souboru Podfile a podle něj jsou pak instalovány. To přináší výhodu při používání verzovacích systémů. Stačí verzovat pouze Podfile s definicí knihoven a samotný kód knihoven nemusí být v repozitáři vůbec přítomen. [34]

Použití CocoaPods je velmi pohodlné, přináší mnoho výhod a všechny knihovny, které budou v projektu využity tento nástroj podporují, proto je instalace všech knihoven řešena tímto nástrojem.

2.2.3 Storyboard

Uživatelské rozhraní na iOS je možné vytvářet dvěma způsoby. Prvním je definice v kódu a druhým je použití storyboardu. Storyboard je soubor ve formátu XML [57] obsahující definici všech UI prvků, jejich pozice, rozměry, barvy a další vlastnosti. Tyto soubory jsou v Xcode upravovatelné pomocí grafického rozhraní, které funguje na principu WYSIWYG [44]. Programátor si může pohodlně navrhnout celé uživatelské rozhraní bez jediného řádku kódu. Takto vytvořené uživatelské rozhraní lze pak propojit s kódem a ke všem prvkům z kódu přistupovat a upravovat je. [23]

Tento postup přináší na první pohled významnou úsporu času a pohodlnější práci, ale v praxi je použitelný pouze pro jednodušší aplikace. Při definici komplexnějšího uživatelského rozhraní, kdy je v editoru velké množství prvků, se celý storyboard stává značně nepřehledným. Ve storyboardu je možno definovat i závislosti autolayoutu popsané v následující kapitole a znázornění těchto závislostí pak činí storyboard ještě méně přehledným.

Dalším problémem storyboardu je nepohodlná práce v týmu. Celé uživatelské rozhraní je uloženo v jediném souboru a paralelní práce více vývojářů najednou je tak velmi problematická, a to i při použití verzovacích systémů

jako například Git [51] nebo SVN [2], kde dochází ke konfliktům, které se špatně řeší.

Z výše zmíněných důvodů není v aplikaci využit storyboard, ale veškeré UI prvky jsou definovány přímo v kódu. Storyboard je využit pouze pro definici uživatelského rozhraní Apple Watch, protože, jak bylo zmíněno v kapitole 2.1.3.2, UI prvky na Apple Watch nelze dynamicky vytvářet v kódu a jiná možnost než použití storyboardu tedy není.

2.2.4 Autolayout

Až do příchodu modelu iPhone 5, měly displeje iPhonů stejné rozlišení 320 × 480 obrazových bodů. iPhone 4 a 4S sice již disponoval retina displejem a měl tak dvojnásobné fyzické rozlišení, ale z programátorského hlediska se stále pracovalo s původním rozlišením, pouze na retina displejích bylo výsledné zobrazení jemnější. Díky tomu nebyl potřeba žádný pokročilejší systém pro rozmisťování UI prvků. Ty byly zpravidla umisťovány na absolutní pozice a byly jim definovány absolutní velikosti v obrazových bodech, protože programátor si mohl být jistý, že výsledek bude na všech zařízeních vypadat totožně.

S příchodem iPhonu 5, který měl poprvé jinou velikost displeje, firma Apple představila iOS 6 a společně s ním i technologii autolayout. Jedná se o princip rozmisťování UI prvků na základě definovaných závislostí mezi nimi. Namísto přesné pozice jsou definovány vzdálenosti mezi jednotlivými prvky a jejich absolutní nebo procentuální velikost a ty se pak podle těchto nadefinovaných vztahů automaticky rozmístí na obrazovku. [19]

2.2.4.1 Knihovna Masonry

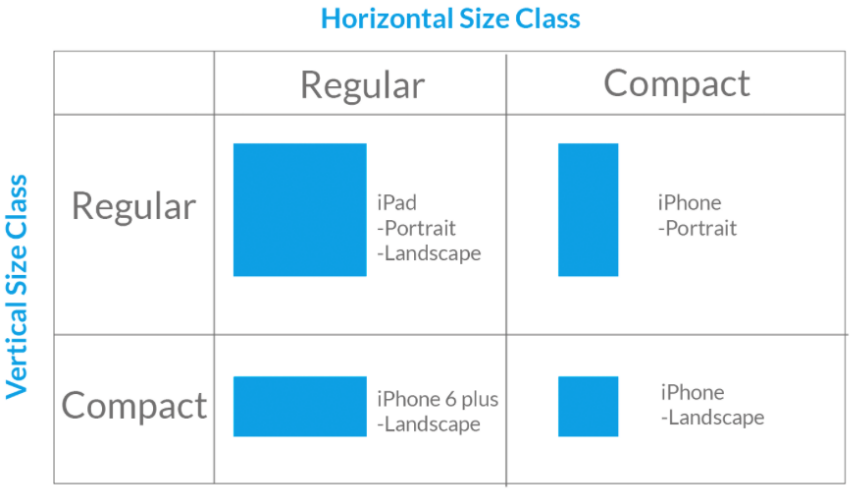
Definice autolayoutu pro rozsáhlejší obrazovky s mnoha UI prvky často znamená velmi zdlouhavý a nepřehledný kód, proto je pro definici autolayoutu využita knihovna Masonry, která nabízí přehlednější a kratší zápis definice jednotlivých vztahů. Tato knihovna je pouze nadstavbou nad standardním autolayoutem, proto je možné v kódu kombinovat oba přístupy. Díky tomu není Masonry nikterak omezující, ale pouze urychluje vývoj a umožňuje tvořit přehlednější kód. [30]

2.2.5 Adaptive UI

Adaptive UI je technologie, kterou Apple uvedl společně s představením iOS 8. Princip této technologie je postaven na třídě `UITraitCollection`, která obrazovkám poskytuje informaci o aktuálních rozměrech displeje, na základě čehož mohou obrazovky reagovat a měnit rozmístění a chování UI prvků. Tato informace není poskytována jako hodnota počtu obrazových bodů, ale jako velikostní třída. `UITraitCollection` tedy obsahuje informace o dvou velikostních třídách, pro horizontální osu a pro vertikální osu. Tyto třídy jsou pouze

dvě (regular a compact) a jsou jednoznačně definovány pro každé zařízení. Všechna iOS zařízení samozřejmě disponují automatickým otáčením displeje na výšku a na šířku (portrait a landscape), proto jsou velikostní třídy definovány i pro všechny tyto dostupné polohy. Kompletní definice velikostních tříd pro všechna zařízení je znázorněna na obrázku 2.3.

Pomocí adaptive UI v kombinaci s autolayoutem tak lze na iOS vytvořit uživatelské rozhraní, které je naprosto nezávislé na skutečném rozlišení displeje. Díky tomu není třeba vytvářet uživatelské rozhraní pro iPhone a iPad zvlášť. Pokud jsou při implementaci správně definovány a nastaveny závislosti mezi UI prvky a správně nastaveno chování pro konkrétní velikostní třídy, tak je aplikace bez větších zásahů dobře použitelná na všech zařízeních. [39]



Obrázek 2.3: Vizualizace size classes pro všechna zařízení [39]

2.2.6 UISplitViewController

`UISplitViewController` byl až do příchodu iOS 8 dostupný pouze pro iPad. S příchodem iOS 8 firma Apple umožnila `UISplitViewController` používat i na iPhonech zejména kvůli modelu iPhone 6 Plus. Ten již disponuje dostatečně velkým displej, aby mělo význam na něm `UISplitViewController` používat.

`UISplitViewController` v iOS 8 již podporuje velikostní třídy popsané v předchozí kapitole a dokáže se tedy sám dynamicky přizpůsobovat velikosti displeje. Pokud má displej horizontální velikostní třídu regular, tak se chová jako `UISplitViewController` a zobrazuje dvě obrazovky vedle sebe. Pokud má horizontální velikostní třídu `compact`, tedy velikost, při které by se dvě obrazovky vedle sebe nevešly, tak se chová prakticky jako `UINavigationController`. Je zobrazena pouze levá obrazovka a při výběru se místo otevření druhé ob-

razovky vpravo zobrazí tato obrazovka přes aktuální a do navigační lišty se přidá tlačítko zpět. Díky tomu je možné na mnoha místech používat pouze `UISplitViewController`, který vždy sám upraví svoje chování na základě aktuální velikosti displeje. [43]

2.2.7 CoreData

Jedním z hlavních požadavků na aplikaci je off-line dostupnost, všechna data stažená ze serveru je tak nutné ukládat lokálně do telefonu. Pro tento případ iOS nabízí databázovou vrstvu CoreData. Základem CoreData je SQLite [52] databáze, do které se všechna data ukládají. Jedná se o jednoduchou databázi, která je celá uložena v jednom souboru a je nejčastěji využívána pro lokální ukládání dat aplikací. Nad touto databází je vystavěna objektová vrstva, pomocí které programátor k databázi přistupuje. Díky této objektové nadstavbě je programátor odstíněn od klasických databázových operací a může s daty rovnou pracovat jako s obyčejnými objekty.

Ke CoreData existuje řada alternativ jako například Realm [46] a další podobné frameworky třetích stran. Přestože tyto nadstavby mohou být v některých případech výhodnější, CoreData mají zásadní výhodu v pokročilé integraci s celým iOS SDK, které obsahuje mnoho nástrojů pro práci s touto vrstvou. Další z výhod je i možnost využití RestKitu, který je založen právě na CoreData. [12]

2.2.8 RestKit

RestKit je opensource framework pro snadnou komunikaci a práci s REST [41] API. Jeho základem je databázová vrstva CoreData a knihovna pro síťovou komunikaci AFNetworking [1]. RestKit tak zapouzdřuje veškerou komunikaci s API i následnou perzistenci dat. Dále obsahuje množství užitečných vlastností jako automatické mazání objektů odstraněných ze serveru, automatickou konverzi textového řetězce na datum a další. Programátor díky tomu pouze definuje mapování mezi odpověďmi ze serveru a CoreData objekty a nemusí se zabývat rutinním, opakujícím se kódem. [58]

Drobnou nevýhodou RestKitu je jeho striktní zaměření na standard REST. Pokud API není striktně RESTful, tak přestože je RestKit velmi přizpůsobitelný a nabízí široké možnosti nastavení, tak je práce s ním velmi nepohodlná. Vzhledem k tomu, že naše navržené API dodržuje všechny standardy REST, tak je použití RestKitu více než vhodné.

2.2.9 NSFetchedResultsController

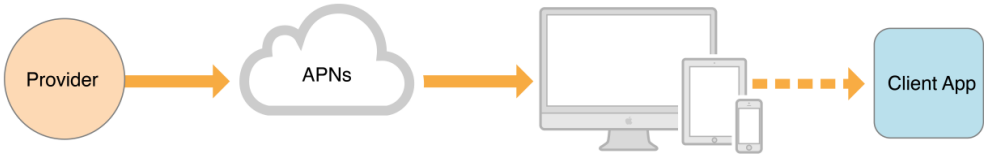
Při využití databázové vrstvy CoreData je možné využít třídy `NSFetchedResultsController`. Jedná se o třídu pro pohodlné a efektivní získávání dat z databáze, které je přizpůsobené pro práci s tabulkami. Kromě

pohodlné práce s tabulkou tato třída trvale sleduje všechny změny na objektech odpovídajících nastavené podmínce a při jakékoliv změně těchto objektů notifikuje tabulku a aktualizuje její obsah. Díky této vlastnosti je obsah na obrazovce neustále udržován v konzistenci s daty v databázi. [9]
`NSFetchedResultsController` je v aplikaci použit na všech obrazovkách, kde se vyskytují tabulky.

2.2.10 Push notifikace

Jedním z požadavků na aplikaci je možnost přijímání push notifikací, které mohou sloužit k upozorňování návštěvníků na změny v programu apod.

Push notifikace je pojem označující krátkou zprávu poslanou serverem konkrétnímu zařízení. Zde je nutné zdůraznit, že komunikace je inicována serverem a nikoli klientem. Základním prvkem fungování notifikací na platformě iOS je APNs (Apple Push Notification Service). Jedná se o službu, která se stará o doručování notifikací na zařízení. Server, který chce odeslat push notifikaci, předá potřebné informace APNs a ten se postará o jejich doručení na zařízení. [15] Tato komunikace je znázorněna na obrázku 2.4.



Obrázek 2.4: Znázornění komunikace při posílání push notifikace na zařízení [15]

Pro identifikaci zařízení se používá `device token`, který je generován systémem iOS. Tento token je unikátní pro každou aplikaci a každé zařízení a jednoznačně určuje, kam má být notifikace doručena. Pro získání tokenu musí aplikace nejdříve požádat systém o oprávnění a až ve chvíli, kdy uživatel notifikace povolí, je aplikaci vrácen `device token`. Tento získaný token je jedinou informací, kterou server potřebuje pro zasílání notifikací na konkrétní zařízení.

Kapitola 3

Návrh implementace

Tato kapitola se zabývá návrhem doménového modelu, z něj pak vychází definice serverového API a poslední částí této kapitoly je návrh wireframes pro všechny části aplikace.

3.1 Doménový model

Návrh doménového modelu je v tomto případě zcela zásadní pro serverovou část, tedy API i pro klientské aplikace. Zejména je potřeba vyřešit problém jednoho společného modelu pro různé druhy akcí. Každý druh akce využívá různé entity, které je třeba zobecnit tak, aby pokryly všechny případy užití. Většina hlavních entit je na první pohled viditelná již z funkčních požadavků, další potom vychází ze specifických případů užití. Jejich význam je popsán dále a diagram se znázorněním vzájemných vztahů je vidět na obrázku 3.1.

3.1.1 Session

Tato entita odpovídá časově omezené události. V případě festivalů se jedná o vystoupení umělce nebo promítání filmu, v případě konferencí se jedná o přednášku. Má vždy svůj čas začátku a konce, koná se na určitém místě a někdo je jejím autorem. Tyto události budou viditelné v časové ose a bude je možné přidávat do osobního programu (plánovat).

Pro všechny typy akcí plní tato entita popsanou roli, ale její využití může být pro různé akce rozdílné. Zatímco u konferencí má význam název události, její popis a další podobné atributy, u filmových a hudebních festivalů nikoli. Přednáška by vždy měla mít název, ale název vystoupení umělce na hudebním festivalu nebo název promítání filmu nemá význam, proto se počítá s tím, že tyto atributy zůstanou pro tyto události prázdné.

3.1.2 Performer

Tato entita odpovídá autorovi jedné nebo více událostí. V případně hudebního festivalu je to vystupující umělec a v případě konference je to přednášející (řečník). V případě filmového festivalu, zde nastává problém určit autora promítání. Ač to z logiky pojmenování entity může být matoucí, tak ideálním řešením je vnímat jako autora promítání samotný promítaný film. Seznam vystupujících se tak pro filmové festivaly stane seznamem filmů.

Kardinalita vazby mezi entitami Performer a Session je obecně definována jako M:N, tedy vystupující může mít několik událostí a naopak i událost několik vystupujících. Tento vztah ale prakticky platí pouze pro případ konference, kde přednášející často má více přednášek, stejně jako přednáška více přednášejících. Pro filmové festivaly je vazba omezena na 1:N, tedy film může mít více promítání, ale promítání má vždy pouze jeden film. Pro hudební festivaly se zde sice v naprosté většině případů jedná o vazbu 1:1, kdy umělec má pouze jedno vystoupení, ale občas se můžeme setkat i s tím, že umělec vystoupí vícekrát, proto je zde ponecháno omezení 1:N stejně jako u filmových festivalů. Opět zde tedy platí omezení, že v rámci jednoho vystoupení vystupuje pouze jeden umělec.

3.1.3 Track

Tato entita představuje místo konání události. Pro různé případy to může být koncertní pódium, kino nebo například přednáškový sál.

Vztah s entitou Session je definován jako 1:N, tedy jednom místě konání se může konat více událostí, zatímco událost se vždy koná pouze na jednom místě.

3.1.4 TrackGroup

Tato entita slouží pouze pro seskupování jednotlivých míst. Zmíněné seskupování se většinou neuplatní v případě hudebních festivalů, ale může být praktické pro filmové festivaly, které se mohou konat ve více městech současně nebo pro konference konané ve více budovách.

Vztah s entitou Track je zřejmý.

3.1.5 InfoSection a InfoItem

Entita InfoSection slouží pouze pro rozdělení informací do sekcí. Informacemi jsou zde myšleny užitečné informace pro návštěvníky akcí jako například informace o parkování, ubytování, stravování apod.

3.1.6 Newie

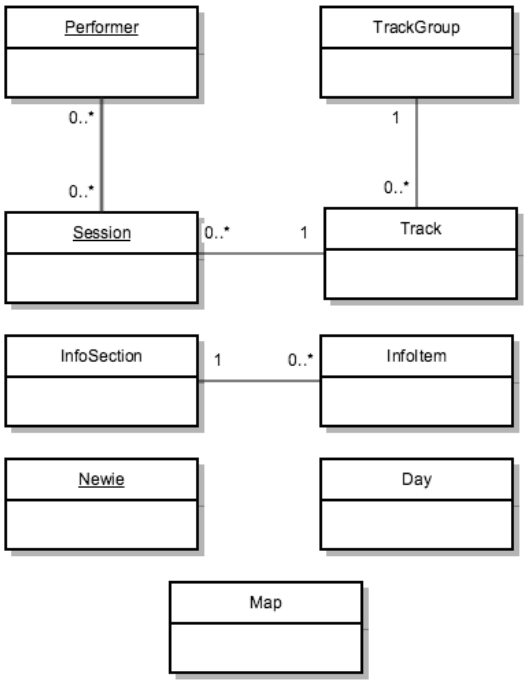
Entita reprezentuje krátkou aktuální zprávu informující návštěvníky o důležitých změnách nebo zajímavostech.

3.1.7 Map

Entita Map se může zdát vzhledem k zadaným požadavkům zbytečná, protože mapa bude pouze jedna ve formě jednoduchého obrázku. Bude ale užitečná při případném budoucím rozšíření map o další možnosti.

3.1.8 Day

Tato entita reprezentuje den konání festivalu. Pokud bychom v aplikaci pracovali s běžnými kalendářními dny, které začínají a končí půlnocí, znamenalo by to nevhodné rozdělení událostí do jednotlivých dnů. Události konající se po půlnoci by byly zařazeny do následujícího dne, přestože v kontextu akce logicky patří ke dni předchozímu. Entita Day tedy definuje festivalový den jeho přesným začátkem a koncem. Entita není ve vztahu s žádnou jinou, bude pouze využívána v některých místech aplikace pro správné zobrazování dat.



Obrázek 3.1: Doménový model

3.2 Definice API

Vzhledem k vytyčeným požadavkům a jednoduchosti navrženého doménového modelu byl pro API zvolen standard REST. Zjevné výhody tohoto standardu jsou rozebrány v další kapitole.

3.2.1 REST

REST je zkratkou pro Representational State Transfer. Jedná se o architekturu rozhraní, kterou v roce 2000 v rámci své disertační práce definoval jeden z autorů HTTP [53] protokolu Roy Fielding. Je to bezstavová architektura, jejíž základním prvkem je zdroj jednoznačně identifikovaný svým unikátním URI [55]. REST definuje čtyři metody přístupu k těmto zdrojům, známé pod označením CRUD (create, read, update, delete), které jsou implementovány pomocí odpovídajících metod HTTP protokolu.

- POST – vytvoření zdroje (create)
- GET – získání zdroje (read)
- PUT – aktualizace zdroje (update)
- DELETE – odstranění zdroje (delete)

Co REST nedefinuje, je reprezentace zdroje, respektive formát reprezentace. Běžně je využíván formát JSON [29], ale může být použito i XML [57] nebo CSV [48].

Rozdíl oproti dalším používaným standardům jako jsou například XML-RPC [59] nebo SOAP [42] je sémantika operací. Oba zmíněné standardy jsou orientovány na vzdálené volání metod, je tedy distribuováno chování, při komunikaci provádíme akce. Oproti tomu v RESTu se dotazujeme na zdroje, přenáší se stav.

Standard REST je na poli mobilních aplikací velmi rozšířený díky své jednoduchosti. Na všech běžných platformách je jeho implementace triviální a pro většinu platforem existuje i mnoho knihovnen či frameworků. Z těchto důvodů je vzhledem ke kladeným požadavkům ideální technologií pro navrhované API. Jako formát reprezentace zdrojů bude využita technologie JSON, která je na mobilních platformách široce podporována. Díky povaze RESTu, může být kdykoliv jednoduše doplněna podpora i pro jiné formáty. [41]

3.2.2 Definované endpointy

Endpoint je v kontextu RESTu bod, ze kterého je API, respektive daný zdroj dostupný. [47] Specifikace REST nepopisuje žádný strukturovaný formát definice či dokumentace (jako například WSDL [31] často používaný v kombinaci

se SOAP), proto je pro tyto účely běžně používán obyčejný text. Každý endpoint má definovanou URL, ze které je popisovaný zdroj dostupný, strukturu zdroje popsanou vzorovou reprezentací a případný popis. Pro zde popisované API jsou definovány následující endpointy.

3.2.2.1 GET /days

Tento endpoint poskytuje seznam dnů, ve kterých se akce koná, včetně jejich přesných začátků a konců.

```
{
    "days": [
        {
            "day": "2014-09-14",
            "start": "2014-09-14T10:00:00+0200",
            "end": "2014-09-15T02:00:00+0200"
        }, {
            "day": "2014-09-15",
            "start": "2014-09-15T08:00:00+0200",
            "end": "2014-09-15T23:00:00+0200"
        }
    ]
}
```

3.2.2.2 GET /program

Tento endpoint poskytuje informace o kompletním programu. Obsahuje seznam všech vystupujících, událostí i míst konání.

Uvedená definice obsahuje pouze základní vlastnosti objektů jako identifikátory, názvy, popisy, časové údaje. V návrhu se počítá s dalším rozšířením vlastností jednotlivých entit a to například v podobě tagů vystupujících pro možnost filtrování či barev míst konání pro lepší orientaci ve výsledné aplikaci. Tyto četné detaily nejsou pro tento návrh stěžejní, proto zde nejsou uvedeny.

Vztahy definované v doménovém modelu jsou zde v případě míst konání a jejich skupin implementovány pomocí vložených objektů. V případě vztahu události s vystupujícím a s místem konání je využito odkazování pomocí indentifikátorů objektů.

Podle zvyklostí REST návrhu by mohly být popisované tři sekce (vystupující, události a místa konání) rozděleny na tři samostatné endpointy, ale jejich sloučení je výhodné jednak z hlediska úspory datové komunikace (jeden větší požadavek je vždy díky nižší režii lepší než více menších) a jednak má svůj význam při zachování konzistence dat. Mohla by nastat situace, že v seznamu událostí přibyde událost vystupujícího, kterého si aplikace ještě nestihla stáhnout z jiného endpointu. To lze samozřejmě řešit vložením reprezentace chybějícího vystupujícího přímo do informace o události namísto

odkazování pouhým ID, což by ovšem vedlo ke zbytečné duplikaci dat a tím i větší datové náročnosti.

```
{
    "performers": [
        {
            "performer_id": 1,
            "name": "Ing. František Vopršálek",
            "description": "Přední odborník v oblasti biomedicíny.",
            "image": "http://example.com/images/510812.jpg"
        },{
            "performer_id": 2,
            "name": "Bc. Marie Sladká",
            "description": "Primářka ORL Nemocnice Motol",
            "image": "http://example.com/images/752809.jpg"
        }
    ],
    "groups": [
        {
            "group_id": 2,
            "name": "Kongerosvé centrum - Pavilon A",
            "tracks": [
                {
                    "track_id": "1",
                    "name": "Přednáškový sál 105",
                    "order": 1
                }, {
                    "track_id": "2",
                    "name": "Zasedací místnost",
                    "order": 2
                }
            ]
        }
    ],
    "sessions": [
        {
            "session_id": 1,
            "track_id": 1,
            "title": "Novinky v oblasti radiologie",
            "time_from": "2014-09-14T10:30:00+0200",
            "time_to": "2014-09-14T12:30:00+0200",
            "performers": [
                {
                    "performer_id": 1
                }
            ]
        }, {
            "session_id": 2,
            "track_id": 2,
```

```
            "title": "Rozpočet Nemocnice Motol pro rok 2015",
            "time_from": "2014-09-14T13:00:00+0200",
            "time_to": "2014-09-14T14:00:00+0200",
            "performers": [
                {
                    "performer_id": 2
                }
            ]
        }
    ]
}
```

3.2.2.3 GET /news

Tento endpoint poskytuje seznam všech novinek. Novinky mají svůj čas vytvoření, titulek, zprávu a ilustrační obrázek.

```
{
    "news":[
        {
            "news_id":2,
            "date":"2014-09-23T19:25:25+0200",
            "title":"Na konferenci vystoupí i světoznámý odborník Smith",
            "text":"Lorem ipsum dolor sit amet, consectetur adipiscing
            elit. Nam et lacus venenatis, euismod nunc mattis, ultrices.",
            "image":"http://example.com/images/752809.jpg"
        }, {
            "news_id":5,
            "date":"2014-09-23T19:25:25+0200",
            "title":"Všechny přednášky bloku X budou zdarma",
            "text":"Lorem ipsum dolor sit amet, consectetur adipiscing
            elit. Nam et lacus venenatis, euismod nunc mattis, ultrices.",
            "image":"http://example.com/images/709459.jpg"
        }
    ]
}
```

3.2.2.4 GET /info

Tento endpoint poskytuje seznam všech informačních sekcí a jako vnořené objekty i všechny prvky těchto sekcí. Do textu informace je často potřeba vložit i obrázky nebo strukturované seznamy, proto může být tento text opatřen HTML [56] značkami. Klientská aplikace musí text interpretovat jako HTML kód.

```
{
    "sections":[
        {
            "name":"Stravování",
```

```
            "section_id":1,
            "info":[
                {
                    "info_id":1,
                    "text":"<h1>Restaurace</h1><p>Restaurace jsou
                    v městečku nedaleko areálu.</p>",
                    "title":"Občerstvení"
                }, {
                    "info_id":2,
                    "text":"V areálu bude k dispozici
                    cisterna s pitnou vodou...",
                    "title":"Pitný režim"
                }
            ]
        }, {
            "name":"Hygiena",
            "section_id":4,
            "info":[
                {
                    "info_id":3,
                    "text":"Toalety budou umístěny po 50m
                    okolo stanového městečka.",
                    "title":"Toalety"
                }
            ]
        }
    ]
}
```

3.2.2.5 GET /maps

Tento endpoint zatím poskytuje pouze jednoduchý odkaz na obrázek, ale je do budoucna připraven pro seznam všech map různých typů. API je tak připraveno například pro situaci, kdy pro akci budou nabízeny tři statické orientační plány a interaktivní mapa. Z toho důvodu je v reprezentaci objektu mapy přítomen zdánlivě nadbytečný atribut `type`.

```
{
    "maps": [
        {
            "map_id": 1,
            "type": 1,
            "image": "http://example.com/map.jpg"
        }
    ]
}
```

3.2.2.6 POST /device

Tento endpoint slouží především pro registraci zařízení k vzdáleným push notifikacím. Pomocí něj klientská aplikace sdělí serveru všechny informace, potřebné pro správnou funkčnost notifikací. Tyto informace se mohou pro různé klientské systémy lišit, protože každý klientský systém může používat vlastní unikátní způsob řešení push notifikací, proto musí klientská aplikace společně s potřebými údaji vždy poslat i typ systému. Údaje nezbytné pro push notifikace na systému iOS jsou společně s principem iOS notifikací blíže popsány v kapitole 2.2.10.

Tento endpoint může být také případně využit pro jednoduché statistiky. Klientská aplikace může k údajům pro push notifikace přidat i další údaje o zařízení jako například informace o verzi aplikace, hardwaru zařízení nebo o verzi operačního systému.

```
1 {
2     "device": {
3         "system": "ios",
4         "device_token": "NSZ16KF7201KD6RTAJU455OFJ10UQMIO",
5         "model": "iPhone 4S"
6     }
7 }
```

3.2.3 Formát data a času

Pro reprezentaci data, byl zvolen standard ISO 8601 z důvodu spolehlivé kompletní informace o datu i přesném čase. Tento standard popisuje formát zápisu času včetně časového pásma, při komunikaci se serverem tedy není potřeba žádná předchozí znalost časového nastavení serveru. Čas a datum v tomto formátu vždy naprosto jednoznačně definuje daný moment v čase. [40]

Tento standard je obecně velmi rozšířený a pro většinu platforem existuje mnoho nástrojů pro práci s tímto formátem, proto je použití typicky bezproblémové.

3.2.4 Hlavičky Last-Modified a If-Modified-Since

Kombinace HTTP hlavičkek `Last-Modified` a `If-Modified-Since` slouží k podmíněným požadavkům na server, což vede k úspoře datového přenosu při komunikaci. Klient při prvním požadavku na daný endpoint dostane odpověď se stavovým kódem `200 OK`, v hlavičce `Last-Modified` datum a čas poslední úpravy zdroje a v těle zprávy reprezentaci příslušného zdroje. Tento čas si klient uloží a při příštím požadavku ho odesílá v hlavičce `If-Modified-Since`. Pokud se zdroj od tohoto času nezměnil, je klientovi zaslána pouze odpověď se stavovým kódem `304 Not Modified` s prázdným tělem zprávy. V případě změny je zaslána běžná odpověď `200 OK`, v hlavičce je zasláno nové datum změny a v těle zprávy reprezentace příslušného zdroje. Tento proces eliminuje

zbytečné přenášení stejných dat a zásadně tak odlehčuje datovou komunikaci, proto bude API tento proces podporovat. [53]

3.2.5 Hlavička X-Only-Modified-Since a inkrementální změny

V případě velkých festivalů, kde mohou být řádově desítky až stovky vystupujících a událostí, je i přes použití podmíněných požadavků popsaných v předchozí kapitole přenášena většina dat zbytečně. Stačí změna času jednoho vystoupení a celý program je přenášen znovu, přestože je změna pouze v jedné události. Z tohoto důvodu je pro API definována vlastní hlavička `X-Only-Modified-Since`, která má podobnou funkci jako `If-Modified-Since`. Zatímco při použití `If-Modified-Since` vrátí API na základě času a data buďto kompletní odpověď nebo stavový kód 304, tak při použití hlavičky `X-Only-Modified-Since` jsou serverem vráceny pouze položky, které byly od definovaného času změněny, namísto kompletní odpovědi. Tím je dosaženo ještě větší úspory dat.

Jednou z výhod této optimalizace je, že je nepovinná. Pokud klientské aplikace z jakéhokoliv důvodu nebudou schopny tuto funkčnost implementovat, API v případě požadavku neobsahujícího hlavičku `X-Only-Modified-Since`, vrátí pokaždé kompletní odpověď.

3.2.6 Odstraňování objektů a synchronizace s lokální databází

Inkrementální změny popsané v předchozí kapitole přinášejí na jedné straně obrovskou úsporu dat, ale na druhé straně přinášejí i komplikace s odstraňováním objektů a synchronizací s lokálními úložišti klientských aplikací. Ve všech odpovědích kromě první jsou obsaženy pouze položky, u kterých proběhly změny, namísto celé kolekce. Není tedy možné porovnat lokálně uloženou kolekci na straně klienta s odpovědí z API a chybějící objekty v odpovědi odstranit i v lokální databázi.

Tento problém je vyřešen přidáním příznaku `deleted` k objektům, které byly na serveru odstraněny. Klientská aplikace na základě tohoto příznaku příslušné objekty odstraní i z lokální databáze. Tento postup přináší potenciální nevýhody, jako například možné zahlcování databáze na serveru z důvodu ponechávání veškerých objektů v databázi nebo pracnější implementace této logiky na serveru. Tyto problémy jsou však pro popisované API zanedbatelné, protože odstraněných objektů bývá v porovnání s jejich celkovým počtem zpravidla velmi málo.

3.2.7 Lokalizace

Vzhledem k požadavku vícejazyčnosti je potřeba zajistit nejen lokalizaci aplikace, ale i lokalizaci dat, která do aplikace přichází ze serveru. Pro tento účel

definuje protokol HTTP hlavičku Accept-Language, ve které klient posílá informace o svých jazykových preferencích. Tato informace je reprezentována jako čárkami oddělený seznam jazykových kódů a jejich vah. Tato hlavička může vypadat například takto:

```
Accept-Language: cs;q=1.0, en-GB;q=0.8, en;q=0.6
```

Klientské aplikace musí posílat všechny požadavky na API s touto hlavičkou a API bude podle těchto priorit vracet lokalizovaná data. [53]

3.3 Wireframes

Wireframes jsou jednou z částí prototypování uživatelského rozhraní, někdy jsou wireframes také označovány jako lo-fi prototyp. Jedná se o znázornění základní představy o rozložení jednotlivých komponent. Nejčastěji bývají wireframes tvořeny buďto jednoduchými kreslícími programy, nebo i kresleny tužkou na papír. Wireframes slouží jednak pro základní představu o funkčnosti systému a jednak jako základ grafického návrhu. [36]

Navržené wireframes jsou zpracovány v programu Balsamiq, což je software určený přímo pro tento účel. Nabízí připravené objekty pro běžné uživatelské prvky jako například tlačítka, lišty či přepínače a obsahuje mnoho pokročilých nástrojů pro pohodlnou tvorbu prototypů. [27]

3.3.1 Mobile first design

Mobile first design je metoda návrhu uživatelského rozhraní, který je využíván při návrhu webových stránek. Jedná se o postup, při kterém je vytvářeno nejprve rozhraní pro malé displeje (mobilní zařízení) a z něj následně vychází návrh pro větší displeje (stolní počítače). Typicky je jednodušší vměstnat všechny zásadní a nezbytné informace na malý displej a s většími displeji pak přidávat více informací a podrobností než naopak. [37]

Při návrhu iOS aplikace pochopitelně není třeba řešit tak široký záběr různých velikostí zařízení jako při návrhu webu. Web musí být správně navržen pro zařízení od malých levných telefonů, přes tablety, až po velké displeje stolních počítačů. Oproti tomu na iOS pracujeme pouze s několika velikostmi iPhonu, které se od sebe liší pouze minimálně, a prakticky s jedinou velikostí iPadu (viz kapitola 2.1.2). I přes toto zásadní zjednodušení je možné metodu mobile first, která je využívána pro weby, aplikovat i při návrhu uživatelského rozhraní pro iOS.

Pokud bychom se měli mobile first principu striktně držet, tak bychom měli začít návrhem pro zařízení s nejmenším displejem, tedy v tomto případě Apple Watch. Jak ale bylo popsáno v kapitole 2.1.3.1, aplikace na Apple Watch je pouze rozšířením mateřské aplikace, tudíž nemusí a ani by neměla nabízet stejný obsah, ale měla by pouze poskytovat rychlý přístup k nejdůležitějším

datům. Wireframes pro Apple Watch tedy nejsou vhodným základem pro mobile first návrh, proto byly prvně navrženy wireframes pro iPhone a z nich potom vychází wireframes pro iPad.

3.3.2 iPhone

Navržené wireframes vychází především z funkčních požadavků vytyčených v kapitole 1.2 a dále také ze zvyklostí platformy iOS. Firma Apple vydává a pravidelně aktualizuje rozsáhlé a podrobné Human Interface Guidelines, ve kterých je popsáno, jak používat dané systémové prvky a jaká pravidla dodržovat, aby byla aplikace dobře použitelná a v souladu se systémem iOS a dalšími aplikacemi. [21]

3.3.2.1 Navigace a menu

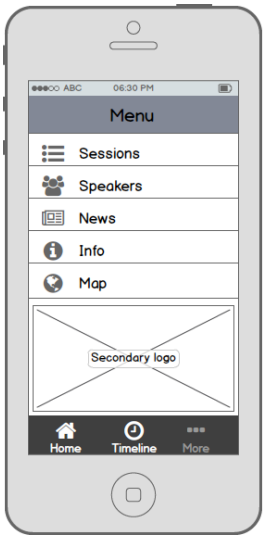
Podle zmíněných HIG jsou doporučeny dva hlavní způsoby navigace. Pro pohyb uživatele hierarchickou strukturou obrazovek je určena horní navigační lišta (`UINavigationController`), která zobrazuje titulek aktuální obrazovky a případné tlačítko zpět pro návrat na předchozí obrazovku. Pro přepínání hierarchicky rovnocenných částí aplikace je určena spodní lišta s ikonami a titulky jednotlivých částí (`UITabBarController`). Na těchto dvou základních modelech je založena i veškerá navigace v navrhované aplikaci.

`UITabBarController` umožňuje přepínání mezi domovskou obrazovkou, časovou osou, seznamem vystupujících, seznamem událostí, novinkami, informacemi a mapou. `UITabBarController` je šířkou displeje omezen v počtu zobrazených položek. V případě, že se všechny položky na displej nevejdou, je tento problém podle HIG řešen nahrazením poslední položky tlačítkem, které vede na zobrazení zbývajících položek formou seznamu. Návrh této obrazovky je zobrazen na obrázku 3.2. [21]

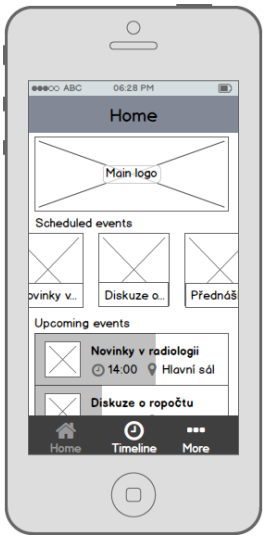
Všechny položky mají ilustrační ikonu a titulek. Výběrem některé z položek se dostaneme na odpovídající obrazovku. Pod seznamem položek je pak prostor pro klientskou grafiku, například loga sponzorů apod.

3.3.2.2 Domovská obrazovka

Tato obrazovka je zobrazena ihned při spuštění aplikace a představuje stručný a přehledný pohled na to, co se aktuálně děje. Na domovské obrazovce tedy uživatel ihned vidí svůj naplánovaný program ve formě horizontálně posouvatelné tabulky a pod ním i seznam všech nadcházejících událostí. Částečné podbarvení řádku události indikuje její průběh. Výběrem některé z naplánovaných nebo nadcházejících událostí se uživatel dostane na obrazovku detailu vybrané události. Návrh této obrazovky je zobrazen na obrázku 3.3.



Obrázek 3.2: Návrh obrazovky menu pro iPhone



Obrázek 3.3: Návrh domovské obrazovky pro iPhone

3.3.2.3 Časová osa

Obrazovka s časovou osou nahrazuje klasické papírové zobrazení programu. Jedná se o tabulku všech událostí, na jejíž horizontální ose jsou události rozmístěné podle času a na vertikální ose podle místa konání. Tato tabulka vždy zobrazuje právě probíhající den. Na jiné dny je možno přepnout tlačítkem v levé části navigační lišty, které zároveň uživatele informuje, který den je právě zobrazen. Jako znázornění aktuálního času slouží poloprůhledná vrstva,

která zakrývá uplynulé události. Výběrem některé z událostí se uživatel dostane na obrazovku detailu vybrané události. Návrh této obrazovky je zobrazen na obrázku 3.4.

Obrázek 3.4: Návrh obrazovky s časovou osou pro iPhone

3.3.2.4 Seznam vystupujících

Tato obrazovka zobrazuje v jednoduché tabulce všechny vystupující seřazené podle jména včetně jejich obrázků a krátkého titulku. Tažením prstem přes řádek je zobrazeno tlačítko pro přidání vystupujícího do osobního programu. V případě, že má vystupující přiřazenu pouze jednu událost, je do programu přidána tato jediná událost. V případě více událostí, je uživateli zobrazeno okno se seznamem událostí, ze kterých si může vybrat.

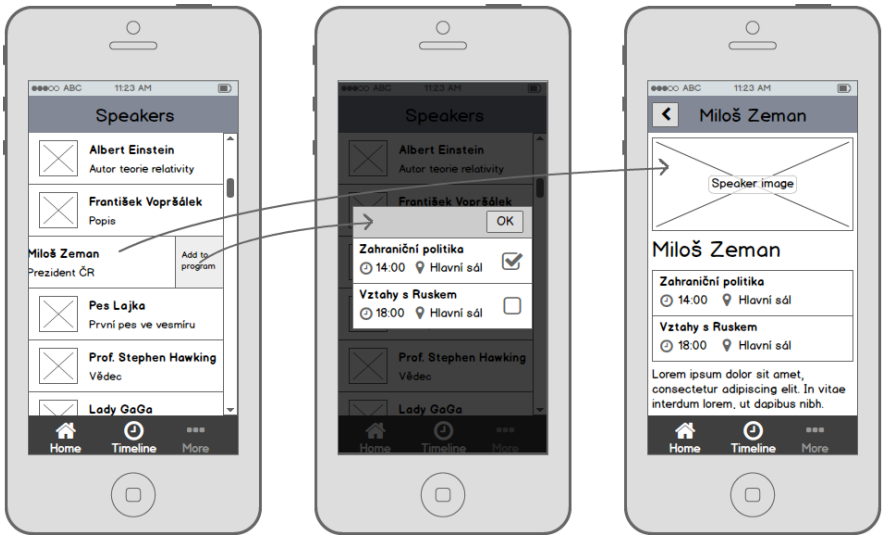
Výběrem vystupujícího se uživatel dostane na zobrazení jeho detailu. Na obrazovce detailu je pak větší obrázek následovaný jménem vystupujícího, seznamem všech jeho událostí, popisem a v budoucnu případnými dalšími detaily. Výběrem některé z událostí se může uživatel dostat na její detail.

Návrh popsaných dvou obrazovek je zobrazen na obrázku 3.5.

3.3.2.5 Seznam událostí

Hned ze začátku je nutno podotknout, že tato obrazovka, stejně jako její odpovídající položka v Tab Baru a obrazovka detailu události, bude přítomna pouze v případě konferencí. Jak u hudebního, tak u filmového festivalu by toto zobrazení nemělo význam. Důvody jsou popsány v kapitole 3.1.1.

Obrazovka obsahuje seznam všech událostí seřazených podle názvu ve formě jednoduché tabulky. Stejně jako v seznamu vystupujících je i zde možné



Obrázek 3.5: Návrh obrazovky seznamu vystupujících a detailu vystupujícího pro iPhone

tažením prstu zobrazit tlačítko pro přidání do programu a přidat si vybranou událost do osobního programu. Pokud událost právě probíhá, je řádek události částečně podbarven. Šířka tohoto podbarvení odpovídá aktuálně uplynulému času události. Toto podbarvení se objevuje i na domovské obrazovce v seznamu nadcházejících událostí.

Výběrem události je uživateli zobrazen její detail. Obrazovka detailu události je v mnohém podobná obrazovce detailu vystupujícího. Obsahuje ilustrační obrázek, název přednášky, detaily týkající se času a místa konání, seznam všech vystupujících v rámci této události, její popis a případné další detaily.

Návrh popsaných dvou obrazovek je zobrazen na obrázku 3.6.

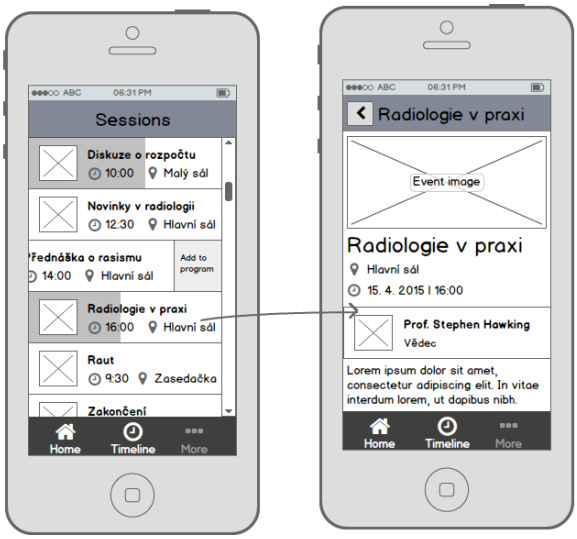
3.3.2.6 Novinky

Obrazovka zobrazuje všechny novinky pod sebou seřazené od nejnovější. U každé novinky je uveden její datum a čas přidání, nadpis, ilustrační obrázek a její text. Novinky jsou zamýšleny pouze jako krátké sdělení, proto na této obrazovce uživatel není zbytečně obtěžován případným otevíráním obrazovky s detailem novinky, ale vidí kompletní obsah novinek najednou.

Návrh obrazovky novinek je zobrazen na obrázku 3.7.

3.3.2.7 Informace

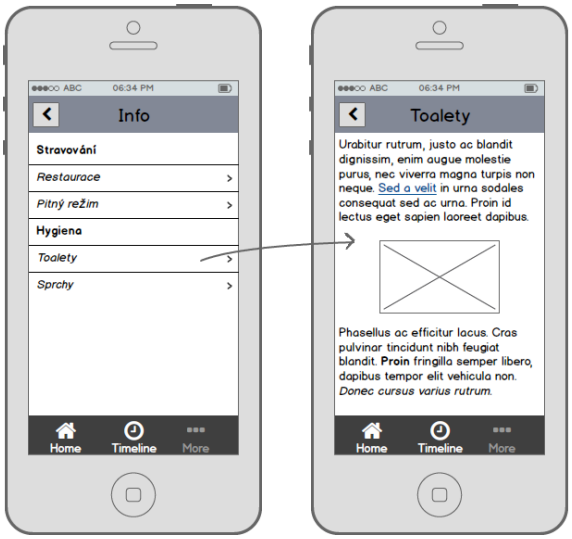
Struktura informací popsaná v požadavcích i v navrženém doménovém modelu se odráží i na této obrazovce. Jedná se o tabulku rozdělenou do sekcí podle



Obrázek 3.6: Návrh obrazovky seznamu událostí pro iPhone

Obrázek 3.7: Návrh obrazovky novinek pro iPhone

jednotlivých kategorií informací. Výběrem některé z informací je zobrazena obrazovka detailu s jejím obsahem. Obrazovka detailu informace zobrazuje pouze interpretovaný HTML kód (uvedeno v kapitole 3.2.2.4) a kompletní vzhled i obsah tedy záleží na konkrétních datech. Návrh této obrazovky je zobrazen na obrázku 3.8.

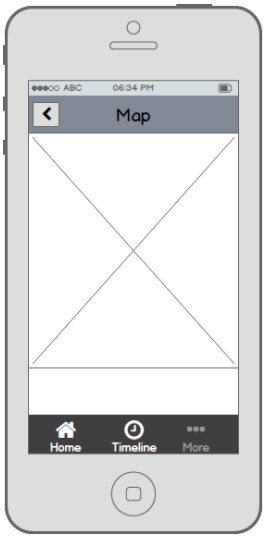


Obrázek 3.8: Návrh obrazovky informací pro iPhone

3.3.2.8 Mapa

Obrazovka mapy je prozatím velmi jednoduchá, obsahuje mapu ve formě obrázku, který je možné standardním systémovým gestem zvětšovat či zmenšovat a tažením prstu obrázek posouvat. Do budoucna se zde počítá s možností interaktivní mapy namísto statického obrázku, v takovém případě by byl návrh obdobný.

Návrh této obrazovky je zobrazen na obrázku 3.9.



Obrázek 3.9: Návrh obrazovky s mapou pro iPhone

3.3.3 iPad

Návrhy wireframes pro iPad vychází podle použitého principu mobile first popsaného v kapiotle 3.3.1 z velké části z návrhů pro iPhone. Ty jsou pouze upraveny tak, aby dobře využívaly větší displej a tím i větší prostor pro obsah na iPadu. Uživatel díky tomu může vidět více obsahu a detailů bez nutnosti posouvání nebo přepínání obrazovek.

3.3.3.1 UISplitViewController

Základním prvkem navrženého uživatelského rozhraní pro iPad je stejně jako na iPhonu `UITabBarController`. K hierarchické navigaci zde ale slouží namísto samotného `UINavigationControlleru` jeho kombinace s `UISplitViewControllerem`. Ten na iPadu obecně umožňuje zobrazovat zároveň obrazovku se seznamem objektů a obrazovku s detailem vybraného objektu. Uživatel se tak při potřebě přepnutí právě zobrazeného detailu nemusí vracet na předchozí obrazovku se seznamem a znovu otevírat detail, ale může ze stále zobrazeného seznamu rovnou otevírat jinou obrazovku detailu.

Tento způsob navigace dobře využívá větší prostor na displeji iPadu a nabízí uživatelům větší komfort při procházení obsahu.

3.3.3.2 Menu

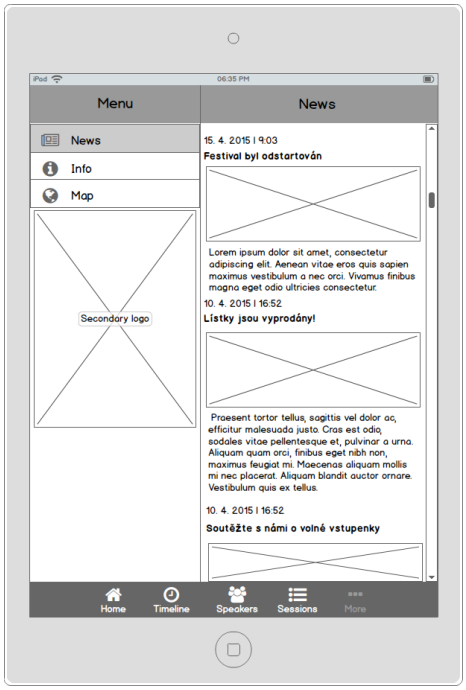
Obrazovka menu zobrazující seznam položek, které se nevešly do Tab Baru, je na iPadu, stejně jako další obrazovky, řešena pomocí `UISplitViewControlleru`. Na pozici vlevo se nachází obrazovka se seznamem položek a po výběru položky se pak vybraná obrazovka zobrazí na pozici vpravo. Rozložení obrazovek zůstává jak u menu, tak u ostatních obrazovek stejné jako na iPhonu, pouze jsou na displeji zobrazeny dvě obrazovky zároveň.

Toto řešení je zobrazeno na obrázku 3.10.

3.3.3.3 Další obrazovky

Na domovské obrazovce je nejprve zobrazena zcela totožná obrazovka jako na iPhonu a až po případném výběru některé z naplánovaných nebo nadcházejících událostí se obrazovka rozdělí do `UISplitViewControlleru`, přičemž vlevo zůstává zmenšená domovská obrazovka a vpravo detail vybrané události. Stejný postup platí i pro obrazovku s časovou osou. V obou případech může být detail skryt tlačítkem v navigační liště a levá obrazovka je zpět roztažena přes celou šířku. Tento model navigace je znázorněn na obrázku 3.11.

Obrazovky se seznamem vystupujících a seznamem událostí jsou taktéž řešeny pomocí `UISplitViewControlleru`. Na pozici vlevo je vždy seznam vystupujících, respektive událostí a na pozici vpravo pak detail vybraného vystupujícícho, respektive detail události.



Obrázek 3.10: Řešení menu na iPadu

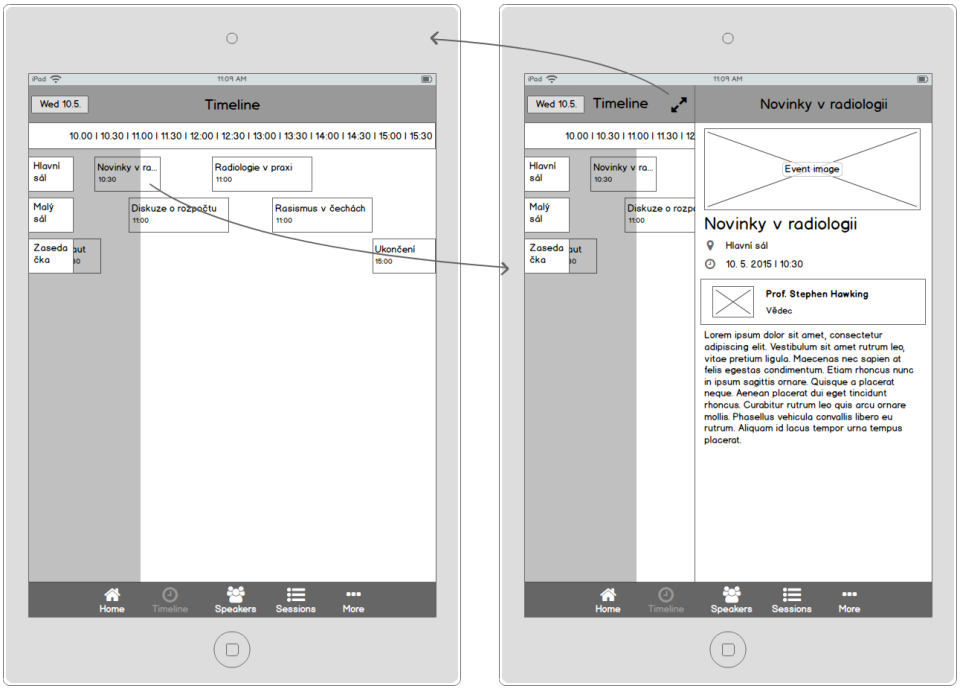
Tento návrh jednak dobře využívá větší displej iPadu a jednak umožňuje použití již hotových obrazovek pro iPhone. Díky tomu při implementaci nebude nutné programovat obrazovky pro iPad zvlášť.

Návrh obrazovky se seznamem vystupujících a zobrazeným detailem je zobrazen na obrázku 3.12.

3.3.4 Apple Watch

Při návrhu wireframes pro hodinky bylo v prvé řadě přihlíženo k malé velikosti displeje, která činí přibližně pouhé 4 cm (viz kapitola 2.1.2). Dalším aspektem, který je nutno brát v potaz, je předpokládaná délka uživatelské interakce. Přestože se jedná o nové zařízení a nejsou tedy dostupná žádná relevatní data popisující tuto skutečnost, můžeme předpokládat, že délka interakce s hodinkami se bude pohybovat v řádech jednotek, maximálně desítek sekund. Jednak ovládání hodinek zaměstnává vždy obě ruce (na jedné má uživatel hodinky a druhou je ovládá), což může být nepohodlné a jednak má uživatel vždy u sebe i iPhone. Proto pro jakoukoliv operaci vyžadující více času (čtení textů apod.) je pro uživatele ve všech směrech jednodušší využít právě iPhone.

Právě kvůli těmto limitacím neobsahují navržené obrazovky žádné místo pro klientskou grafiku, je minimalizováno množství ilustračních obrázků a i z hlediska obsahu je aplikace velmi okleštěná.



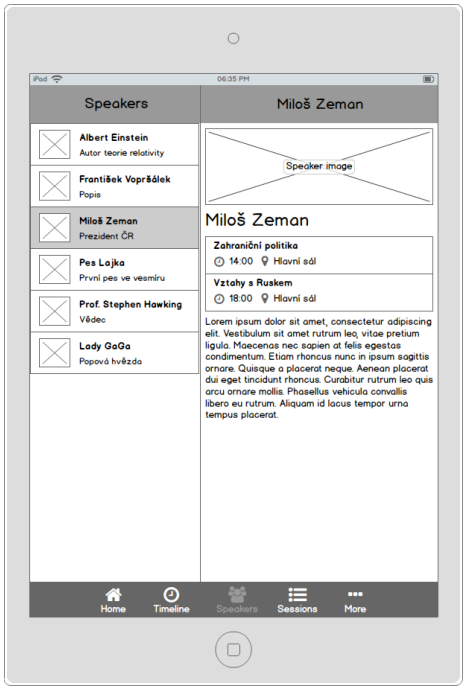
Obrázek 3.11: Obrazovka s časovou osou pro iPad

Aplikace na Apple Watch se mohou objevovat ve třech různých podobách. První je samotná aplikace, kterou uživatel otevírá z nabídky aplikací. Ta může obsahovat celou hierarchii obrazovek s komplexním uživatelským rozhraním, které uživateli umožňuje s aplikací interagovat. Druhou podobou je glance rozhraní, což je jedna informační obrazovka zobrazující aktuální stav. Uživatel se do galerie glance obrazovek všech svých aplikací dostane dotykovým gestem z domovské obrazovky hodinek. Tyto obrazovky nelze posouvat a nelze s nimi ani jakkoli interagovat. Poslední podobou je přizpůsobitelná notifikační obrazovka. Pokud aplikace na hodinkách přijme notifikaci, zobrazí se tato nadefinovaná obrazovka naplněná informacemi obsaženými v notifikaci. [18] Navržené wireframes znázorňují funkčnost, vzhled a obsah těchto třech částí.

3.3.4.1 Aplikace

V zájmu jednoduchosti rozhraní a snahy rychle poskytnout uživateli pouze nejdůležitější informace je obsah omezen pouze na seznam nadcházejících událostí. Jako další možný obsah pro rychlé zobrazení na hodinkách se nabízí seznam novinek. Tato sekce je na hodinkách vynechána, protože všechny novinky budou chodit i ve fromě notifikací, tudíž uživatel o tyto informace nepřijde a aplikace na hodinkách zůstane maximálně jednoduchá.

Seznam nadcházejících událostí je výchozí obrazovkou po spuštění aplikace



Obrázek 3.12: Obrazovka se seznamem vystupujících pro iPad

a je možné v něm zobrazit buďto všechny nadcházející události, nebo vyfiltrovat pouze uživatelem naplánované události. Přepínání mezi všemi událostmi a naplánovanými je realizováno pomocí standardního kontextového menu, které může uživatel vyvolat použitím ForceTouch [25]. Částečné podbarvení řádku slouží jako indikátor průběhu události stejně jako na iPhonu.

Výběrem události ze seznamu je uživateli zobrazen detail této události. Ten obsahuje ilustrační obrázek, název, údaje o času a místě konání a popis. I zde je možno použitím ForceTouch vyvolat kontextovou nabídku, která uživateli umožní naplánovat si událost do osobního programu nebo ji naopak odstranit.

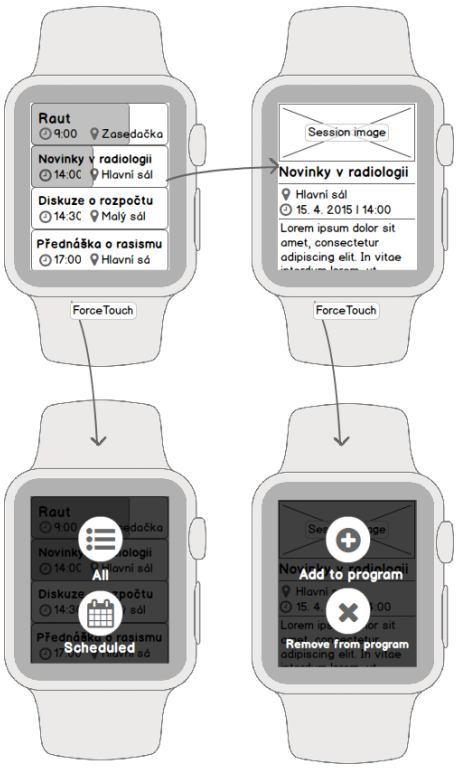
Návrh tohoto rozhraní včetně znázornění uživatelské interakce je zobrazen na obrázku 3.13.

3.3.4.2 Glance rozhraní

Tato informační obrazovka zobrazuje pouze údaje o nejbližší události z uživatelova osobního programu. Obsahuje název události, zbývající čas do začátku události a místo konání. V případě, že některá z naplánovaných událostí právě probíhá, je místo zbývajícího času zobrazen průběžný čas trvání události.

Pokud má uživatel po zobrazené události naplánovanou další, je ve spodním řádku stručná informace o času začátku a názvu této události.

Návrh tohoto rozhraní je zobrazen na obrázku 3.14.



Obrázek 3.13: Návrh rozhraní hlavní aplikace pro Apple Watch

Obrázek 3.14: Návrh Glance rozhraní pro Apple Watch

3.3.4.3 Notifikační obrazovka

Aplikace bude zobrazovat dva druhy notifikací. Prvním druhem je lokální upozornění na uživatelem naplánovanou událost a druhým druhem je push notifikace posílaná serverem, která slouží k oznámením o změnách v programu apod. Pro lokální notifikace je navržená obrazovka obsahově velmi podobná

glance rozhraní. Uživatel na ní může vidět, za jak dlouho daná událost začíná a kde. Z hlediska rozložení UI prvků je obrazovka z velké části definována systémem. Každá notifikace je vždy zobrazena se stejnou hlavičkou, obsahující ikonu aplikace a název aplikace. Dále následuje prostor pro vlastní konfigurovatelný obsah a obrazovku uzavírá opět společná část se systémovým tlačítkem ke skrytí notifikace. Návrh této obrazovky je zobrazen na obrázku 3.15.

Pro push notifikace ze serveru bude použit výchozí systémový vzhled, protože pro ně nebyla definována žádná konkrétní struktura a aplikace tak očekává pouze jednoduchý textový řetězec.

Obrázek 3.15: Rozhraní lokální notifikace pro Apple Watch upozorňující na nadcházející událost

Kapitola 4

Popis implementace

První část implementace se věnuje datovému modelu a obecně funkční logice aplikace. Jedná se například o definici modelových tříd na úrovni CoreData, nastavení RestKitu a jeho mapovacích funkcí, plánování notifikací či automatickou aktualizaci dat. Druhá část implementace je zaměřena na uživatelské rozhraní a zabývá se především napojením UI prvků na datový model, jejich správným rozložením na obrazovkách, animacemi, přizpůsobením aplikace pro různé velikosti displejů a v neposlední řadě i lokalizací. Poslední částí implementace je aplikace pro Apple Watch. Tato část se týká hlavně sdílení dat a funkční logiky s mateřskou aplikací a definice uživatelského rozhraní.

V následujících kapitolách jsou některé důležité nebo technicky zajímavé části implementace blíže popsány.

4.1 Architektura MVC

Celková architektura aplikace vychází z návrhového vzoru MVC (Model-View-Controller). K používání tohoto návrhového vzoru přímo nabádá jednak architektura vlastních systémových tříd a nástrojů a jednak explicitně i firma Apple ve svých doporučeních pro vývojáře. [24] Aplikace je tak rozdělena na modelové třídy, které zajišťují veškerou logiku, controllery obsluhující všechny uživatelské akce a view, které slouží pouze k prezentování příslušného obsahu. Takto oddělená logika aplikace od zobrazení umožňuje znovupoužití kódu na více místech, což je dobře viditelné například při implementaci aplikace pro Apple Watch, která je popsána v kapitole 4.11.

4.2 Datový model

Základem pro třídy datového modelu je navržený doménový model. Základní entity patrné z doménového modelu jsou pouze doplněny o potřebné atributy a metody upravující chování entit na základě typu akce. Například metoda

entity Session (událost) pro název události vrací v případě konference skutečný název přednášky a v případě festivalu název vystupujícího, protože pro případ festivalu název události nedává smysl a není tedy vyplněn. Stejná situace platí například i pro obrázky událostí.

Všechny entity datového modelu mají společnou nadtřídu BaseEntity, která definuje společné atributy pro všechny svoje podtřídy. To jsou především atribut `deleted` sloužící pro odstraňování objektů popsané v kapitole 3.2.6 a atribut `lastUpdated`, který slouží k uchování informace o poslední aktualizaci konkrétního objektu.

4.3 Automatická aktualizace dat

Alespoň základní data jsou pro aplikaci naprosto nezbytná a bez prvního úspěšného stažení dat tedy aplikace nemůže být spuštěna. Těmito nezbytnými daty jsou dny konání akce a informace o programu, tedy seznam vystupujících, událostí a míst konání. Při spuštění aplikace je zkontrolována existence těchto nezbytných dat. Pokud nejsou k dispozici, uživatel je upozorněn a není do aplikace vpuštěn, dokud se nezbytná data nepodaří úspěšně stáhnout.

Všechna data v aplikaci je žádoucí udržovat co nejaktuálnější, ale zároveň je třeba šetřit datový provoz, proto je potřeba držet tyto dva protichůdné požadavky v rovnováze. Stahování dat při každém vstupu na jakoukoliv obrazovku by bylo příliš datově náročné, proto je místo okamžitého stažení nejprve zkontrolována aktualita zobrazovaných dat pomocí atributu `lastUpdated` a v případě překročení určeného intervalu stáří jsou teprve data automaticky aktualizována. Tyto intervaly jsou uloženy v kódu jako konstanty a jsou nastaveny individuálně pro každou entitu podle důležitosti a odhadovaného intervalu změny dat na serveru. Interval pro novinky je například nastaven na 15 minut zatímco interval pro aktualizaci programu, který se pravděpodobně bude v průběhu akce měnit velmi málo, je nastaven na 120 minut. Aktualizaci všech obrazovek je také možné vynutit ručně pomocí standardního gesta - popotažením obsahu obrazovky směrem dolů.

Toto opatření sníží nejen objem přenášených dat, ale i spotřebu energie, protože právě síťové požadavky jsou pro zařízení velmi energeticky náročné.

4.4 Podmíněné požadavky

Kromě ukládání časů aktualizace jednotlivých objektů pro automatickou obnovu dat jsou navíc pro každý endpoint ukládány časy posledních změn na serveru přicházející v hlavičkách `Last-Modified`. Tyto uložené časy jsou pak použity při podmíněných požadavcích v hlavičkách `If-Modified-Since`. Ač by pro tento účel teoreticky mohly stačit právě časy poslední aktualizace objektů, podle specifikace HTTP je vhodné posílat časy příchozí v `Last-Modified` hla-

vičkách kvůli možné desynchronizaci času mezi klientským zařízením a serverem.

Příchozí odpověď od serveru se stavovým kódem `304 Not Modified` dokáže automaticky řešit RestKit a z pohledu programátora se v tu chvíli chová totožně jako při odpovědi `200 OK`, proto není třeba tuto situaci jakkoliv řešit. Implementace z hlediska odesílání je triviální, ke každému požadavku je pouze přidána hlavička s uloženým časem odpovídajícímu danému endpointu.

4.5 Odstraňování objektů a inkrementální změny

Implementace inkrementálních změn popsaných v kapitole 3.2.5 i synchronizace lokální databáze popsaná v kapitole 3.2.6 je díky vlastnostem RestKitu triviální.

Z hlediska synchronizace databáze RestKit umí porovnat obsah lokální databáze s příchozí odpovědí a podle nalezených rozdílů databázi synchronizovat automaticky. Druhou možností je definovat atribut, podle kterého se rozhodne, zda má být objekt smazán či nikoli. Z důvodu inkrementálních změn je nutné využít druhý popisovaný postup. Díky definovanému atributu `deleted`, který mají všechny objekty v lokální databázi (viz kapitola 4.2), RestKit po vyhodnocení každého síťového požadavku z databáze automaticky odstraní všechny takto označené objekty. [58]

S inkrementálními změnami si RestKit dokáže poradit automaticky sám díky popsanému nastavení mazání objektů. RestKitu při tomto nastavení nevadí, když ze serveru přijdou pouze změněné položky namísto kompletní kolekce, protože nepromazává databázi na základě porovnání kolekcí, ale na základě definovaného atributu. Z hlediska odesílání požadavků jsou v hlavičce `X-Only-Modified-Since` odesílány stejné časy jako v hlavičkách `If-Modified-Since` popsaných v předchozí kapitole.

4.6 API

Implementace API není součástí této práce, proto bylo při implementaci síťové komunikace vycházeno pouze z definice uvedené v sekci 3.2. Pro testování funkčnosti v průběhu vývoje byla použita služba Apiary. Ta je primárně určena pro dokumentování API, ale kromě toho umožňuje i vytvoření statického testovacího API proti kterému je možno vyvíjet a testovat. [3] Podle definice byly v Apiary nastaveny všechny endpointy a do celého API byla doplněna testovací data. Díky tomu bude po dodání finální implementace API stačit pouze změnit kořenovou adresu.

4.7 Plánování lokálních notifikací

Podle vytyčených požadavků si uživatel aplikace může plánovat události do svého osobního programu a následně je na začátky těchto událostí upozorňován lokálními notifikacemi. Pro plánování lokálních notifikací a obecně pro práci s nimi existuje v iOS jednoduchý soubor metod a implementace je tedy v tomto směru triviální. Je vytvořen objekt `UILocalNotification` [20], kterému je nastaven text notifikace a čas upozornění. Takto vytvořená notifikace je předána systému iOS, který si ji uloží a následně se postará o její vyvolání v určený čas.

Kromě popsaného jednoduchého plánování je ale potřeba vyřešit i potenciální nekonzistenci dat při práci s notifikacemi. V praxi může dojít k situaci, kdy se čas události, pro kterou je již v systému naplánovaná notifikace, na serveru změní. Tato změna se díky automatické aktualizaci dat ze serveru správně promítne i do databáze, ale naplánovaná notifikace v systému zůstane nezměněna.

Tento problém je vyřešen přidáním identifikátoru události k notifikaci předtím, než je předána systému k naplánování. Při aktualizaci dat ze serveru jsou pak všechny naplánované notifikace pomocí přidaných identifikátorů porovnány s odpovídajícími událostmi a upraveny, případně úplně odstraněny, pokud je událost zrušena.

4.8 Grafický návrh

Kompletní grafický návrh byl podle zadání dodán vedoucím práce. Dodaná grafika vychází z navržených wireframes a respektuje vytyčené požadavky, zejména požadavek na snadné přizpůsobení vzhledu pro konkrétního klienta. Celý grafický návrh je postaven na sadě pěti barev a každý UI prvek má definovanou barvu z této sady.

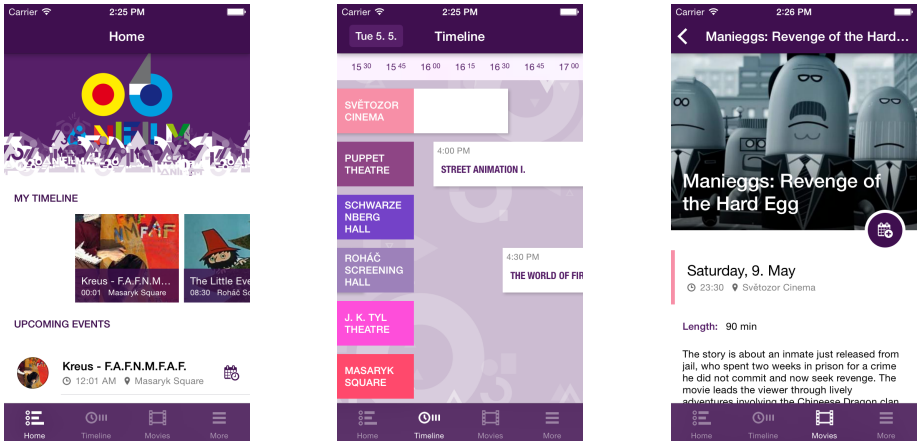
Všechny barvy z popsané sady jsou v kódu definovány jako globální konstanty a pro přebarvení aplikace pro daného klienta tak stačí pouze tyto konstanty změnit.

Dále jsou ve wireframes patrná dvě místa pro klientskou grafiku a to na domovské obrazovce, kde se počítá s hlavním logem akce a na obrazovce menu, kde se počítá s prostorem pro doplňkovou grafiku nebo například loga sponzorů. Tato vyhrazená místa jsou zachována i v grafice a při úpravě pro konkrétní akci stačí pouze vyměnit obrázky.

Ukázka výsledného grafického návrhu je zobrazena na obrázku 4.1.

4.9 Časová osa

Implementace časové osy je založena primárně na komponentě `UIScrollView` [20]. `UIScrollView` je třída umožňující zobrazovat horizontálně i vertikálně



Obrázek 4.1: Grafický návrh vycházející z navržených wireframes

posuvný obsah a je základním prvkem pro zobrazování obsahu, který svou velikostí přesahuje velikost zobrazovací plochy.

Na obrazovce s časovou osou je z hlediska implementace zajímavé použití třech `UIScrollView` najednou. Při pohledu na návrh časové osy na obrázku 3.4 je vidět, že ideální řešení je ponechat levou část s názvy míst konání stále viditelnou stejně jako horní osu s údaji o čase, aby uživatel měl nezávisle na posouvání obsahem stále přehled, kde se nachází. Tento požadavek je vyřešen právě trojící `UIScrollView`. Jedno obsahuje pouze popisky míst konání, druhé lištu s osou času a třetí obsahuje všechny události. Všechna tři `UIScrollView` jsou obsahově zarovnána k sobě a posouvají se všechna zároveň. Toto synchronizované posouvání je realizováno pomocí sledování posuvu hlavního `UIScrollView` s událostmi. Při každém byť i minimálním posunu hlavního `UIScrollView` je hodnota tohoto posuvu nastavena i zbylým dvěma `UIScrollView` a osa času se tak plynule horizontálně posouvá společně s obsahem. Stejně tak se s obsahem vertikálně posouvají i bloky s místy konání.

4.10 Lokalizace

iOS obsahuje velmi dobrou vestavěnou podporu pro lokalizaci aplikací. Xcode automaticky vytvoří potřebné jazykové soubory pro zvolené jazyky, které jsou formátovány jako jednoduchý slovník v podobě klíč – hodnota. V kódu je pak lokalizovaný text získán pomocí makra `NSLocalizedString` [20], kterému je jako argument předán zmíněný klíč odkazující na hodnotu v souboru s lokalizovanými texty. Systém iOS vybere soubor se správným jazykem na základě uživatelem definovaných priorit jazyků v nastavení telefonu. Přidání dalšího jazyka znamená pouze přidání souboru s překlady a jednoduchá úprava v nastavení projektu.

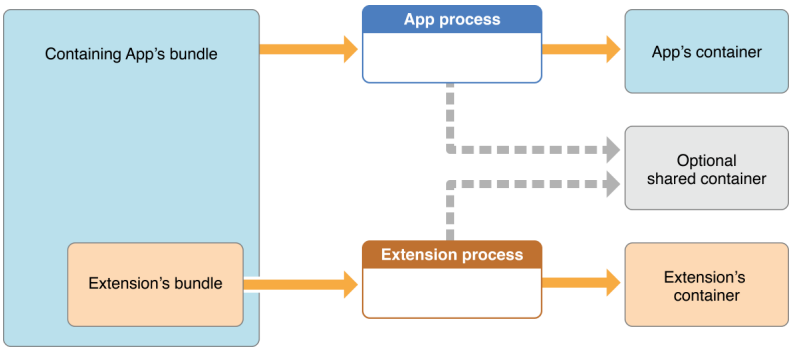
Lokalizace dat z API je řešena posíláním všech požadavků s hlavičkou Accept-Language, což je blíže popsáno v kapitole 3.2.7. Tato hlavička obsahuje seznam preferovaných jazyků podle nastavení telefonu. Na základě této hlavičky API vrátí odpověď v požadovaném jazyce. iOS tuto hlavičku automaticky přidává ke všem odchozím síťovým požadavkům, tudíž programátor se lokalizací dat z API nemusí zabývat.

4.11 Apple Watch

Implementace aplikace pro Apple Watch je zajímavá především z hlediska sdílení kódu a dat mezi mateřskou aplikací a WathKit Extension. Přestože obě tyto části jsou, na rozdíl od WatchKit App, fyzicky uloženy v telefonu, sdílení mezi nimi není samozřejmé. Z pohledu systému je totiž WatchKit Extension vnímána jako samostatná aplikace a nemá tak přístup k datům ani kódu mateřské aplikace. Tento problém řeší funkce AppGroups popsaná v následující kapitole.

4.11.1 AppGroups

AppGroups umožňuje definovat kontejner, který může být sdílený mezi více aplikacemi. Ke zmíněnému kontejneru mají všechny tyto aplikace právo čtení i zápisu a díky tomu je možné sdílet mezi nimi data. Tento sdílený kontejner je vždy definován svým jednoznačným identifikátorem, pomocí kterého k němu aplikace přistupují. [13] Struktura tohoto sdílení je dobře patrná z obrázku 4.2.



Obrázek 4.2: Sdílení kontejneru pomocí AppGroups [13]

Cestu k tomuto kontejneru lze pak jednoduše získat voláním následující metody na objektu `[NSFileManager defaultManager]`. [20]

```
- (NSURL *)containerURLForSecurityApplicationGroupIdentifier:
(NSString *)groupIdentifier
```

4.11.2 Sdílení aplikační logiky

Sdílení kódu mezi mateřskou aplikací a WatchKit Extension může být řešeno dvěma způsoby.

Prvním způsobem je připojení souborů s kódem, který má být společný, jak k mateřské aplikaci, tak k WatchKit Extension. Tento způsob je jednoduchý a umožňuje běh totožného kódu v obou částech. Může ale dojít k situaci, kdy kód určený původně pro mateřskou aplikaci nelze připojit k WatchKit Extension z důvodu použití metod, které ve WatchKit Extension nejsou podporovány. Příkladem budiž výše popsané plánování lokálních notifikací.

```
[[UIApplication sharedApplication] scheduleLocalNotification:
notification];
```

Objekt `[UIApplication sharedApplication]` nutný k naplánování notifikace ve WatchKit Extension není dostupný a aplikaci by tak nebylo možno přeložit.

Tento problém lze řešit druhým způsobem, kterým je delegace úkonů, které WatchKit Extension například nemůže z důvodu některých omezení vykonat, do mateřské aplikace. Kromě řešení těchto omezení může tento způsob sloužit právě i ke sdílení logiky. Tato delegace je realizována pomocí následující metody.

```
+ (BOOL)openParentApplication:(NSDictionary *)userInfo
                        reply:(void (^)(NSDictionary *replyInfo,
                                        NSError *error))reply
```

Jak napovídá název metody, je přímo otevřena mateřská aplikace a v argumentu je jí předán objekt s informacemi nutnými pro vykonání požadovaného úkonu. Na straně mateřské aplikace je na základě tohoto volání spuštěna metoda `handleWatchKitExtensionRequest`.

```
- (void)application:(UIApplication *)application
handleWatchKitExtensionRequest:(NSDictionary *)userInfo
              reply:(void (^)(NSDictionary *replyInfo))reply
```

V této metodě je podle předané informace z argumentu nutné rozpoznat, jaký úkon má být vykonán, následně je spuštěn požadovaný kód v rámci mateřské aplikace a po jeho dokončení je do WatchKit Extension odeslána odpověď.

Oba popsané způsoby jsou v implementované aplikaci využity. První způsob je uplatněn například pro sdílení modelových tříd nebo třídy pro inicializaci CoreData. Způsob druhý je využit při plánování události do osobního programu a to jednak z důvodu již zmíněné nemožnosti plánování notifikací z WatchKit Extension a jednak z důvodu žádoucího sdílení kódu. [20]

4.11.3 Sdílení databáze

Jak bylo popsáno v kapitole 2.2.7, databázová vrstva CoreData je postavena nad souborovou databází SQLite. Soubor této databáze je běžně vytvořen při prvním spuštění aplikace v jejím kontejneru. Pro sdílení celé databáze je potřeba zajistit právě sdílení tohoto souboru, což nám umožňují výše popsané AppGroups. Při vytváření databázového souboru tak stačí pouze vytvořit soubor ve sdíleném kontejneru namísto vlastního kontejneru aplikace.

Kapitola 5

Testování

Během vývoje byly všechny dílčí části aplikace průběžně testovány programátorem, po dokončení vývoje byly provedeny systémové testy skupinou interních testerů a následně byla aplikace otestována na reálné akci reálnými uživateli.

5.1 Programátorské testování

Developer testing neboli programátorské testování je okamžité ověřování funkčnosti kódu ihned po jeho napsání. Tento druh testování je pro většinu programátorů samozřejmostí, ale přesto je často podceňován. Pokud je toto testování důkladné, může zásadně ušetřit čas a náklady. Chyby mohou být ihned po jejich nalezení opraveny autorem, zatímco při nedůsledném testování chybu objeví až tester a následně ji musí nahlásit programátorovi, čímž narůstá režie a zbytečně se prodlužuje čas vývoje. [35]

Při vývoji aplikace byl kladen velký důraz na programátorské testy, aby zatížení testerů ve fázi systémových testů bylo co nejnižší.

5.2 Systémové testování

Systémové testování testuje hotovou aplikaci jako celek a ověřuje ji z pohledu zákazníka. Při tomto procesu jsou testovány funkční i nefunkční požadavky. Testování většinou probíhá podle předem připravených scénářů, které simulují předpokládané chování uživatelů v praxi. Může být ale testováno i zcela náhodné chování a zadávání vstupů. Testování může probíhat ve více kolech, ve kterých jsou nalezené chyby a nedostatky opravovány a následně opět testovány. [35]

Systémové testování bylo provedeno skupinou interních testerů na připravených vzorových datech pro všechny tři druhy akcí. Po nahlášení chyb proběhla oprava a bylo provedeno druhé kolo testů, které již neodhalilo žádné další chyby.

5.3 Crash reporting

Kromě textového reportu o chybách od testerů je nezbytným nástrojem pro testování crash reporting, což je automatické odesílání logů o pádech aplikace. Služeb pro crash reporting na mobilních zařízeních je celá řada. Mezi nejznámější patří například Crashlytics [32], HockeyApp [28] a další. Všechny tyto služby nabízí podobnou funkčnost. Všechny záznamy o pádech jsou doplněny o velké množství užitečných dat jako přesný čas pádu, četnost pádů, verze aplikace, verze systému nebo i výpis zásobníku volání, ze kterého může programátor ihned vidět, ve kterém místě kódu pád nastal. Kromě využití při testování lze využívat crash reporting i v produkčním prostředí pro zachycení chyb, které nebyly při testování nalezeny. [28]

Pro tuto práci byla využita služba HockeyApp, což je komplexní služba pro testování aplikací. Kromě distribuce testovacích verzí testerům a zmíněného crash reportingu nabízí i pokročilý systém uživatelských práv k jednotlivým verzím nebo statistiky počtu stažení jednotlivých verzí a další.

5.4 API

Vzhledem k tomu, že programátorské testování probíhalo průběžně od samého začátku vývoje, kdy ještě API nebylo implementováno, byla aplikace v této fázi testována proti službě Apiary (viz sekce 4.6). V průběhu systémového testování již byl k dispozici funkční server implementovaný podle definice uvedené v sekci 3.2, proto bylo pro systémové testy využito hotové funkční řešení. Tento server byl dodán vedoucím práce.

5.5 Reálné nasazení

Po dokončení systémového testování a opravení nalezených chyb byla aplikace nasazena na skutečném filmovém festivalu a používána řádově několika stovkami lidí. Podle záznamů z HockeyApp i podle odezvy uživatelů proběhlo nasazení úspěšně a nedošlo k žádným problémům.

Závěr

Na základě analýzy byla navržena a implementována aplikace splňující všechny požadavky definované zadáním i funkční a nefunkční požadavky vytyčené v sekci 1.2.

Aplikace slouží návštěvníkům popisovaných společensko kulturních akcí jako kompletní průvodce a svou funkčností plnohodnotně nahrazuje papírové programy i některé další komunikační kanály, což bylo hlavním cílem této práce.

Implementovaná aplikace funguje univerzálně na všech iOS zařízeních (iPhone, iPod Touch, iPad) a disponuje i zadaným rozšířením pro hodinky Apple Watch. Výsledná grafická podoba aplikace dodaná vedoucím práce vychází z navržených wireframes. Aplikace podle zadání získává data pomocí webových služeb ze serveru skrz API definované na základě předcházející analýzy. Podle této definice byla vedoucím práce zajištěna jeho implementace, jejíž správnost byla ověřena v rámci testování. Výsledek byl otestován skupinou testerů a následně byla funkčnost ověřena i v reálném prostředí. Zadání je tak možno považovat ve všech bodech za splněné.

Budoucí vývoj

Při vývoji aplikace bylo od počátku dbáno na možnou rozšiřitelnost a možnost do aplikce snadno doimplementovat další funkce. Příkladem mohou být například mapy, kde se do budoucna počítá s využitím interaktivních map namísto statických obrázků. V aplikaci se nabízí mnoho dalších možností jako například filmové trailery, zvukové ukázky hudebních interpretů nebo propojení se sociálními sítěmi. V popisovaném modelu univerzální aplikace pro kulturní akce je celkově určitě velký potenciál a spousta prostoru pro mnohá vylepšení a další vývoj, proto věřím, že projekt se bude dále rozvíjet a pokračovat.

Literatura

[1] Alamofire Software Foundation: *AFNetworking* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://github.com/AFNetworking/AFNetworking

[2] Apache Software Foundation: *Apache Subversion* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://subversion.apache.org/

[3] Apiary Ltd.: *Fast-track your API Design* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://apiary.io/how-it-works

[4] Apple, Inc.: *Apple Reinvents the Phone with iPhone* [online]. 2007, [cit. 2015-05-03]. Dostupné z: https://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html

[5] Apple, Inc.: *Apple – iPhone – Features – OS X* [online]. 2008, [cit. 2015-05-03]. Dostupné z: http://web.archive.org/web/20080111051348/http://www.apple.com/iphone/features/index.html#macosx

[6] Apple, Inc.: *Apple Announces iPhone 2.0 Software Beta* [online]. 2008, [cit. 2015-05-03]. Dostupné z: https://www.apple.com/pr/library/2008/03/06Apple-Announces-iPhone-2-0-Software-Beta.html

[7] Apple, Inc.: *Apple Presents iPhone 4* [online]. 2010, [cit. 2015-05-03]. Dostupné z: https://www.apple.com/pr/library/2010/06/07Apple-Presents-iPhone-4.html

[8] Apple, Inc.: *Apple Unveils iOS 7* [online]. 2013, [cit. 2015-05-03]. Dostupné z: https://www.apple.com/pr/library/2013/06/10Apple-Unveils-iOS-7.html

[9] Apple, Inc.: *NSFetchedResultsController Class Reference* [online]. 2013, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/prerelease/ios/documentation/CoreData/Reference/NSFetchedResultsController_Class/index.html

[10] Apple, Inc.: *Apple Releases iOS 8 SDK With Over 4,000 New APIs* [online]. 2014, [cit. 2015-05-03]. Dostupné z: http://www.apple.com/pr/library/2014/06/02Apple-Releases-iOS-8-SDK-With-Over-4-000-New-APIs.html

[11] Apple, Inc.: *Apple Unveils iOS 8, the Biggest Release Since the Launch of the App Store* [online]. 2014, [cit. 2015-05-03]. Dostupné z: http://www.apple.com/pr/library/2014/06/02Apple-Unveils-iOS-8-the-Biggest-Release-Since-the-Launch-of-the-App-Store.html

[12] Apple, Inc.: *Core Data Programming Guide* [online]. 2014, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreData/cdProgrammingGuide.html

[13] Apple, Inc.: *App Extension Programming Guide* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/prerelease/ios/documentation/General/Conceptual/ExtensibilityPG/ExtensionScenarios.html

[14] Apple, Inc.: *App Store Distribution* [online]. 2015, [cit. 2015-03-30]. Dostupné z: https://developer.apple.com/support/appstore/

[15] Apple, Inc.: *Apple Push Notification Service* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

[16] Apple, Inc.: *Apple Watch - Product Images & Info* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://www.apple.com/pr/products/apple-watch/Apple-Watch.html

[17] Apple, Inc.: *Apple Watch and its Elements – The First Look* [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://www.rapidvaluesolutions.com/apple-watch-and-its-elements-the-first-look/

[18] Apple, Inc.: *Apple Watch Programming Guide* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/documentation/General/Conceptual/WatchKitProgrammingGuide/

[19] Apple, Inc.: *Auto Layout Guide* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/AutolayoutPG.pdf

[20] Apple, Inc.: *iOS Developer Library* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/navigation/

[21] Apple, Inc.: *iOS Human Interface Guidelines* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

[22] Apple, Inc.: *iPod Touch* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://www.apple.com/cz/ipod-touch/specs.html

[23] Apple, Inc.: *Tutorial: Storyboards* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/SecondTutorial.html

[24] Apple, Inc.: *Using Design Patterns* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesignPatterns.html

[25] Apple, Inc.: *Watch Technology* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://www.apple.com/watch/technology/

[26] Apple, Inc.: *Xcode - The complete toolset for building great apps* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://developer.apple.com/xcode/features/

[27] Balsamiq Studios, LLC: *Balsamiq* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://balsamiq.com/

[28] BIT STADIUM: *HockeyApp - The platform for your apps.* [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://hockeyapp.net/features/

[29] BRAY, T., Google, Inc.: *The JavaScript Object Notation (JSON) Data Interchange Format* [online]. 2014, [cit. 2015-05-03]. Dostupné z: http://tools.ietf.org/html/rfc7159

[30] BUDELMANN, Jonas, PAYNE, Robert: *Masonry* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://github.com/SnapKit/Masonry

[31] CHRISTENSEN, E., CURBERA, F., MEREDITH, G., WEERAWARANA, S.: *Web Services Description Language (WSDL) 1.1* [online]. 2001, [cit. 2015-05-03]. Dostupné z: http://www.w3.org/TR/wsdl

[32] Crashlytics: *Crashlytics* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://try.crashlytics.com/

[33] EveryiPhone.com: *iPhone Q&A* [online]. 2013, [cit. 2015-05-03]. Dostupné z: http://www.everymac.com/systems/apple/iphone/iphone-faq/iphone-processor-types.html

[34] FULLER, Kyle, THEROX, Orta, GIDDINS, Samuel and The CocoaPods Dev Team: *CocoaPods* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://cocoapods.org/about

[35] HLAVA, T.: *Fáze a úrovně provádění testů* [online]. 2011, [cit. 2015-05-03]. Dostupné z: http://testovanisoftwaru.cz/druhy-typy-a-kategorie-testu/faze-testu/

[36] ŽIKOVSKÝ, P.: *Návrh UI, prototypy* [online]. 2014, [cit. 2015-05-03]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x02-navh_a_prototyping.pdf

[37] JOHNSON, J.: *Mobile First Design: Why It's Great and Why It Sucks* [online]. 2013, [cit. 2015-05-03]. Dostupné z: http://designshack.net/articles/css/mobilefirst/

[38] JONES, B.: *Apple Retina Display* [online]. 2010, [cit. 2015-05-03]. Dostupné z: http://prometheus.med.utah.edu/~bwjones/2010/06/apple-retina-display/

[39] KOFMAN, M.: *How to Create a One-Size-Fits-All App for iOS: iOS 8 Adaptive UI* [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://www.applicoinc.com/blog/create-one-size-fits-app-ios-ios-8s-adaptive-ui/

[40] KUHN, M.: *A summary of the international standard date and time notation* [online]. 2004, [cit. 2015-05-03]. Dostupné z: http://www.cl.cam.ac.uk/~mgk25/iso-time.html

[41] MASSÉ, M.: *REST API Design Rulebook*. First edition, O'Reilly Media, Inc., 2012, ISBN 978-1-449-31050-9.

[42] MITRA, N., LAFON, E. Y., W3C: *SOAP Version 1.2 Part 0: Primer (Second Edition)* [online]. 2007, [cit. 2015-05-03]. Dostupné z: http://www.w3.org/TR/soap12-part0/

[43] MURASHEV, N.: *UISplitViewController* [online]. 2014, [cit. 2015-05-03]. Dostupné z: http://nshipster.com/uisplitviewcontroller/

[44] Oxford University Press: *WYSIWYG* [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://www.oxforddictionaries.com/definition/english/WYSIWYG

[45] PixelCut: *The Ultimate Guide To iPhone Resolutions* [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions

[46] Realm: *Realm is a mobile database* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://realm.io/

[47] ROHLIN, A.: *The Definition of Web Service EndPoint* [online]. [cit. 2015-05-03]. Dostupné z: http://www.ehow.com/info_12212371_definition-service-endpoint.html

[48] SHAFRANOVICH, Y., SolidMatrix Technologies, Inc.: *Common Format and MIME Type for Comma-Separated Values (CSV) Files* [online]. 2005, [cit. 2015-05-03]. Dostupné z: http://tools.ietf.org/html/rfc4180

[49] SHANKLIN, W.: *2014 iPad Comparison Guide* [online]. 2014, [cit. 2015-05-03]. Dostupné z: http://www.gizmag.com/ipad-comparison-2014-2015/34443/

[50] SINGH, A.: *A Brief History of Mac OS X* [online]. 2003, [cit. 2015-05-03]. Dostupné z: http://osxbook.com/book/bonus/ancient/whatismacosx/history.html

[51] Software Freedom Conservancy: *Git About* [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://git-scm.com/about

[52] SQLite Consortium: *SQLite* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://www.sqlite.org/

[53] The Internet Society: *Hypertext Transfer Protocol – HTTP/1.1* [online]. 1999, [cit. 2015-05-03]. Dostupné z: https://tools.ietf.org/html/rfc2616

[54] The Open Group: *The Open Brand - Register of Certified Products* [online]. 2014, [cit. 2015-05-03]. Dostupné z: http://www.opengroup.org/openbrand/register/brand3607.htm

[55] URI Planning Interest Group, W3C/IETF: *URIs, URLs, and URNs: Clarifications and Recommendations 1.0* [online]. 2001, [cit. 2015-05-03]. Dostupné z: http://www.w3.org/TR/uri-clarification/

[56] W3C: *HTML 4.01 Specification* [online]. 1991, [cit. 2015-05-03]. Dostupné z: http://www.w3.org/TR/html401/

[57] W3C: *Extensible Markup Language (XML)* [online]. 2015, [cit. 2015-05-03]. Dostupné z: http://www.w3.org/XML/

[58] WATTERS, Blake and the RestKit team: *RestKit* [online]. 2015, [cit. 2015-05-03]. Dostupné z: https://github.com/RestKit/RestKit

[59] WINER, D.: *XML-RPC Specification* [online]. 1999, [cit. 2015-05-03]. Dostupné z: http://xmlrpc.scripting.com/spec.html

[60] ZENDULKA, J.: *Projektování programových systémů - Požadavky a jejich specifikace* [online]. 2015, [cit. 2015-05-03]. Dostupné z: `http://www.fit.vutbr.cz/study/courses/PPS/public/pdf/5_2.pdf`

Příloha A

Seznam použitých zkratek

API Application programming interface

HIG Human interface guidelines

HTML HyperText markup language

HTTP Hypertext transfer protocol

JSON JavaScript object notation

MVC Model-view-controller

REST Representational state transfer

UI User interface (uživatelské rozhraní)

XML Extensible markup language

LTE Long term evolution

SDK Software development kit

APNs Apple push notification service

Příloha **B**

Obsah přiloženého CD

- `Implementace` ..... adresář se zdrojovými soubory implementace
  - `Application` ..... mateřská aplikace
  - `WatchKit App` ..... aplikace pro hodinky
  - `WatchKit Extension` ..... rozšíření mateřské aplikace
  - `Podfile` ..... definice knihoven třetích stran
- `Text` ..... adresář s textem práce
  - `tex` ..... zdrojová forma práce ve formátu LaTeX
  - `DP_Misar_Jan_2015.pdf` ..... text práce ve formátu PDF
- `Wireframes` ..... adresář s obrázky navržených wireframes