VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

TWITTER CLIENT PRO APPLE TV

TWITTER CLIENT FOR APPLE TV

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAREK ZVARA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN HRUBÝ, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19878/2016/xzvara00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Zvara Marek**
Obor: Informační technologie
Téma: **Twitter client pro Apple TV**
Twitter Client for Apple TV
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte programování aplikací pro iOS a tvOS. Prostudujte přístupové rozhraní k systému Twitter.
2. Navrhněte aplikaci pro tvOS poskytující funkcionalitu prohlížení uživatelského obsahu Twitteru včetně základních reakcí (Like, RT).
3. Aplikaci implementujte.
4. Testujte aplikaci v emulátoru i na reálném zařízení.

Literatura:

- Dokumentace knihoven pro iOS a tvOS.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hrubý Martin, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto práca popisuje a analyzuje platformu tvOS a porovnáva jej výhody a nevýhody voči platforme iOS. Okrem toho ponúka prehľad aktuálneho stavu Twitter API. Na základe týchto poznatkov ponúka popis návrhu, úspešne zrealizovanej implementácie a testovania aplikácie. Twitter klient pracuje so základnými funkcionalitami ako prezeranie časovej osi, retweet, like/unlike tweetu či reakcia v diskusii k danému tweetu.

Abstract

This paper describes and analyses the tvOS platform and compares its advantages and disadvantages to the iOS platform. In addition, it offers an overview of the current state of the Twitter API. Based on this knowledge, it provides a description of the design, successful implementation and testing of the application. Twitter client works with basic features such as scrolling the timeline, retweet, like / unlike tweet, or responding to a tweet discussion.

Kľúčové slová

Twitter, klient, Apple TV, swift, MVC, paralelne procesy, twitter API, REST, retweet, OAuth, tvOS, Apple

Keywords

Twitter, client, Apple TV, swift, MVC, multithreading, twitter API, REST, retweet, OAuth, tvOS, Apple

Citácia

ZVARA, Marek. *Twitter client pro Apple TV*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Hrubý Martin.

Twitter client pro Apple TV

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Martina Hrubého, Ph.D.. Ďalšie informácie mi poskytla verejná dokumentácia pre vývojárov od Applu a Twittru. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Marek Zvara
24. júla 2017

Poďakovanie

Ďakujem za odbornú pomoc vedúcemu bakalárskej práce pánovi Ing. Martinovi Hrubému, Ph.D., za jeho cenné rady a konzultácie pri písaní tejto práce.

Ďalej ďakujem svojim rodičom, starým rodičom a všetkým členom mojej rodiny, ktorí pri mne stáli počas štúdia a podporovali ma, vďaka čomu som mohol napísať túto bakalársku prácu.

Ďakujem svojmu dlhoročnému kamarátovi Jurajovi Hallonovi, ktorý mi umožnil publikovať článok o mojej práci na svojom webovom magazíne www.svetapple.sk, a tak prispel k verejnému testovaniu aplikácie. Taktiež ďakujem šéfredaktorovi Danielovi Pražákovi z online magazínu www.letemsvetemapplem.eu, ktorý mi taktiež umožnil publikovať článok o práci, a tak tiež prispel ku skvalitneniu verejného testovania aplikácie.

Ďalej ďakujem všetkým ďalším kamarátom za cenné a užitočné rady pri písaní práce a za ich vzácny čas pomôcť mi s riešením niektorých problémov.

Nakoniec ďakujem všetkým anonymným účastníkom verejného testovania aplikácie, ktorí tak prispeli ku skvalitneniu aplikácie.

Obsah

Kapitola 1

Úvod

V Českej republike sa Twitter radí medzi menej populárnejšie sociálne siete, keďže počet aktívnych užívateľov je len okolo 300 tisíc [63]. Ak ho ale berieme v celosvetovom merítku, ide o jednu z najrozšírenejších sociálnych sietí na svete, ktorá stále rastie. Svedčí o tom aj jeho 328 miliónov aktívnych užívateľov [38]. Práve jeho koncept mikroblogov, ktorý umožňuje užívateľom čítať a posielať správy ostatných užívateľov, a tak vysoký počet aktívnych užívateľov ho radia medzi zdroj zaujímavých a cenných informácii a dát. Vďaka jeho otvorenému prístupu a možnosti využiť Twitter API sa ponúkajú takmer neobmedzené alternatívy, ako s týmito dátami pracovať a šíriť ich ďalej. Okrem webového rozhrania pre prístup sa práve vďaka tejto otvorenosti ponúkajú príležitosti, ako užívateľovi poskytnúť obsah aj na iných platformách.

Jednou z týchto platform je set-top box Apple TV od firmy Apple. Ten umožňuje prenos obsahu (video, fotky, hudba) do televízie pomocou WiFi či ethernet kábla a taktiež umožňuje inštalovať aplikácie tretích strán.

Cieľom tejto práce je vytvoriť Twitter klienta ako aplikáciu pre Apple TV. Tá umožní užívateľovi prezerať obsah dostupný na Twitteri a vykonávať základné akcie Twitteru ako Like či Retweet.

V druhej kapitole si popíšeme východiská práce, a to samotné spôsoby programovania pre Apple TV a štruktúru Twitter API. V tretej kapitole sa budeme zaoberať samotným návrhom aplikácie, ako jednotlivé operácie budú prebiehať a ako bude vyzerať celkový kontext aplikácie. V štvrtej kapitole si popíšeme implementačné detaily a v piatej kapitole spôsoby testovania a výsledky testovania. V závere je zhrnuté hodnotenie celej aplikácie a možnosti rozšírenia.

Kapitola 2

Východiská práce

V tejto časti sa zameriame na popis 2 rozličných prístupov v programovaní pre platformu tvOS. Tieto koncepty porovnáme a vyvodíme ich výhody a nevýhody. Ďalej sa zameriame na porovnanie dostupných frameworkov v tvOS v porovnaní s iOS. Popíšeme si, aké problémy vznikajú absenciou niektorých z nich a aké výhody ponúkajú nové frameworky pre tvOS. Nakoniec si popíšeme možnosti ovládania Apple TV a aktuálny stav Twitter API.

2.1 Programátorské koncepty pre tvOS

Operačný systém tvOS pre Apple TV vychádza zo systému iOS a je na ňom kompletne založený. Napriek tomu je ale v istých smeroch odlišný. Neobsahuje, prípadne úplne nepodporuje niektoré frameworky z iOS systému a naopak, má podporu frameworkov, ktoré nenájdeme u iOS systému [22].

Pri tvorbe nových aplikácii pre tvOS sa dá využiť táto príbuznosť s iOS platformou. Novú aplikáciu je možné postaviť na existujúcom kóde iOS verzie. tvOS má, tak ako aj iOS, podporu vývoja v jazykoch Swift a Objective-C. Okrem týchto dvoch jazykov ponúka tvOS možnosť vývoja v JavaScripte pre tvorbu takzvaných klient-server aplikácii (viď kapitola 2.1.2).

Rovnako ako pri iOS, tak aj pri vývoji tvOS aplikácii je možné použiť vývojové prostredie XCode od firmy Apple. Pre vývoj (nutnosť iba pri podpore špecifických vlastností aplikácie [16]) a distribúciu aplikácie je potrebné mať vygenerovaný špecifický Apple TV provisioning profile, ktorým sa daná aplikácia podpíše.

iOS a tvOS aplikácie sú odlišné entity. To znamená, že ak máme spoločný kód pre iOS a tvOS platformu, tak pri distribúcii sa negeneruje jeden spoločný binárny kód pre obe platformy [4].

Na rozdiel od iOS, tvOS ponúka 2 možnosti vývoja aplikácii:

1. Tradičné aplikácie
2. Klient- server aplikácie

Ich výhody a nevýhody si lepšie popíšeme v nasledujúcich podkapitolách.

2.1.1 Tradičné aplikácie

Proces tvorby týchto aplikácii je totožný ako pri tvorbe aplikácii pre iOS. To znamená, že je možné využívať bežné prostriedky ako UIStoryboardy, UIKit, Auto layout, a pod. Tento

prístup umožňuje vytvoriť hry, utility, aplikácie pracujúce s médiami či iné typy aplikácii a to rovnakým postupom a technikami ako pri tvorbe iOS aplikácii. Takto vytvorené aplikácie môžu byť použité ako na tvOS platforme, tak na iOS platforme.

2.1.2 Klient – server aplikácie

tvOS ponúka nový spôsob tvorby aplikácii, a to pre aplikácie, ktoré sú založené na koncepte klient – server. Ich hlavným zmyslom je streamovať média, používať webové technológie ako *HTTPS, XMLHTTPRequest, DOM či JavaScript*. Pri tvorbe tohoto druhu aplikácii sa používajú 3 základné piliere:

- TVML [3] – ide o typ XML jazyka a znamená „Television Markup Language“
- TVMLJS [21] – je množina JavaScriptového API, ktorá umožňuje špecifikovať chovanie a zobrazovanie dát do rozhrania špecifikovaného pomocou TVML
- TVMLKit [20] – ide o framework špecifický len pre tvOS, ktorý premosťuje natívny kód s JavaScriptovým kódom v užívateľskom rozhraní.

Koncept je postavený na tom, že sa v JavaScriptovom súbore pomocou TVMLJS špecifikuje inicializačné chovanie pri spustení aplikácie. Následne sa pomocou TVMLKitu načíta tento JavaScriptový súbor zo serveru, ktorý špecifikuje logiku pre prácu s dátami a načítava TVML dokumenty. Tie špecifikujú vzhľad užívateľského rozhrania a pomocou TVMLKitu sa následne zobrazujú na obrazovke Apple TV. JavaScriptové súbory a TVML súbory sa nachádzajú na samostatnom serveri a nie sú súčasťou natívneho kódu aplikácie. Vizuálny prehľad fungovania tohto konceptu zobrazuje príloha A.

Pre tvorbu TVML dokumentov Apple ponúka množstvo šablón [5]. Každá šablóna ponúka jedinečné zobrazenie informácii na celú obrazovku. Jednotlivé šablóny sa dajú upravovať pridávaním alebo mazaním elementov.

2.1.3 Porovnanie konceptov

TVML koncept je vhodný, keď je potrebné vytvoriť streamovaciu aplikáciu, kde je minimálna užívateľská interakcia, užívateľ bude skôr pasívne sledovať či počúvať obsah. Využíva šablóny, takže užívateľ je do istej miery zvyknutý na prostredie z iných aplikácii, ktoré tiež využívajú tento prístup. Použitie tohto konceptu je tiež výhodné, keď máme bežiaci server, kde sú dáta rozdelené predvídateľným spôsobom a navigácia bude intuitívna a známa. Vďaka tomu, že JavaScriptové a TVML súbory sú uložené na serveri a nie sú súčasťou aplikácie, je možné dynamicky upravovať vzhľad a obsah aplikácie bez nutnosti aktualizácie.

Tradičný prístup ponúka možnosť ponúknuť užívateľovi plne pohlcujúci zážitok vlastne definovaným užívateľským rozhraním. Preto je vhodný pre aplikácie, kde sa od užívateľa vyžaduje väčšia miera interakcie, ako napríklad pri tvorbe hier. Užívateľovi nie je nalinkované jasne dané rozhranie pomocou šablóny, takže aplikácia môže byť pre neho zaujímavejšia a viacej interaktívna.

Je nutné podotknúť, že v prípade potreby je možné tieto dva prístupy kombinovať a vytvoriť určitú hybridnú aplikáciu, keďže TVMLKit je framework, ktorý dokáže premostiť natívny kód aplikácie s kódom JavaScriptu.

2.2 Porovnanie dostupných frameworkov

V tejto podkapitole si bližšie rozoberieme dopad absencie niektorých frameworkov na programovanie pre platformu tvOS. Taktiež si popíšeme prínos nových dvoch komponent `TVServices` a `Focus Enginu`.

Aj keď je tvOS postavený na základoch iOS systému, má isté obmedzenia. David Olesch, viceprezident vývoja pre Jackrabbit Mobile, vytvoril porovnanie frameworkov (viď príloha B) dostupných na iOS a tvOS platformách [37]. Ide ale zrejme o porovnanie staršej verzie tvOS, pretože v porovnaní s oficiálnou dokumentáciou k tvOS 10.0 API boli niektoré frameworky pridané [22].

2.2.1 Absencia WebView

Najväčším obmedzením je, že tvOS nepodporuje WebView. WebView sú používané v mnohých iOS aplikáciách a pri prenose na tvOS či pri tvorbe novej aplikácie táto absencia spôsobuje mnohé komplikácie a aplikácia musí byť prerobená.

Pre vývoj multiplatformných aplikácii je použitie WebView veľmi výhodné. Určitý typ obsahu môže byť takto zobrazený v aplikácii nezávisle na platforme, čo zjednodušuje vývoj. Obsah je ľahko zmeniteľný aj po vydaní aplikácie.

Kvôli absencii tohto prvku je prezeranie webových stránok vo všeobecnosti nemožné. Aplikácie, ktoré využívajú autentifikáciu tretích strán pomocou OAuth protokolu [36], musia použiť iné spôsoby prihlasovania. Tie zvyknú byť ale menej užívateľsky prívetivé, napríklad využitím pin kódu a ďalšieho zariadenia s podporou prezerania webu.

2.2.2 Absencia EventKitu, AdressBook či iMessage

Tieto frameworky sú tiež nedostupné v tvOS. Takže aplikácie nemôžu využiť dáta uložené v iCloude ako kontakty, či udalosti kalendára. Framework MessagesUI je tiež nedostupný, čo berie možnosť posielať alebo prijímať iMessage správy.

2.2.3 TVServices

Ide o úplne nový framework, ktorý je možné nájsť len v tvOS (viď dokumentáciu [23]). Cieľom tohto frameworku je popísať obsah aplikácie, ktorý bude zobrazený v takzvanom *„Top Shelf“*. Užívateľ si v zozname aplikácii môže usporiadať aplikácie do riadkov, podľa svojho uváženia. Ak je aplikácia umiestnená v najvrchnejšom riadku, je považovaná za dôležitú pre užívateľa a môže poskytnúť špeciálny obsah v priestore nad riadkom aplikácii. Tento priestor sa volá *„Top Shelf“* (viď príloha C). Môže obsahovať statický obrázok, náhľad obsahu aplikácie alebo dať užívateľovi možnosť skočiť priamo do určitej časti aplikácie [19].

2.2.4 Focus engine

tvOS prináša nový koncept interakcie užívateľa s aplikáciou. Užívateľ nekomunikuje s užívateľským rozhraním pomocou prstov ako u iOS systému, ale pomocou diaľkového ovládača.

Preto tvOS prináša systém zvaný *„Focus Engine“* [6]. Podstatou tohto systému je, že vždy je zaostrený len na jeden element v danom čase a nikdy nie na viaceré. Užívateľ si je toho vedomý vďaka vizuálnemu odlíšeniu zaostreného prvku, väčšinou zvýraznením do popredia a zmenou farby. Tento systém následne automaticky rozhodne, ktorý prvok bude zaostrený, a to na základe dotykov užívateľa na sklenenej dotykovej ploche diaľkového

ovládača. Keď užívateľ potiahne doprava, tak sa nájde najbližší prvok na pravej strane od aktuálne zaostreného prvku a zaostrí sa, ak je to možné.

Spolu s týmto novým konceptom prináša tvOS aj API, ktorým je možné zisťovať napríklad, či bol prvok zaostrený, či môže byť vôbec zaostrený, manuálne vyvolať aktualizovanie celého ostriaceho systému alebo upravovať vzhľad zaostreného prvku a pod. [9]

2.3 Obmedzenie veľkosti perzistentných dát

tvOS neponúka možnosť ukladania perzistentných dát. Apple TV ponúka úložný priestor – 32 GB alebo 64 GB, čo je zrovnateľné s veľkosťou úložiska napr. iPhonu SE, ktorý má 16 GB alebo 64 GB. Problém vzniká ale napríklad pri pozeraní Full HD filmu, ktorý môže mať 4 – 6 GB, takže Apple TV si rezervuje čo najväčší možný priestor pre jeho prehranie [4]. Z toho vyplýva, že dáta sú 100 % dostupné len pri behu aplikácie. Ak aplikácia beží na pozadí má určitú mieru podpory cachovania, no v momente keď systém potrebuje priestor, tak dané dáta uvoľní.

Na rozdiel od iOS systému v tvOS nie je možné zapisovať do NSDocumentDirectory. Jediný možný priestor, kam sa dajú dočasne zapísať dáta je NSCachesDirectory a NSTemporaryDirectory [28].

V prípade, že je potrebné uložiť perzistentné dáta menšie ako 1MB je možné použiť iCloud KVS a v prípade väčšieho množstva dát zas CloudKit [10].

Maximálna povolená veľkosť aplikácie pre tvOS je momentálne 4 GB. Veľkosť perzistentného úložiska v NSUserDefaults je 500 KB [4].

2.4 Ovládanie Apple TV

Komunikácia s užívateľským rozhraním prebieha pomocou špeciálneho diaľkového ovládača. Ten obsahuje 3 základné prvky, ktoré sa dajú využiť pri vývoji aplikácie pre tvOS. Sú to:

1. Sklenená dotyková plocha – nachádza sa na vrchu ovládača a užívateľ vďaka nej môže zadávať obdobné gestá, na ktoré je zvyknutý z iOS systému
2. Mikrofón – ktorý slúži na komunikáciu so Siri (jej dostupnosť je obmedzená na určitú podmnožinu krajín [2])
3. Gyroskop a pohybový senzor

Vďaka sklenenej dotykovej ploche sa dajú využiť štandardné API funkcie pre detekciu gest, ako je tomu pri iOS systéme. Napríklad potiahnutie doľava, doprava, dvojklik a podobne. Okrem toho ale tvOS ešte ponúka špeciálne nové API s metódami ako *pressesBegan(), pressesEnded(), pressesChanged() alebo pressesCanceled()*, ktorými je možné detekovať a kontrolovať udalosti pri stlačení niektorého z tlačidiel ovládača [7]. Treba ale brať na vedomie, že tieto tlačidlá majú Applom preddefinované chovanie, ktoré sa od nich očakáva v istých situáciách [15]. V prípade, že toto chovanie nie je zabezpečené, tak aplikácia nemusí prejsť schvaľovacím procesom pri vydávaní.

2.5 Twitter API

Pre sieťovú komunikáciu s Twitterom máme k dispozícii API rozhranie [46]. Twitter nám ponúka viacero typov API a pri každom záleží na okolnostiach a potrebách danej aplikácie, ktoré je vhodné použiť [60]. Sú to:

1. REST API
2. Streaming API
3. Webhook API
4. Ads API
5. MoPub
6. Fabric

REST API[55] je vhodné pre čítanie a zapisovanie Twitter dát. To znamená vytvoriť nový tweet, čítať užívateľov profil, vytvoriť retweet či označiť tweet ako obľúbený a mnoho ďalšieho [54]. Užívateľ sa musí vždy identifikovať voči aplikácii vytvorenej v developerskej konzole Twitteru [59] pomocou OAuth protokolu. Odpovede prichádzajú vo formáte JSON.

Streaming API[56] je zas vhodné pre monitorovanie a spracovanie Tweetov v reálnom čase. Vyžaduje otvorené HTTP spojenie pre prijímanie toku dát. Je vhodné na sledovanie špecifického používateľa, témy alebo pre dolovanie dát.

Webhook API[62] slúži na prístup k dátam účtu v reálnom čase. Vyžaduje nakonfigurovanie cieľovej URL adresy a poslanie výzvy na zasielanie odpovede. Následne Twitter začne posielať POST správy priamo na zadanú URL adresu, keď nastane relevantná udalosť. V súčasnosti Twitter podporuje iba jeden webhook API s názvom Account Activity API [39], ktorý je v Beta verzii.

Ads API[58] je ponúkané pre partnerov Twitteru pre prácu s reklamnou platformou. Vďaka tomuto API je možné vytvoriť vlastné reklamné riešenia, čiže nástroje pre manažovanie a zadávanie reklamných kampaní.

MoPub[35] je samostatná platforma určená pre mobilné zariadenia (iOS, Android a mobilný web). Slúži na spravovanie reklamy, zvyšovanie príjmov z reklamy, získavanie štatistík a podobne.

Poslednou kategóriou je **Fabric**[26], ktorý ponúka viacero nástrojov pre rôzne platformy - Android, Apple, Unity či Web. Okrem nástrojov pre distribúciu, získavanie analýz z testovania či zhromažďovania pádov aplikácii ponúka aj *TwitterKit*. Ten zapuzdruje všetky potrebné operácie potrebné na komunikáciu s Twitterom. Tento Kit má zatiaľ len podporu pre Android, Unity a iOS. Pre tvOS tento KIT aktuálne nemá podporu [27].

2.5.1 Rest API a jeho Rate limit

Rest API pracuje s atribútom zvaným *„Rate limit“* . To znamená, že počet správ typu GET a POST nemôže prekročiť istý limit. Pri tomto limite Twitter pracuje s atribútom *„window“*. Do neho sa zahrňujú všetky dotazy a jeho dĺžka je 15 minút. Štandardne je nastavené, že na jedno *„window“* môže byť maximálne 15 dotazov. Veľa dotazov má ale definovanú svoju hodnotu [53], napríklad pre dotaz na stiahnutie užívateľovej časovej osi s tweetmi je limit 900 dotazov na jedno *„window“*.

V prípade prekročenia tohto limitu sa ako odpoveď príjme správa s chybovým kódom a správou, že daný limit bol prekročený a je potrebné počkať, kým nevyprší čas aktuálnemu *window*. Ak aplikácia moc často prekračuje daný limit, tak je možné, že bude uvedená na blacklist a komunikácia takejto aplikácie s Twitter API je následne nemožná [40].

2.5.2 Základné objekty Twitteru

Twitter API pracuje so 4 základnými objektami, sú to:

- Tweet - objekt popisujúci daný tweet status. Hlavným atribútom je 64 bit identifikátor, ktorý definuje unikátnosť daného tweetu. Okrem toho má množstvo ďalších atribútov, ako dátum vytvorenia, meno autora, príznaky či je obľúbený alebo retweetovaný a pod. [57]

- User - Tento objekt je tiež definovaný svojím 64 bit identifikátorom. Obsahuje dáta ako meno užívateľa, URL adresy na jeho profil, profilovú fotku, banerovú fotku či farby jeho profilu [61].

- Entity - obsahuje metadáta a ďalšie kontextové informácie o obsahu publikovanom na Twitteri. Tento objekt je vždy viazaný na daný tweet, ktorý popisuje. Môže obsahovať zoznam použitých hashtagov, url adries, označenia iných užívateľov, či zoznam pripojených médii, ako fotka, video, či GIF obrázok [44].

- Place - popisuje miesto, kde bol daný tweet vytvorený. Obsahuje informácie ako názov mesta, geografické súradnice či názov štátu [51].

2.5.3 Autentifikácia užívateľa

Pre autentifikáciu prístupu k dátam Twitter používa OAuth protokol [47]. Ide o neproprietárny protokol, ktorý definuje štandard pre bezpečnú a jednoduchú autentifikáciu užívateľa na webe či mobilnej alebo desktopovej aplikácii [36].

Bezpečnosť spočíva v tom, že užívateľ neposkytuje svoje prihlasovacie údaje aplikáciám tretích strán. Vďaka tomu, že ide o štandard, existuje mnoho príkladov kódu a knižníc implementujúcich tento protokol kompatibilných s Twitterom.

Twitter rozlišuje dva typy autentifikácie:

- User authentication - Ide o najviac používanú formu autentifikácie na Twitteri. Využíva OAuth 1.0A [30]. Posiela sa dotaz s podpisom, ktorý identifikuje identitu aplikácie a práva poskytnuté koncovým používateľom v podobe takzvaného *„access tokenu“*.

- Application-only authentication - Ide o typ autentifikácie, kedy nie je potrebný kontext užívateľa. Je založená na OAuth 2.0 [31] a *„access tokene“* užívateľa.

Každý používateľ musí dostať vlastný takzvaný *„access tokenu“*, ktorý ho charakterizuje a špecifikuje práva, ktoré poskytuje aplikácii. Na jeho získanie Twitter ponúka viacero možností [48]:

- Sign in with Twitter - pre webové platformy a mobilné stránky. Na danú stránku sa umiestni tlačidlo pre prihlásenie sa do Twitteru a následne je pomocou API dotazov užívateľ presmerovaný na stránku Twitteru pre prihlásenie a poskytnutie potrebných práv.

- 3-legged OAuth - tento prístup je zhodný s prvým prístupom. Užívateľ je presmerovaný na stránku Twitteru, ale pomocou iného API dotazu. Rozdiel je v tom, že užívateľ bude vždy vyzvaný k poskytnutiu práv pre danú aplikáciu, aj keď to už raz urobil.
- PIN-based authorization - je vhodné pre aplikácie, ktoré nemajú prístup k webovému prehliadaču. Je vygenerovaná URL adresa, ktorú je potrebné zadať na zariadení s prístupom k webovému prehliadaču. Užívateľ následne povolí práva a dostane 7 miestny pin kód. Ten sa zadá naspäť do aplikácie a pošle sa overovací dotaz, čím sa získa *„access token“*.
- Tokens from dev.twitter.com - Twitter poskytne pri vytvorení Twitter aplikácie autorovi jeho vlastný access token, ktorý môže použiť na testovanie.
- xAuth - Užívateľ priamo zadáva svoje prihlasovacie meno a heslo. Táto forma musí byť povolená Twitterom a dostatočne odôvodnená.
- OAuth Echo - Autorizácia je stále vyžadovaná, ale prebieha na tretej strane.
- Application-only authentication - Autentizácia založená na kontexte samotnej aplikácie, kde sa nevyžaduje kontext užívateľa. Access token sa získa poslaním API dotazov špecifických pre OAuth 2.0

Politika Twitteru vyžaduje, aby každý, kto chce komunikovať s jeho API mal založený profil aplikácie v developerskej konzole Twitteru [59]. Tej je následne pridelený *Consumer Key (API Key)*, čo je verejný kľúč a *Consumer Secret (API Secret)*, čo je zas privátny kľúč. Tieto dva kľúče slúžia následne k vygenerovaniu podpisu pre každú požiadavku a identifikujú tak danú aplikáciu. Okrem toho má aplikácia v profile developerskej konzoly nastavené práva, ktoré bude vyžadovať od užívateľa pri prihlasovaní. Môže si vyžiadať 3 typy práv [41]:

- Len na čítanie - ide o čítanie užívateľských dát v profile, tweety a pod.
- Čítanie a zápis - ide o metódy spojené s čítaním dát a generovaním nových dát, napríklad vytvorenie nového tweetu.
- Čítanie, Zápis a prístup k Priamym správam

Každá zmena nastavení práv v tomto profile spôsobí vygenerovanie nového verejného a privátneho kľúča aplikácie.

Kapitola 3

Návrh aplikácie

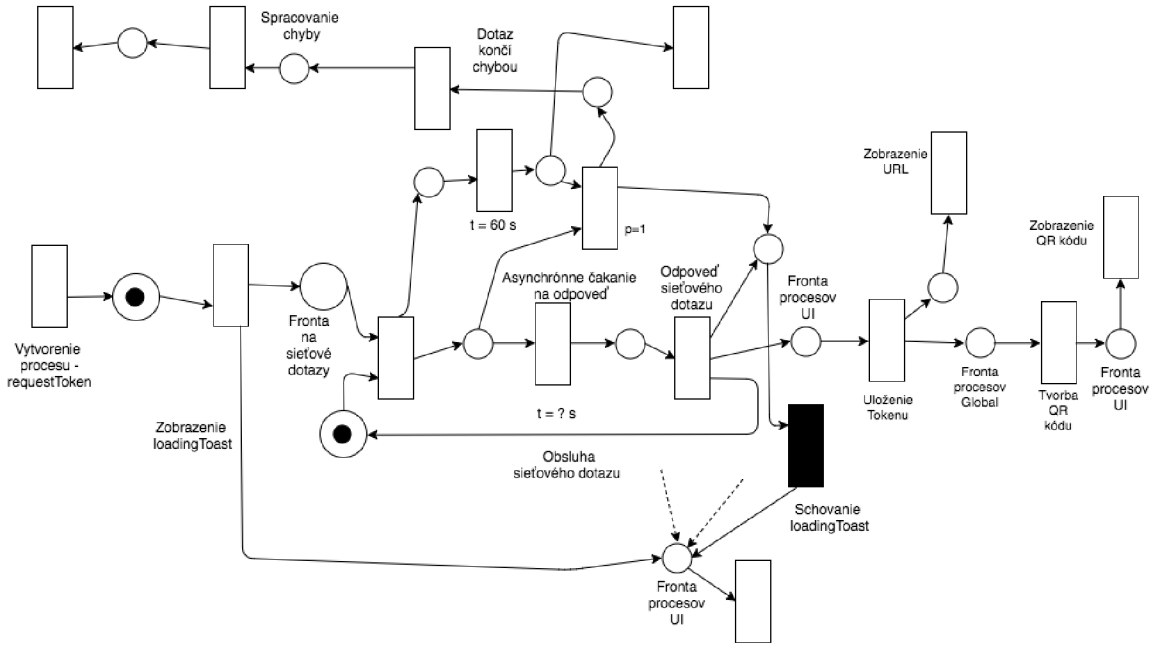
V tejto kapitole sa budeme venovať návrhu samotnej aplikácie. Predstavíme si jej funkcionality ako procesy namodelované pomocou petriho sietí. Ďalej si popíšeme schému UI aplikácie a nakoniec celého kontextu chodu aplikácie.

3.1 Funkcie aplikácie

Po rozobraní zadania sa odvodili požiadavky, ktoré by mala aplikácia spĺňať. Cieľom bolo vytvoriť aplikáciu, ktorá užívateľovi umožní prihlásiť sa do svojho profilu a následne prezerať užívateľský obsah Twitteru vrátane základných reakcií ako Like a Retweet.

3.1.1 Prihlásenie sa do profilu

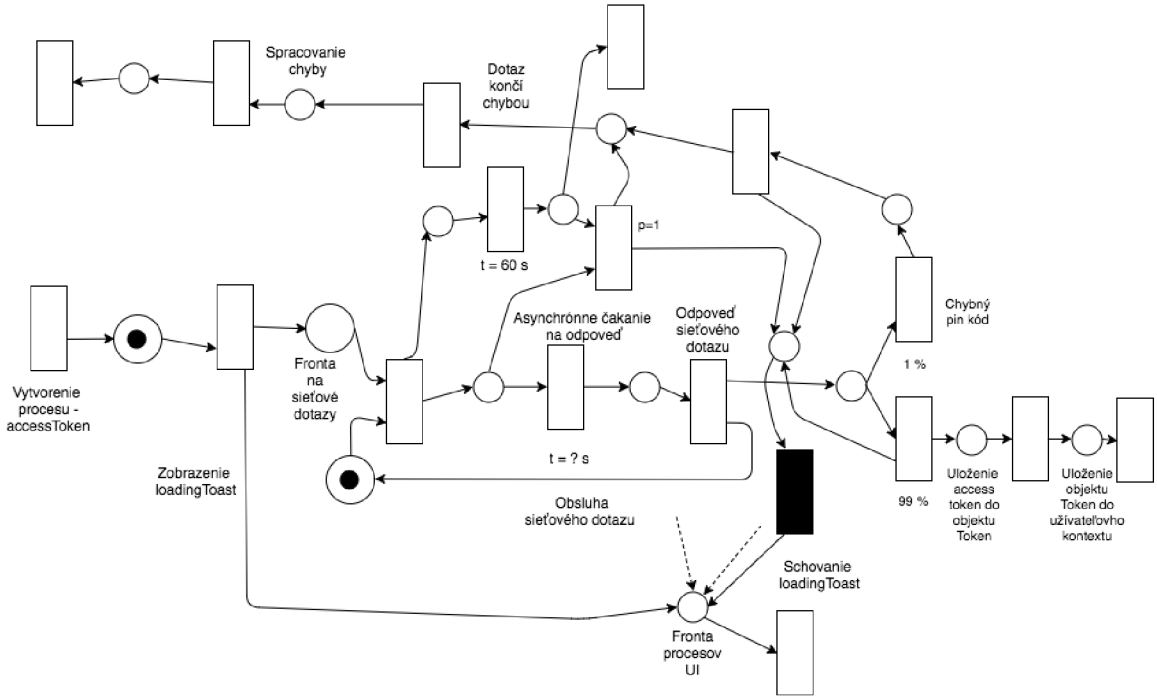
Prvou z funkcionalít bude teda možnosť prihlásenia sa do profilu užívateľa. Tento proces zobrazuje petriho sieť na obrázku č. 3.1.



Obr. 3.1: Získanie request tokenu

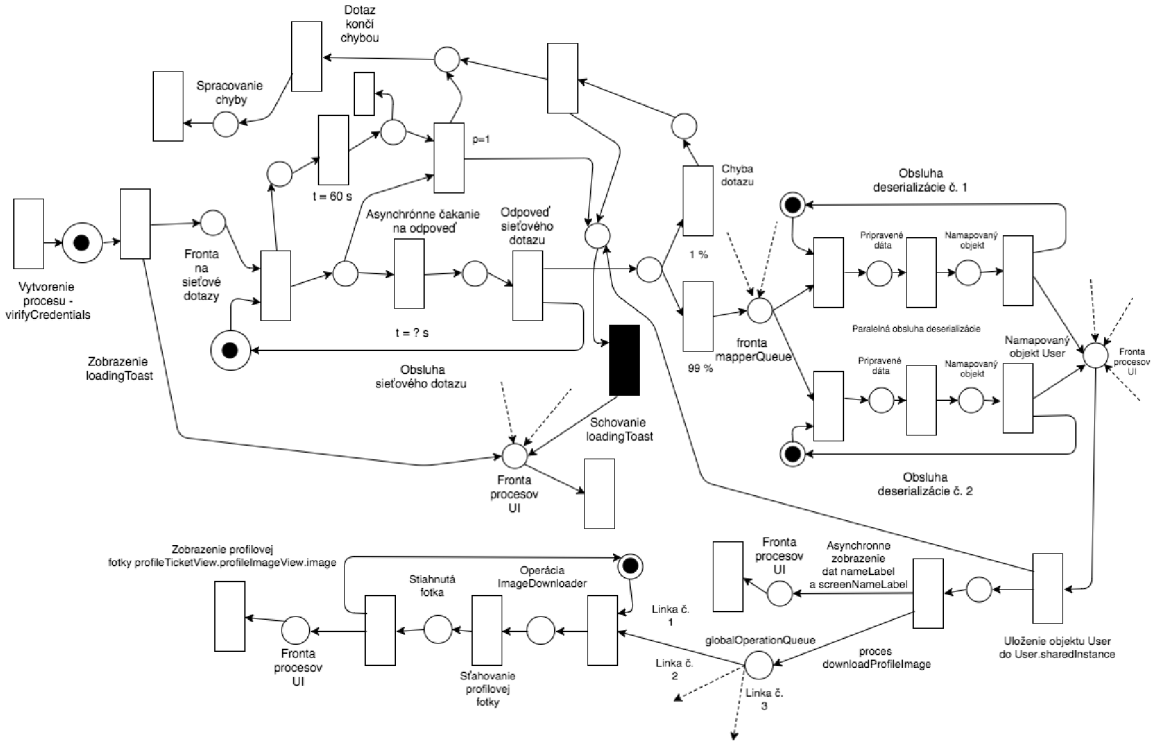
Aplikácia si zažiada o takzvaný *request token* pomocou Twitter API. Pomocou neho sa vygeneruje špeciálne URL, ktoré je potrebné navštíviť na inom zariadení, kde je webový prehliadač. Aby bol tento prístup viacej priateľsky ku koncovému užívateľovi, tak sa URL adresa zakóduje ešte do QR kódu, ktorý je jednoduchší na oskenovanie.

Následne, keď užívateľ potvrdí práva pre aplikáciu a zadá vygenerovaný PIN kód späť do aplikácie, tak sa príjme privátny a verejný *access token*. Spolu s privátny a verejným *consumer key* budú slúžiť na vygenerovanie podpisu daného sieťového dotazu. Tieto dáta sa uložia pre uchovanie kontextu daného užívateľa. Tento proces popisuje obrázok č. 3.2.



Obr. 3.2: Získanie access tokenu

Potom sa vygeneruje dotaz `verifyCredentials` na obdržanie informácii o užívateľovi: meno užívateľa, jeho prezývka a link na profilovú fotku. Tieto dáta sa tiež uložia pre uchovanie kontextu daného užívateľa. Po obdržaní týchto dát sa vygeneruje dotaz na stiahnutie užívateľovej profilovej fotky. Tento proces zobrazuje obrázok č. 3.3.

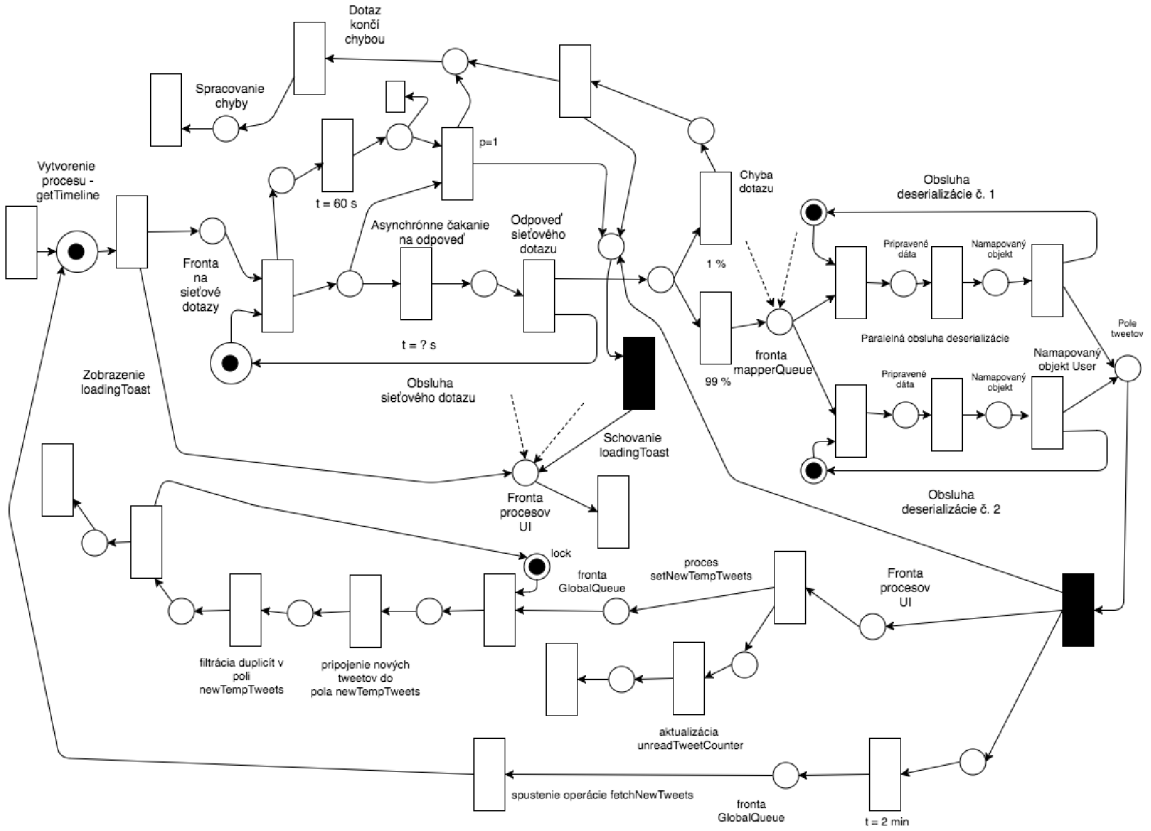


Obr. 3.3: Získanie profilových dát

Užívateľ je v tomto momente prihlásený do svojho profilu.

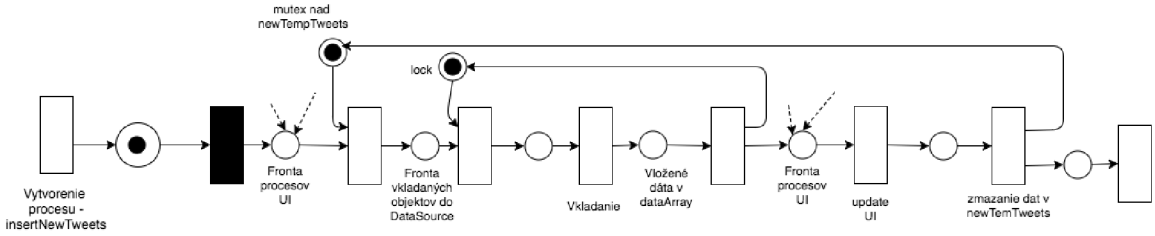
3.1.2 Prezeranie obsahu

Ďalšou z funkcionalít aplikácie je prezeranie časovej osi užívateľa s konkrétnymi tweetmi. Tento proces sa musí vykonávať pravidelne v určitých časových intervaloch. Tento interval je nastavený na hodnotu 2 minút, aby sa zamedzilo prekročeniu `Rate limitu` a užívateľ mal stále pocit aktuálnosti dát aplikácie. Proces je vyobrazený na obrázku č. 3.4.



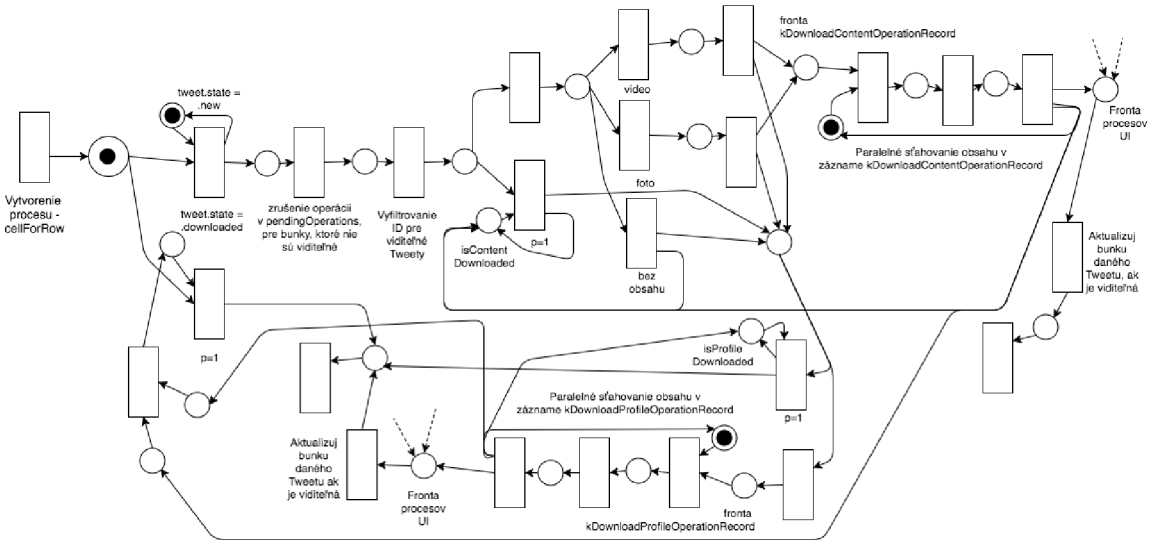
Obr. 3.4: Získanie obsahu časovej osi

Tweety z poľa `newTempTweets` sa zobrazia v UI užívateľovi, až keď sa užívateľ dostane na úplný vrchol zoznamu tweetov. Tento proces popisuje obrázok č. 3.5.



Obr. 3.5: Vkladanie obsahu do UI

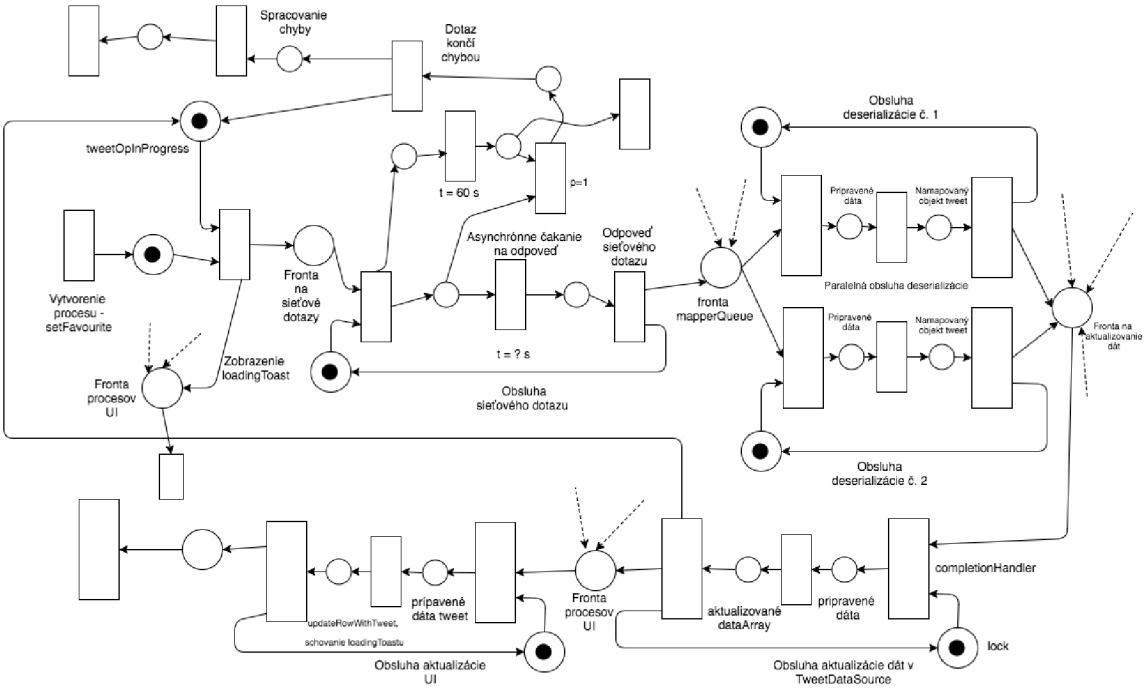
Keď sú samotné tweety dostupné v UI užívateľa, tak sa musia spustiť procesy ktoré stiahnu profilové fotky autorov jednotlivých tweetov a náhľady pre potencionálny audiovizuálny obsah tweetu. Je nutné podotknúť, že procesy sa spúšťajú len pre viditeľné bunky, aby sa zamedzilo nežiadúcemu zaťažovaniu siete a zdrojov. Tieto procesy sú vyobrazené na obrázku č. 3.6.



Obr. 3.6: Sťahovanie profilových obrázkov a náhľadu audiovizuálneho obsahu

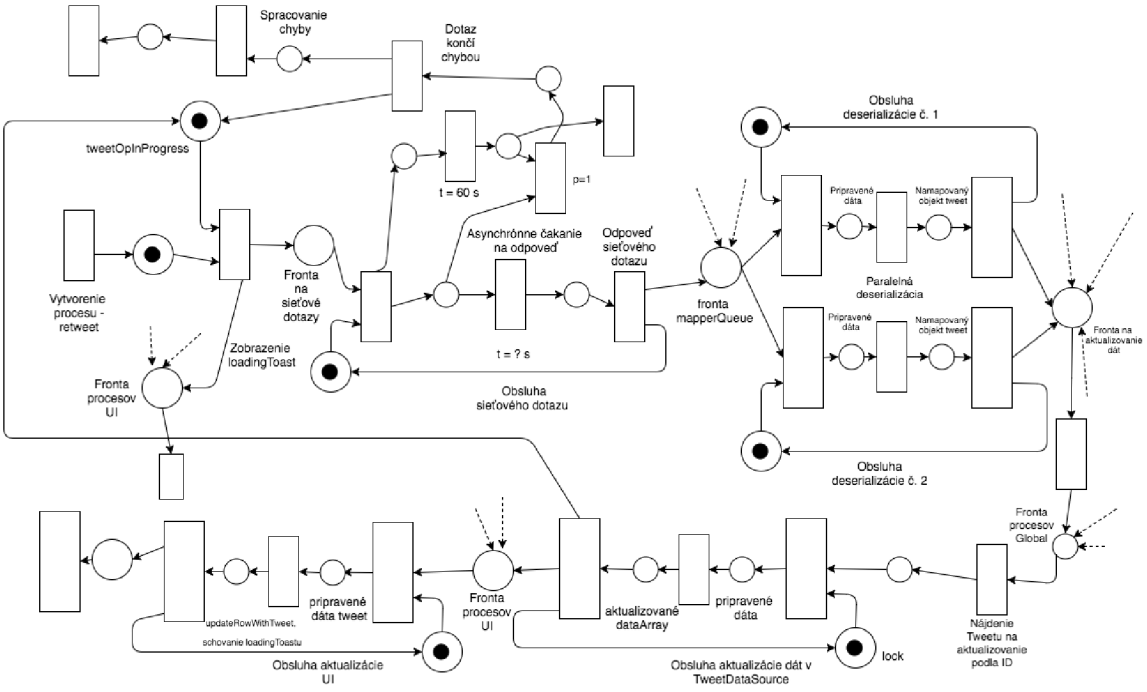
3.1.3 Like a Retweet

Jednou z hlavných funkcionalít aplikácie je možnosť označiť/odznačiť Tweet za obľúbený. Tento proces `setFavourite/setUnfavourite` je zobrazený na obrázku č. 3.7.



Obr. 3.7: Petriho sieť pre funkcionalitu Like

Druhou hlavnou funkcionalitou je možnosť označiť/odznačiť Tweet za retweetnutý. Tento proces `retweet/unretweet` je zobrazený na obrázku č. 3.8.



Obr. 3.8: Petriho sieť pre funkcionalitu Retweet

3.1.4 Diskusia

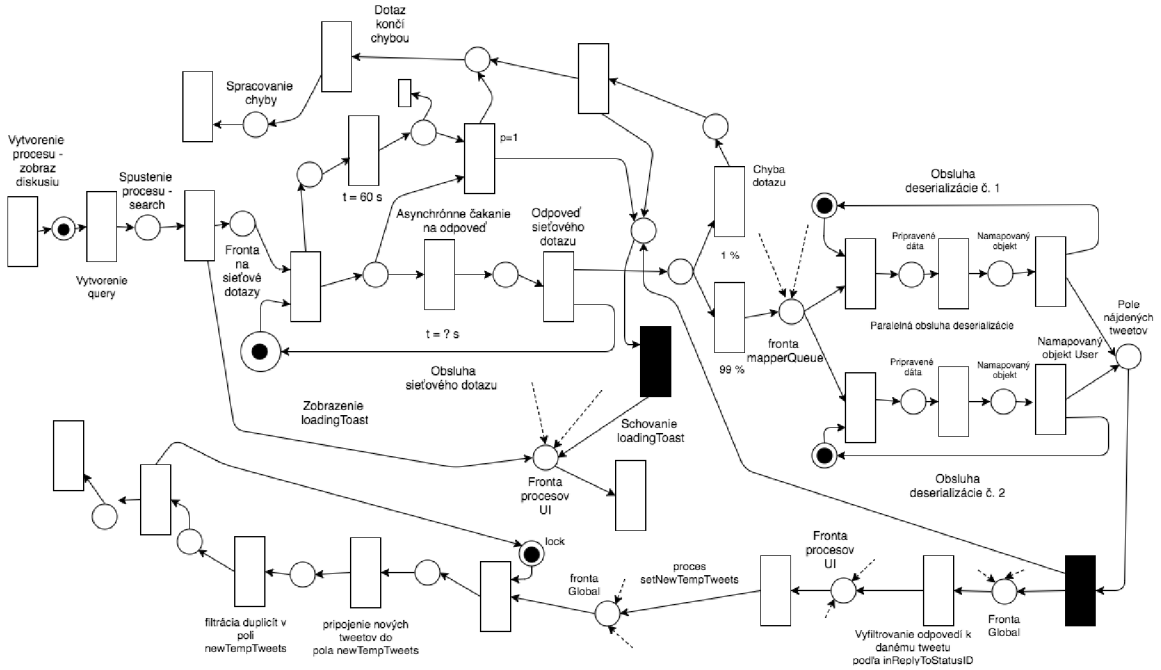
Treba podotknúť, že táto funkcionalita nemôže byť plnohodnotná. Keďže Twitter API nemá priamu podporu pre dotazovanie diskusie daného tweetu, tak bude potrebné simulovať túto funkcionalitu inými prvkami Twitter API.

Najskôr sa vyhľadá query pozostávajúce z `@` a zo `screenName`, čiže užívateľského mena autora tweetu. Výsledkom je nespočetne veľa tweetov, v ktorých je užívateľ spomenutý.

Následne je potrebné vyfiltrovať len tie tweety, ktoré sa viažu na tweet, ku ktorému sa zobrazuje diskusia. To sa dá zabezpečiť pomocou príznaku `inReplyToStatusID` každého tweetu. Ten sa porovná s ID diskutovaného tweetu.

Výsledok týchto dát sa zobrazí užívateľovi v jeho UI. Nevýhodou tohto prístupu je, že vyhľadávanie query vráti len obmedzený počet najnovších tweetov. Preto sa môže stať, že užívateľ neuvidí žiadne položky diskusie, aj keď reálne existujú.

Tento proces zobrazuje obrázok č. 3.9.



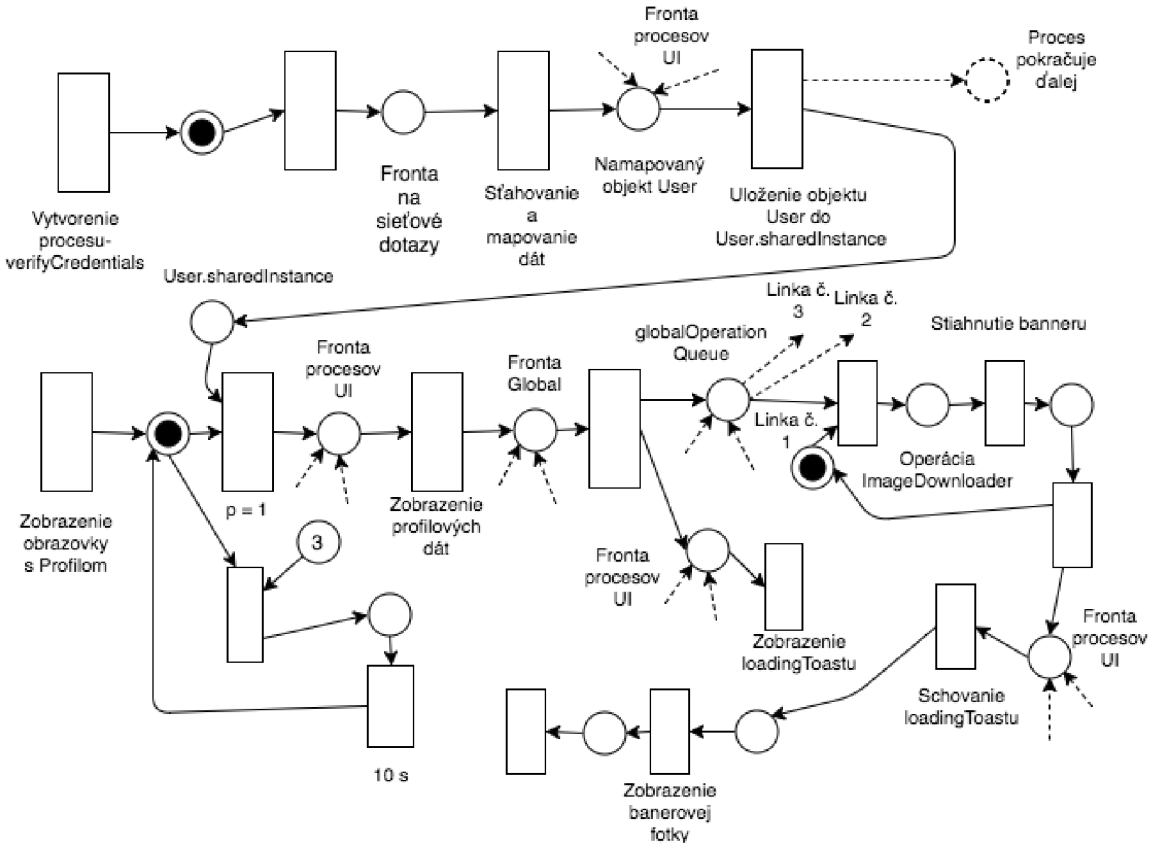
Obr. 3.9: Petriho sieť pre zobrazenie diskusie

3.1.5 Prezeranie profilu

Jednou z ďalších funkcionalít aplikácie je možnosť prezerania informácii o profile užívateľa. Táto funkcionalita je závislá na dátach získaných z procesu `verifyCredentials`, popísanej na obrázku č. 3.3.

V momente keď sa zobrazí obrazovka s profilom, tak sa musí overiť, že tieto dáta o profile sú dostupné. Ak nie sú, tak sa generuje ďalšia kontrola o 10 sekúnd neskôr. Toto znovu kontrolovanie môže prebehnúť maximálne 3-krát.

Ak sú dáta dostupné, tak sa zobrazia užívateľovi v UI a generuje sa dotaz na stiahnutie jeho banerovej fotky. Tento proces zobrazuje obrázok č. 3.10.



Obr. 3.10: Petriho sieť pre prezeranie profilu

3.2 Návrh UI aplikácie

V tejto podkapitole si predstavíme návrh samotného užívateľského rozhrania aplikácie. Predstavíme si jednotlivé obrazovky aplikácie pomocou schematického návrhu a popíšeme si jednotlivé akcie, ktoré bude možné vykonávať v UI aplikácie.

3.2.1 Schématický návrh UI

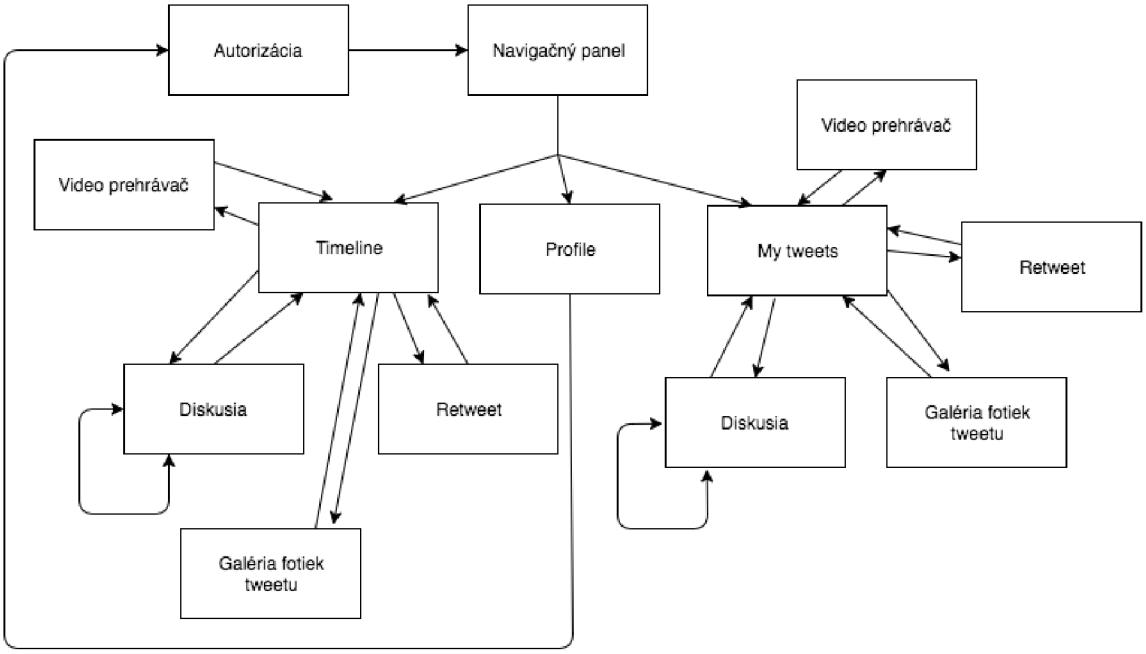
Vzhľad užívateľského rozhrania patrí medzi hlavné faktory prvého dojmu užívateľa. Preto som zvolil intuitívne a vzhľadovo príjemné prostredie.

Obrázok 3.11 predstavuje schému jednotlivých obrazoviek. Východiskovou obrazovkou je Autorizácia, na ktorej sa užívateľ prihlasuje pomocou pin kódu. Následne sa zobrazí kontajnerová obrazovka Navigačný panel. Tá predstavuje UITabbarViewController, ktorý umožňuje pomocou hornej navigačnej lišty prepínať medzi obrazovkami Timeline, Profile a My tweets.

Obrazovka Timeline sa v Navigačnom paneli zobrazí vždy ako prvá. Obsahuje tweety z časovej osi užívateľa. Z tejto obrazovky má užívateľ následne možnosť prejsť do obrazovky pre Retweet, či spustiť Video prehrávač pre video obsah daného tweetu. Ďalej môže zobraziť obrazovku Galéria fotiek pre daný tweet. Pre konkrétny tweet sa dá zobraziť aj obrazovka Diskusia. Tá je špecifická tým, že ak zobrazí nejaké príspevky v diskusii, tak je možné sa hierarchicky vnárať do diskusií pre tieto príspevky. Ide teda o UINavigationController.

Zo schémy je jasné, že ako pri obrazovke Timeline tak aj pri obrazovke My tweets má užívateľ rovnaké možnosti. To znamená hierarchicky sa vnárať do diskusií pre svoje vlastné tweety, či spúšťať video prehrávač alebo otvoriť galériu fotiek daného tweetu. Užívateľ môže rovnako ako vo webovom rozhraní Twittru, tak aj v tejto aplikácii Retweetnuť svoje vlastné tweety.

Obrazovka Profile obsahuje základné informácie o užívateľovom profile. Ide o počítadla sledovaných a sledujúcich užívateľov, počítadlo tweetov či obľúbených tweetov. Na tejto obrazovke je možné sa odhlásiť a prejsť späť na východiskovú obrazovku Autorizácia.



Obr. 3.11: Schéma UI aplikácie

3.2.2 Akcie UI

Obrazovka Autorizácia obsahuje jedno tlačítko `Next`. Na jeho kliknutie sa overí, že užívateľ zadal číselný PIN kód. Následne sa vyvolá proces získania `access tokenu`, popísaný na obrázku č. 3.2. Po jeho úspešnom dokončení sa prechádza na obrazovku Navigačný panel zobrazujúcu obrazovku Timeline.

Obrazovka Timeline a My tweets majú rovnaké typy akcií. Preto si ich popíšeme spoločne. Na týchto obrazovkách je možné použiť akcie diaľkového ovládača `swipe left`, `swipe right`, `double click`, `click` či `long press click`.

- **Swipe right:** Potiahnutím doprava na sklenenej dotykovej ploche diaľkového ovládača sa vyvolá spustenie procesu Like (obrázok č. 3.7).
- **Swipe left:** Potiahnutím doľava na sklenenej dotykovej ploche diaľkového ovládača sa buď zobrazí obrazovka pre Retweet alebo sa spustí proces Unretweet (obrázok č. 3.8). Záleží na vnútornom stave `retweeted` daného tweetu.
- **Click:** Kliknutím na daný tweet sa prechádza na obrazovku Diskusia a spúšťa sa proces pre diskusiu (obrázok č. 3.9).

- **Double click:** Spúšťa proces vyskrolovania navrch v zozname tweetov.
- **Long press click:** Spúšťa proces pre zobrazenie obrazovky Galérie alebo Video prehrávača pre daný tweet.

Obrazovka Profile zobrazuje tlačidlo `Logout`, ktoré užívateľovi umožňuje odhlásiť sa a prejsť na obrazovku Autorizácia.

Obrazovka Retweet obsahuje tlačidlo `Cancel`, ktorým sa zruší daná obrazovka a vráti sa na predchádzajúcu obrazovku. Tlačidlo `Retweet` spustí proces pre poslanie retweetu (obrázok č. 3.8).

Obrazovka Diskusie obsahuje vstup pre pridávanie príspevkov do diskusie. Ide o UITextField, ktorý po rozkliknutí umožní užívateľovi zadať vstup a tlačidlom `Send` tento vstup odoslať ako nový príspevok.

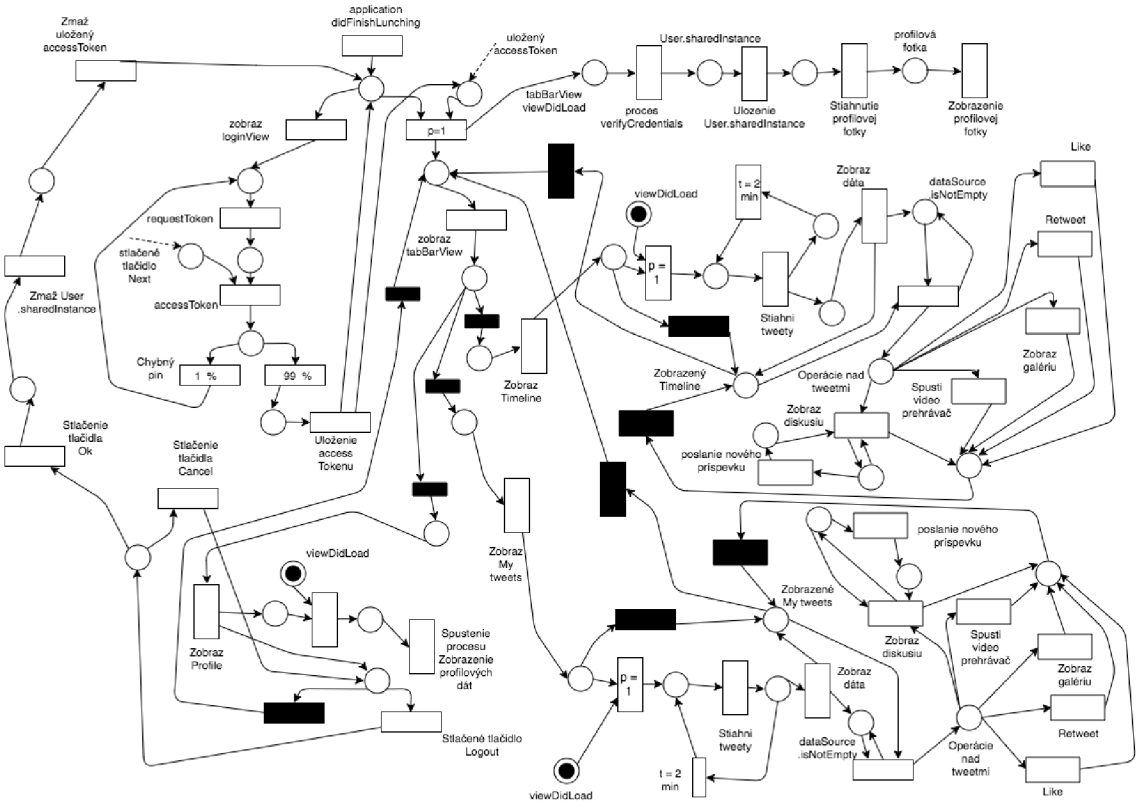
3.3 Celkový kontext chodu aplikácie

Aplikácia začína svoj kontext pri spustení v bode `didFinishLunchingWithOptions`. V tomto bode sa aplikácia rozhodne, či užívateľovi zobrazí obrazovku s Autorizáciou alebo rovno Navigačný panel s časovou osou. Rozhoduje sa podľa toho, či sa jej podarí načítať uložený `accessToken` alebo nie. V prípade, že nie, pokračuje aplikácia autorizáciou. Pokúsi sa požiadať o request token a keď užívateľ zadá PIN kód a stlačí tlačidlo `Next`, tak požiada o accessToken. Ten sa pri úspešnom obdržaní uloží.

Následne sa užívateľovi zobrazuje obrazovka s Navigačným panelom a v ňom ako prvá obrazovka Timeline. Zobrazenie navigačného panelu vyvolá akciu na spustenie procesu `verifyCredentials`, čím sa stiahne meno a prezývka užívateľa a link na jeho profilovú fotku. Následne sa spustí operácia na stiahnutie jeho profilovej fotky. Dáta sa zobrazia v hornom navigačnom paneli.

Užívateľ si môže vybrať a prepínať medzi obrazovkami Timeline, Profile a My Tweets. Pri obrazovke Profile dochádza okrem zobrazenia profilových dát a stiahnutia banerovej fotky aj k zobrazeniu tlačidla `Logout`. Pri jeho stlačení je užívateľ vyzvaný na potvrdenie akcie tlačidlom `Ok`. Jeho stlačením sa zmaže kontext užívateľa. To zahŕňa jeho accessToken a dáta uložené v User.sharedInstance. Následne je užívateľovi zobrazená stránka s Autorizáciou.

Celkový kontext chodu aplikácie zobrazuje obrázok č. 3.12.



Obr. 3.12: Petriho sieť pre celkový kontext aplikácie

Pre lepšiu predstavu celkového kontextu bol vytvorený aj diagram prípadov užitia, ktorý sa nachádza v prílohe D.1.

Kapitola 4

Implementácia aplikácie

V tejto kapitole si popíšeme všeobecné informácie k implementácii aplikácie (voľba jazyka, prostredia a pod.), spôsob implementácie mnohovláknovosti a popíšeme si ako sa správajú jednotlivé vlákna.

Keďže jednotlivé funkcionality aplikácie boli modelované v návrhu aplikácie ako procesy petriho sietí, tak si popíšeme niektoré časti kódu, ktoré odpovedajú daným procesom.

Následne sa zameriame na popis implementácie a využitia vlastného generického UITableView a DataSource v aplikácii.

Nakoniec sa budeme venovať spôsobu implementácie sieťovej komunikácie a protokolu OAuth.

V poslednej podkapitole si popíšeme bezpečnostné prvky aplikácie z hľadiska implementácie.

4.1 Všeobecné informácie

Pre implementáciu bol použitý jazyk Swift verzie 3.0 [18] a IDE spoločnosti Apple XCode 8.2.1 [24]. Pre sieťovú komunikáciu s Twitterom bolo použité REST API verzie 1.1 [54]. Implementácia aplikácie dodržuje softwarovú architektúru Model - View - Controller.

Aplikácia obsahuje 1 hlavný Storyboard, ktorý implentuje View pre všetky obrazovky. V tomto Storyboarde sa využívajú základné UI objekty ponúkané IDE rozhraním (UILabel, UITextField, UIView...) a autolayout. Jeho náhľad zobrazuje príloha E.1.

Pri tvorbe aplikácie bol využitý nástroj pre správu závislostí CocoaPods [25]. Prehľad použitých závislostí v `Podfile` zobrazuje príloha E.2.

4.2 Mnohovláknovosť

Pre zavedenie mnohovláknovosti do aplikácie nám tvOS, rovnako ako iOS, ponúka 2 štandardné API. Je to **Grand Dispatch Queue** [8] a **Operation** [13] (nové pomenovanie v Swift 3.0, totožné s NSOperation). Pri implementácii aplikácie boli využité oba prístupy. Ich bližšie využitie v jednotlivých častiach aplikácie si popíšeme v nasledujúcich podkapitolách.

4.2.1 Grand Dispatch Queue

Okrem vlastnosti vykonávať bloky kódu asynchrónne a paralelne, ponúka aj možnosť plánovať vykonanie týchto blokov na určitý čas. Bloky sa následne vkladajú do front. Dá sa

pracovať so sériovými frontami a paralelnými frontami. Grand Dispatch Queue obsahuje 2 hlavné fronty, a to frontu *Main*, ktorá je sériová a vykonávajú sa z nej všetky kúsky kódu, pokiaľ nie je explicitne definovaná iná fronta. Táto fronta slúži primárne na úlohy spojené s UI- aktualizácia, prekreslenie, zobrazenie dát a pod. Druhá fronta je *Global* s paralelným prístupom k vykonávaniu blokov kódu. Tejto fronte sa dá definovať istá *kvalita vykonávania (Quality of Service)* [14]. Podľa nej sa určí priorita daného bloku, s čím súvisí rýchlosť jeho vykonania.

Okrem iného toto API umožňuje vytváranie vlastných sériových či paralelných front podľa potreby.

Typickým príkladom využitia tohto API je, keď potrebujeme vykonať zložitú úlohu na pozadí z fronty *Global*, bez blokovania UI a po jej dokončení sa aktualizuje UI z fronty *Main*. To sa využíva dosť často aj v našej aplikácii.

Nasledujúce tabuľky popisujú využitie tohto API v konkrétnych situáciách, a to podľa typu danej fronty (Main/Global) a jej triedy kvality. V prvom stĺpci sú vždy bloky, ktoré sa zaraďujú do danej fronty z fronty `Main`, v druhom stĺpci sú bloky, ktoré sa spúšťajú po dokončení kódu z predchádzajúceho stĺpca. Následne je vždy popísaná podmienka zahájenia vykonávania blokov.

Tabuľka 4.1: Vykonávanie blokov z fronty Global s triedou kvality User Interactive

global- userInteractive	main	podmienka spustenia
odhlásenie užívateľa a pripravenie LoginViewController	do RootViewControllera sa nastaví LoginViewController pomocou animácie	užívateľ stlačí tlačidlo Logout v profile
vyskrolovanie tableView pre tweety na vrch bez animácie	aktualizácia zaostrenia na prvú bunku tableView + vynulovanie počítadla nových tweetov	užívateľ spraví dvojklik na diaľkovom ovládači
dekódovanie HTML entít v texte, odstránenie URL adresy pre média z textu, vizuálne odlíšiť hashtagy a anotácie užívateľov	zobrazenie NSAttributedStringu do danej bunky	volanie funkcie `cellForRow()`
nájdenie daného tweetu v dataSource podľa jeho id a aktualizácia tohto objektu	update pre bunku s týmto tweetom	volanie funkcie `updateRow WithTweet(tweet: Tweet)` - v prípade nastavenia tweetu ako obľúbeného alebo ako neobľúbeného

Tabuľka 4.2: Vykonávanie blokov z fronty Global s triedou kvality Utility

global- utility	main	podmienka spustenia
nájdenie IndexPath pre viditeľné bunky	update buniek s nájdenými IndexPath + opakované naplánovanie	musí byť povolené pomocou premennej `autoupdateVisibleCells: Bool` a po uplynutí času `autoUpdateTime` (5 minút)

Tabuľka 4.3: Vykonávanie blokov z fronty Global s triedou kvality Default

global- default	main	podmienka spustenia
vytvorenie QR kódu pre dané URL	zobrazenie QR kódu do daného UIImageView	keď je url adresa s access tokenom dostupná a má sa zobraziť užívateľovi

Pri nasledujúcich blokoch musíme mať istotu, že prvotné bloky začnú vykonávanie z fronty `Main`. Ide o aktualizácie UI, a až po ich dokončení sa vykonajú príslušné bloky s dátami na pozadí v inom vlákne.

Keďže nemáme istotu, že tieto bloky na aktualizáciu UI budú vždy volané s hlavného vlákna `Main`, tak sú explicitne do tohto vlákna zaradené.

Tabuľka 4.4: Bloky kritické na volanie z vlákna Main

main	global- default	podmienka spustenia
vloženie tweetov z dočasného úložiska do dataSource pre UITableView	zmazanie dočasne uložených tweetov v premennej `newTempTweets: [Tweet]`	keď užívateľ vyskroluje tableView navrch
nastavenie počítadla neprečítaných tweetov	vloženie nových tweetov do dočasného úložiska `newTempTweets: [Tweet]` a zmazanie prípadných duplikátov	vždy keď sa stiahnu nové tweety
overenie či treba zobraziť hlásenie žiadne dáta	**background- async after pollingInterval (2 min)**	plánuje sa len ak je
	poslanie dotazu na stiahnutie nových tweetov	dostupné pripojenie na internet

4.2.2 Operation

Výhoda prístupu pomocou Operation v porovnaní s Grand Dispatch Queue je tá, že sa dá zisťovať stav danej operácie, či je pripravená, ukončená alebo zrušená. Medzi jednotlivými operáciami sa dajú vytvárať závislosti, že jedna operácia sa má vykonať, až keď sa iná určitá operácia dokončí. Okrem toho ponúka možnosť spustiť kód po ukončení operácie a hociktorá, prípadne všetky naplánované operácie vo fronte, môžu byť zrušené, keď je treba.

To sa dá výhodne využiť pri práci s UITableView. V našom Twitter klientovi pre Apple TV sú dané tweety zobrazované pomocou UITableView. Každý tweet je reprezentovaný objektom, ktorý môže mať ešte informácie o profilovej fotke, url miniatúry obrázka alebo náhľadu videa daného tweetu. Pre stiahnutie týchto obrázkov je výhodne použiť prístup cez API Operation. Nechceme aby sa pri poli s 500 tweetmi začalo súčasne sťahovať profilová fotka aj náhľad obsahu pre všetky objekty. Daná operácia sa preto spustí až v momente, keď je bunka viditeľná. V opačnom prípade, sa operácie pre už neviditeľné bunky vo fronte zrušia a nedochádza k nadbytočnej záťaži siete a využívania zdrojov.

Pre sťahovanie týchto obrázkov sa využíva trieda `ImageDownloader`, ktorá dedí od triedy Operation. Tá očakáva pri inicializácii referenciu na objekt, ktorý vyhovuje protokolu `ImageProtocol`, vďaka čomu nám nezáleží na dátovom type objektu. Protokol nám ale zabezpečí, že máme prístup k URL adrese, referenciu na uloženie stiahnutých dát a stavu sťahovania. Okrem toho inicializácia očakáva typ obrázku, o ktorý máme záujem, podľa čoho sa následne upraví URL adresa a pridajú sa potrebné query parametre pre stiahnutie

daného typu obrázku. Ide o query parameter špecifikovaný v Twitter API pre stiahnutie obrázku v určitom rozlíšení [52]. Pri vykonávaní operácie sa priebežne kontroluje jej stav, či náhodou nebola zrušená, aby sa zamedzilo zbytočnému vykonávaniu kódu.

Tieto operácie sú následne zaradené do patričnej fronty a vykonávané. Ich prehľad zobrazuje nasledujúca tabuľka č. 4.5.

Tabuľka 4.5: Prehľad Operation v aplikácii

Queue	Operations	Main	Podmienka spustenia
`DOWNLOAD_PAGE_IMAGE_QUEUE`	Stiahnutie obrázka pre objekt MEDIA daného tweetu	Schovanie indikátora aktivity, zobrazenie obrázku a zmazanie danej Operation zo zoznamu inProgress	Užívateľ spustí dlhým stlačením nad bunkou, ktorá obsahuje médium s obrázkami a spušťa sa ak daný obrázok ešte nebol stiahnutý
globalOperationQueue	Stiahnutie banneru pre profil a uloženie ho do objektu User pre prihláseného užívateľa	Schovanie indikátora aktivity, zobrazenie banneru	Užívateľ prepne v UITabBare položku Profile
	Stiahnutie užívateľovej profilovej fotky a uloženie do objektu pre prihláseného užívateľa User	Nastavenie profilovej fotky do vizitky užívateľa v UITabBare	Pri inicializácii TabBarViewControlleru sa zistí, či je internetové pripojenie dostupné a či sú dostupné dáta prihláseného užívateľa, spúšťa sa až keď sú tieto podmienky splnené
`DOWNLOAD_PROFILE_QUEUE`	Stiahnutie profilovej fotku autora tweetu a uloženie jej do objektu daného tweetu	Volanie aktualizácie UI bunky s daným IndexPath nájdeným pomocou id tweetu a zobrazenie profilovej fotky	Daná bunka je viditeľná +
`DOWNLOAD_CONTENT_QUEUE`	Stiahnutie miniatúry obrázka alebo náhľadu videa daného tweetu	Volanie aktualizácie UI pre bunku daným indexPath a zobrazenie miniatúry	operácia pre dané id tweetu ešte nebola spustená

Prácu s API Operation nám zapuzdruje štruktúra `OperationQueueRecord<T: Hashable>` a trieda `PendingOperations<T: Hashable>` v súbore **Operations.swift**. Keďže z `OperationQueue` je problémové pristupovať ku konkrétnej operácii, tak štruktúra `OperationQueueRecord<T: Hashable>` obsahuje ešte slovník. Ten uchováva operácie, ktoré

sú aktuálne v procese vykonávania, podľa špecifického kľúča. Okrem toho obsahuje ešte meno, ktorým sa identifikuje celá daná štruktúra.

Trieda `PendingOperations<T: Hashable>` nám následne pomocou slovníka uchováva všetky potrebné záznamy `OperationQueueRecord`, kde kľúčom je práve meno tohto záznamu. Okrem toho implementuje metódy pre vkladanie, mazanie, či čítanie špecifického záznamu podľa jeho mena.

Vďaka tomu môžeme mať v danom UITableView vytvorené záznamy pre sťahovanie profilových fotiek autorov jednotlivých tweetov a záznamy pre sťahovanie náhľadu audiovizuálneho obsahu jednotlivých tweetov. V prípade, že sa rozhodneme pridať v budúcnosti nový typ operácie s vlastnou frontou, tak jej stačí vytvoriť len vlastný záznam `OperationQueueRecord`.

4.2.3 Tutoriál aplikácie

Pri zobrazovaní úvodného tutoriálu, ktorý informuje užívateľa, ako používať gestá a dvojklik v aplikácii, sa tiež používa Grand Dispatch Queue. Tutoriál obsahuje viacero častí, ktoré na seba nadväzujú. Preto je výhodné využiť možnosť načasovať spustenie jednotlivých častí na konkrétny čas. Nasledujúca tabuľka zobrazuje, ako sú jednotlivé časti načasované. Spúšťajú sa vždy asynchrónne a z Main Queue.

Tabuľka 4.6: Prehľad blokov tutoriálu

Main	**Main - after durationTime**
Zobraz baner s nápovedou na gesto posunu doprava a rozblikaj pravý panel s počítadlom	Schovaj baner a vypni blikanie pravého panelu
Main - after durationTime + waitTime	**Main - after durationTime**
Zobraz baner s nápovedou na gesto posunu doľava a rozblikaj ľavý panel s počítadlom	schovaj baner a vypni blikanie ľavého panelu
Main - after 2 *(durationTime + waitTime)	**Main - after durationTime**
Zobraz nápovedu o dlhom podržaní pre zobrazenie obsahu tweetu	Schovaj baner
Main - after 3 *(durationTime + waitTime)	**Main - after durationTime**
Zobraz nápovedu o dvojkliku pre vyskrolovanie navrch v tableView	Schovaj baner

4.2.4 Overovanie dostupnosti pripojenia na internet

Keďže aplikácia je vysoko závislá na pripojení do siete internet, je preto potrebné v jej réžii kontrolovať stav tohto pripojenia. To znamená, či má zariadenie Apple TV prístup k internetu alebo nie. Následne musí aplikácia patrične zareagovať na zmenu tohto stavu.

Pre prípady kedy je potrebné overovať dostupnosť resp. nedostupnosť internetu bola použitá voľne dostupná knižnica Reachability [34] za pomoci nástroja na správu závislostí a externých knižníc CocoaPods [25]. Tá má implementovanú svoju vlastnú sériovú frontu pre vykonávanie blokov, keď je pripojenie k internetu dostupné resp. nedostupné. Bloky, ktoré sa spracovávajú cez túto frontu sú popísané v nasledujúcej tabuľke 4.7.

Tabuľka 4.7: Prehľad blokov závislých na pripojení do siete internet

reachabilitySerialQueue	main	podmienka spustenia
over dostupnosť Tokenu pre autentizáciu a ak je nedostupný, tak požiadaj o nový	–	zistí sa že internetové pripojenie je aktívne
over dostupnosť dát prihláseného užívateľa, ak sú nedostupné tak ich stiahni	nastavenie profilových dát do UITabBaru	
zavolanie funkcie pre stiahnutie nových tweetov		
Overenie či sú nejaké stiahnuté tweety	ak treba, tak zobraz text, že tweety nie sú dostupné	zistí sa, že internetové pripojenie je neaktívne

4.3 Implementácia vybraných petro-sieťových procesov

V kapitole 3.1 sme si popísali a navrhli jednotlivé funkcionality aplikácie ako procesy petriho sietí. Teraz si ukážeme niektoré vybrané časti týchto návrhov v implementovanej podobe.

4.3.1 Získanie request tokenu

Tento kus kódu sa spúšťa, ak je dostupné internetové spojenie, resp. toto spojenie bolo obnovené. Ide teda o časť `Vytvorenie procesu - requestToken`, ktorá je zobrazená na obrázku č. 3.1.

```
if self.authenticationURLLabel.text == nil ||
   self.authenticationURLLabel.text == "" {
        self.accountService
            .requestToken(
                completionHandler: self.setupViewWithURLFrom
            )
        Toast.showLoadingToast()
}
```

Po dokončení sieťovej operácie a obdržaní odpovede dochádza ku volaniu callbacku `setupViewWithURLFrom`. Táto funkcia odpovedá časti za `Odpoveď sieťového dotazu` v sieti č. 3.1:

```
private func setupViewWithURLFrom(rawTokens tokens: String?,
                                  error: CustomError?) {
    Toast.hideLoadingToast()

    if let tkns = tokens  {

        Bootstrapper.twitterHandler
                    .setOAuthToken(rawData: tkns)

        let accessUrl = accountService
                        .getAuthenticateAccountURL()
```

```
            self.authenticationURLLabel.text = accessUrl

            DispatchQueue.global().async {
                let qrCode = UIImage
                    .createQRForString(qrString: accessUrl)

                DispatchQueue.main.async {
                    self.qrImageView.image = qrCode
                }
            }
        }
        else {
            Utils.showErrorAlert(in: self, error: error!)
        }
    }
```

4.3.2 Prístup k dátam TweetDataSource

Fungovanie DataSourcu bude detailnejšie popísané v kapitole č. 4.5. V tejto podkapitole si ukážeme iba, ako je implementovaná práca s dátami ako napríklad na obrázku č. 3.8 v časti `Obsluha aktualizácie dát`.

DataSource má privátnu premennú `lock`:

```
fileprivate let lock = NSLock()
```

Pri práci s týmto zámkom sa používa prístup pomocou konštrukcie `defer`, ktorá vykonáva kód po ukončení danej funkcie. Pre vloženie dát na danú pozíciu sa potom používa nasledujúca funkcia:

```
internal func insertData(at ip: IndexPath, data: T) -> Bool {
    defer {
        lock.unlock()
    }
    lock.lock()

    var predicate = existsDataAt(indexPath: ip) || ip.row == 0

    if ip.section < dataArray.count {
        predicate = predicate ||
              ip.row == dataArray[ip.section].count
    }

    if predicate {
        self.dataArray[ip.section].insert(data, at: ip.row)

    }

    return predicate
}
```

4.3.3 Vkladanie nových tweetov

Na obrázku č. 3.5 sme si navrhli ako sa vkladajú tweety do UI užívateľovi, keď sa dostane na úplný vrchol zoznamu tweetov. Nasledujúca funkcia ukazuje implementáciu uzamykania premennej `newTempTweets` pomocou mutexu. Premenná je odomknutá až v callbacku po ukončení daného vkladania:

```
func insertNewTweets() {
    DispatchQueue.main.async {
        objc_sync_enter(self.newTempTweets)
        self.insertArrayAtTop(data: self.newTempTweets,
                              completion: {
            if !self.newTempTweets.isEmpty {
                self.newTempTweets.removeAll()
            }

            DispatchQueue.main.async {
                self.showNoDataLabel()
            }
            objc_sync_exit(self.newTempTweets)
        })
    }
}
```

Funkcia `insertArrayAtTop` je funkciou DataSourcu a používa svoj zámok popísaný vyššie v kapitole č. 4.3.2.

4.4 Generické UITableView

Trieda `ExTableViewController<T: CusComparable>: UITableViewController` dedí od UITableViewControlleru a rozširuje ho o mnohé praktické metódy. Ide o generický kontroler, kde argument `T` určuje dátový typ pre `DataSource` daného UITableView. Tento dátový typ `T` musí ale implementovať protokol `CusComparable`, ktorý zabezpečí, že obsahuje premennú `var id: String! {get}`. Tento protokol nám zabezpečuje porovnateľnosť jednotlivých dát v DataSource UITableView.

Táto generická trieda pre UITableView nám následne zapuzdruje určité výhodné metódy a premenné. Súvisia hlavne s prácou s Operation API, ako premenná dátového typu `PendingOperations` (popísané na strane 25) či callback `var onScreenCellHandler: (() -> ())?`. Ten slúži na vykonanie kódu, keď je daná bunka viditeľná pre užívateľa. Ďalej obsahuje metódy pre pozastavenie či opätovné spustenie všetkých operácii uložených v `PendingOperations`.

Okrem toho táto trieda obsahuje metódy pre vkladanie novej bunky s danými dátami do UITableView, či rovno celého poľa buniek s dátami typu `T` na danú pozíciu. O vkladanie dát sa starajú metódy v triede `GenericDataSource`. Tá je popísaná v ďalšej kapitole 4.5.

Tento vlastný UITableView umožňuje povoliť či zakázať automatické obnovovanie UI viditeľných buniek. To sa vykonáva v 200 sekundových intervaloch pomocou Grand Dispatch Queue. Táto funkcionalita súvisí s tým, že bunka pre tweet zobrazuje koľko času ubehlo od jeho vytvorenia. Tento čas by sa mal teda prepočítavať a prekresľovať v prípadoch, že bunka je dlhšiu dobu zobrazená a UITableView sa nepohybuje.

4.5 Generický DataSource

`GenericDataSource<T: CusComparable>: NSObject, UITableViewDataSource` je generický Data Source, a jediné miesto odkiaľ sa môže čítať či zapisovať do dát daného UITableView. Obsahuje premennú `fileprivate var dataArray = [[T]]()`. Ide o 2D pole, kde jeden rozmer predstavuje sekcie a druhý riadky daného UITableView.

Dôležitosť tohto generického DataSource spočíva v premennej `fileprivate let lock = NSLock()`. Ide o zámok používaný nad dátami v premennej `dataArray`. Pri každom pokuse o čítanie, zápis či zmenu týchto dát, sa daný zámok uzamkne a odomyká sa až po vykonaní kritickej sekcie.

Využíva sa vo funkciách pre čítanie, mazanie či zapisovanie dát pre konkrétne `IndexPath`. Návratovou hodnotou týchto metód je vo väčšine prípadov **Bool** hodnota, ktorá informuje, či sa daná operácia úspešne podarila.

Tento DataSource umožňuje nastaviť maximálny počet riadkov v danej sekcii, ktoré sa majú uchovávať v pamäti zariadenia. Defaultne je nastavená hodnota `1000`. Ide o akési opatrenie pre nadmerné zaťažovanie pamäte. Pri vykonávaní metódy `func insertArray( section: Int, position: InsertPosition, data: [T]) -> [IndexPath]` sa zohľadňuje tento parameter. Ak je počet dát po vložení poľa väčší ako tento parameter, výsledné pole sa oreže na túto maximálnu veľkosť na spodnom konci poľa.

Ide o generický DataSource, takže sa očakáva jeho podedenie a nastavenie dátového typu. Preto z metód protokolu **UITableViewDataSource** sú implementované len `func numberOfSections(in tableView: UITableView) -> Int` a `func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int` podľa poľa `dataArray`. Metóda `func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell` je len akousi abstraktnou metódou (dôjde k pádu, ak nie je prepísaná) a očakáva sa, že v podedenej triede bude táto metóda prepísaná konkrétnou implementáciou.

4.6 Sieťová komunikácia

Pre posielanie sieťových požiadaviek je použitá open source knižnica Alamofire[1]. Tá je postavená nad NSURLSession a zapuzdruje operácie pre stiahnutie či odoslanie dát po sieti a vlastné typy požiadaviek. Vďaka tejto knižnici je kód prehľadnejší a čistejší.

Pri vykonávaní sieťových dotazov sa používajú viaceré fronty. Za prvé, sieťové požiadavky sú realizované pomocou NSURLSession, ktorá pracuje s vlastnou frontou implementovanou pomocou API Operation [12]. Vďaka tomu sa dajú jednotlivé požiadavky pozastaviť, spustiť znovu, či úplne zrušiť.

Následne má knižnica Alamofire implementovanú frontu v triede `TaskDelegate`, vďaka ktorej sa vykonáva validácia odpovede a serializačné procesy.

Po úspešnom prijatí deserializovanej odpovede vo formáte JSON je potrebné dané dáta namapovať na príslušné objekty. Mapovanie dát na objekty je časovo náročnejšia operácia, ak máme pole o veľkosti 500 tweetov. Preto je pre toto mapovanie vytvorená pomocou Grand Dispatch Queue vlastná paralelná fronta `cz.isp.mapper-queue`. Tá spracováva mapovanie dát na objekty v separátnom vlákne, aby nedochádzalo k blokovaniu UI. Pre serializáciu a deserializáciu dát bol zvolený POD ObjectMapper[29] pridaný pomocou nástroja CocoaPods [25]. Následne sa z vlákna `Main` spúšťa takzvaný `onCompletion` blok, ktorý odovzdá príslušnému kontroleru namapované dáta alebo objekt informujúci, že požiadavka skončila s nejakou chybou.

Daný `onCompletion` blok je callback, ktorý sa vykonáva asynchrónne po prijatí odpovede a jej deserializácii na príslušný objekt.

Ide teda o referenciu na funkciu, ktorá má vo väčšine prípadov nasledujúci dátový typ: `onCompletion: @escaping (_ tweet: Tweet?, _ error: CustomError?) -> ()`

Celá sieťová komunikácia je vyčlenená do samostatného celku v zložke **Networking**. Obsahuje zložku **Services** a zložku **Protocols**.

V zložke Services sú následne triedy implementujúce komunikáciu s Twitter REST API pomocou knižnice Alamofire [1] a deserializáciu odpovedí na objekty, resp. spätnú serializáciu. Aktuálne je v nej možné nájsť 2 triedy. `TweetService` implementuje dotazy na prácu s tweetmi užívateľa, ako načítanie zoznamu tweetov časovej osi, označenie tweetu ako obľúbeného, či poslanie retweetu. Trieda `AccountService` obsahuje implementácie dotazov pre prácu s účtom užívateľa. Ide teda o dotazy ako vyžiadanie si URL adresy pre autorizáciu užívateľa, vyžiadanie si `Access Tokenu` pre podpisovanie ďalších dotazov alebo overenie platnosti práv aktuálne prihláseného užívateľa, či ďalšie.

Zložka Protocols obsahuje protokoly k týmto triedam, čiže akési rozhrania daných dotazov. Prístup k daným metódam je zabezpečený cez triedu `Bootstrapper`, ktorá pomocou triednych metód umožňuje prístup k danému modulu dotazov. Pre ukážku:

```
class func tweetService() -> TweetServiceProtocol {
    return TweetService()
}
```

Vďaka tomu, že sa v aplikácii pre sieťové moduly používajú protokoly zapuzdrené v triednych metódach triedy `Bootstrapper`, tak je zabezpečená robustnosť a jednoduchá modifikovateľnosť kódu. Ak sa v budúcnosti zmení server s ktorým sa komunikuje alebo spôsob ako sa sním komunikuje, tak stačí vymeniť implementáciu v triedach v zložke Services za novšiu.

4.6.1 OAuth autorizácia

Takmer každý dotaz voči Twitter REST API vyžaduje, aby obsahoval v hlavičke dotazu položku s kľúčom `Authorization` a hodnotou v tvare `OAuth oauth_consumer_key=„%value%", oauth_token=„%value%",oauth_signature_method=„HMAC-SHA1",oauth_timestamp= „%value%",oauth_nonce=„%value%",oauth_version=„1.0",oauth_signature=„%value%"`. Hodnota **%value%** označuje patričnú hodnotu premennej.

Pre reprezentáciu hodnoty `oauth_consumer_key` slúži v kóde trieda **OAuthConsumer**. Tá uchováva privátny a verejný consumer kľúč celej aplikácie vygenerovaný v developerskej konzole Twitteru [59]. Okrem toho obsahuje premennú `callbackURL`, ktorý určuje adresu voči ktorej sa má vykonať dotaz s prijatým `oauth_token` parametrom [45]. Keďže v aplikácii využívame autentizáciu pomocou PIN kódu, je potrebné nastaviť pri inicializácii tohto objektu premennú `callbackURL` na hodnotu **„oob“** [50].

Trieda `Token` obsahuje zas privátny a verejný kľúč daného užívateľa, ktorý sa snaží o komunikáciu. Jeho verejný kľúč tvorí hodnotu položky `oauth_token`.

Následne máme triedu `SignatureMethod`, ktorá sa stará o vygenerovanie podpisového kľúča `oauth_signature`. Ide o akúsi pseudoabstraktnú triedu (keďže vo swifte nie je možné vytvárať abstraktné triedy), ktorá obsahuje funkcie pre získanie mena podpisovej metódy, vytvorenie podpisu, overenie platnosti daného podpisu či vytvorenie podpisového kľúču, takzvaného `signature base string` [43]. Pri prvých dvoch metódach, sa očakáva, že budú prepísané v podtriede, ktorá bude od tejto triedy dediť. Očakávajú teda implementáciu pre konkrétnu podpisovú metódu.

Trieda `HMAC-SHA1` je podtriedou triedy `SignatureMethod` a implementuje teda konkrétny typ podpisu. V tomto prípade ide o šifrovanie pomocou metódy HMAC-SHA1 [33], ktorú vyžaduje Twitter [43]. Pomocou privátnych kľúčov z objektu `Token` a `OAuthConsumer` sa vytvorí podpisový kľúč danej metódy. Pomocou tohto kľúča a danej metódy sa následne vypočíta výsledná hodnota podpisu pre daný dotaz.

Hodnota `oauth_timestamp` určuje časovú známku daného podpisu a hodnota `oauth_nonce` určuje unikátny identifikátor unikátneho dotazu [42].

Okrem toho modul pre OAuth autorizáciu obsahuje ešte ďalšie triedy, ktoré sú nadstavbou nad už popísanými časťami a zapuzdrujú ich. Ide o:

- **OAuthUtil** - Implementuje pomocné metódy, ako napríklad kódovanie a dekódovanie špeciálnych znakov v požiadavke pri počítaní výslednej hodnoty podpisu. Parametre dotazu musia spĺňať formát daný RFC3986 [49].
- **OAuthRequest** - Reprezentuje daný dotaz. Na základe parametrov požiadavky, HTTP metódy a URL adresy vytvorí pole `Authorization` potrebné pre hlavičku dotazu.
- **TwitterOAuth** - Je najvrchnejšou časťou tohto modulu. Zapuzdruje prácu s Tokenom, jeho ukladanie, nastavovanie, načítanie a vytvorenie podpisovej hlavičky pre daný dotaz. Všetka práca pri podpisovaní dotazov by sa mala vykonávať cez túto triedu. Jej inštancia ako singleton sa nachádza v triede `Bootstrapper`

4.7 Bezpečnosť aplikácie

Aplikácia viac-menej nepracuje s kritickými dátami. Vďaka OAuth 1.0 autorizácii užívateľ neposkytuje žiadne prihlasovacie meno ani heslo.

Komunikácia prebieha len medzi aplikáciou a Twitter API, a to za pomoci `HTTPS` protokolu. Tým sa podporuje ochrana `App Transport Security` zavedená Applom [17].

Jediným zraniteľnejším bodom je teda samotný `OAuth token` a jeho spôsob uchovávania na zariadení pre obnovenie kontextu daného užívateľa po opätovnom spustení aplikácie.

NSUserDefaults ponúka priestor na Apple TV ukladať perzistentné dáta (viď kapitola 2.3). Nie je to ale vhodné úložisko pre citlivé dáta. Je reprezentované formou `.plist` súboru, čiže istá forma XML, ktorú môže hocikto zo zariadenia prečítať [32].

Namiesto toho je vhodnejšie použiť `Keychain`, ktorá používa `Triple Digital Encryption Standard` na encryptovanie dát a je preto vhodnejšia pre uchovávanie citlivého obsahu [11].

V aplikácii sa práve pri ukladaní a čítaní OAuth tokenu používa Keychain v metódach triedy `TwitterOAuth`.

Kapitola 5

Testovanie aplikácie

V tejto kapitole si popíšeme, ako prebiehalo testovanie aplikácie. Najskôr si popíšeme testovanie pomocou Unit testov. Potom si načrtneme priebeh testovania v simulátore a testovanie na reálnom zariadení. Ku koncu kapitoly sa budeme zaoberať tým, ako prebiehalo publikovanie aplikácie do AppStoru a výsledkami testovania verejnosťou.

5.1 Unit testy

Niektoré celky aplikácie si vyžadovali vytvorenie unit testov pre zabezpečenie správnosti vykonávania kódu a zautomatizovania ich testovania. Tieto testy sa týkajú celkov, ktoré môžu byť náchylné na zmeny zvonku. Pre ich vznik bol použitý framework XCTest prostredia XCode.

Celkovo ide o 5 testovacích tried. `OauthTests` slúži na testovanie správnosti generovania oAuth podpisu sieťovej komunikácie. Trieda `TweetServiceTests` a trieda `AccountServiceTests` testujú funkčnosť komunikácie s Twitter REST API. Trieda `GenericDataSourceTests` testuje prácu s generickým dataSource, ako čítanie dát a zápis dát. Posledná trieda `ExTableViewTests` testuje generický UITableView, a to či sa jeho metódy vykonávajú s požadovaným výsledkom.

Vďaka týmto testom je jednoduchšie do budúcna odhaliť chyby spojené so zmenami v Twitter REST API, jej autorizácii alebo v práci s UITableView v aplikácii.

5.2 Testovanie v Simulátore

Vývojové prostredie XCode ponúka pre testovanie aplikáciu Simulátor, ktorá umožňuje otestovať aplikáciu na rôznych zariadeniach. Pre testovanie Simulátor, narozdiel od iOS zariadení, ponúka len 1 typ zariadenia Apple TV. Tento simulátor dokázal poskytnúť plnohodnotnú náhradu za reálne zariadenie. Aj napriek tomu má svoje limity a nedokáže odhaliť chyby, ktoré sa prejavia len na reálnom zariadení.

Testovanie prebiehalo po jednotlivých častiach. Po navrhnutí a implementácii časti kódu, bol tento úsek otestovaný a odladený. Následne sa aplikácia otestovala ako celok, či sa vplyvom novej časti kódu v aplikácii nevyskytla nová chyba. Ladenie týchto chýb veľmi uľahčovala konzola prostredia XCode. V prípade chyby sa aplikácia zastaví a v konzole sa vždy zobrazí posledne vykonávaná operácia. Prostredie XCode taktiež ponúka aj prehľad všetkých premenných a ich hodnôt v pamäti. V prípade potreby je možné nastaviť breakpoint na hociktoré miesto kódu a tieto hodnoty pri jeho vykonávaní preskúmať .

5.3 Testovanie na zariadení Apple TV

V priebehu tvorby bola aplikácia pravidelne testovaná na reálnom zariadení. Porovnávalo sa tak chovanie a dojem z aplikácie nadobudnutý pomocou simulátora s reálnym chodom. Testovali sa tak hlavne akcie UI, ktoré sú vykonávané pomocou diaľkového ovládača. Pri ich testovaní sa dbalo primárne na ich spoľahlivosť a rýchlosť odozvy.

5.4 Schválenie AppStorom

Aplikácia bola následne publikovaná a odoslaná do schvaľovacieho procesu obchodu AppStore. Pri prvom pokuse o schválenie bola aplikácia odmietnutá. Dôvodom neboli problémy s fungovaním samotnej aplikácie, ale použitie ochrannej známky Twitteru v ikonke aplikácie a priame použitie slov `Twitter`, či `Apple TV` alebo `tvOS` v názve aplikácie. Prvá verzia schvaľovaním neprešla, lebo by aplikácia mohla byť spájaná priamo so spoločnosťou Twitter alebo Apple. Preto sa aplikácia premenovala z `Twitter client` na `Tweetio` a vták symbolizujúci Twitter v ikone aplikácie bol zamenený za inú verziu. Po odstránení týchto pripomienok aplikácia prešla schvaľovacím procesom a je tak verejne dostupná v AppStore [1]

[1] https://itunes.apple.com/us/app/tweetio-the-first-client-to-manage-your-tweets/id1230529415?l=sk&ls=1&mt=8

5.5 Testovanie verejnosťou

Vďaka vydaniu aplikácie do AppStoru mohla byť aplikácia jednoduchšie otestovaná širšou verejnosťou.

Pre otestovanie verejnosťou bol vytvorený prezentačný web aplikácie [2], aby ľudia nadobudli rýchlejšie povedomie o aplikácii a jej funkcionalitách. Následne boli oslovené 2 veľké webové portály, ktoré sa venujú tématike Apple. Tie dali do povedomia prezentačný web a aplikáciu. Jedná sa o článok na webe LetemSvetemApplem [3] a na webe SvetApple [4].

Prezentačný web obsahuje dotazník na získanie spätnej väzby. Tabuľka č. 5.1 zobrazuje jeho výsledky.

Meno	**Vzhľad**	**Intuitívnosť**	**Funkčnosť**
Dávid Salcer	7/10	8/10	7/10
Steve	10/10	10/10	8/10
Stano	8/10	8/10	7/10
Adam	9/10	9/10	8/10
Arnold	10/10	8/10	9/10
Vladimír Solc	10/10	9/10	8/10

Tabuľka 5.1: Výsledky testovania verejnosťou

Vo formulári mohli ľudia vyjadriť aj svoj názor, čo by zmenili v aplikácii. Medzi najvýznamnejšie postrehy patrí zobrazenie obsahu v priloženom URL linku tweetu. Táto funkcionalita by sa v aplikácii isto zišla, no absencia webového prehliadača to neumožňuje. Ďalším postrehom na zmenu bolo napríklad zmena farby vyznačeného tweetu.

[2] http://tweetio.000webhostapp.com/

[3] https://www.letemsvetemapplem.eu/2017/06/12/pomozte-ceskemu-studentovi-s-vyvojem-twitter-klienta-pro-apple-tv/

[4] http://svetapple.sk/zariadenia/iphone-ipad/tweetio-twitter-klient-apple-tv/

Kapitola 6

Záver

Cieľom tejto práce bolo vytvoriť prehliadač užívateľského účtu sociálnej siete Twitter pre AppleTV, čo je moderné zariadenie špecializované pre výstup na obrazovku televízora. Táto klientska aplikácia umožňuje plnohodnotné prezeranie užívateľského obsahu Twitteru a používanie základných akcií ako Like alebo Retweet. Bola implementovaná v jednom z najmodernejších programovacích jazykov Swift 3, ktorý zabezpečuje rýchlosť, interaktívnosť a bezpečnosť vykonávaného kódu.

Aplikácia spĺňa všetky požiadavky zadania. Umožňuje užívateľovi prezerať obsah jeho časovej osi, používať akcie ako retweet či like. Zadanie bolo rozšírené o zobrazovanie najnovšej diskusie k danému tweetu a o zobrazovanie audiovizuálneho obsahu tweetu. Okrem toho boli v aplikácii implementované možnosti na prezeranie časovej osi užívateľových tweetov a retweetov a na prezerania štatistík o profile.

Aplikácia je svižná a spoľahlivá. Napriek tomu má možnosti na ďalšie rozvíjanie. Pri analýze konkurenčných aplikácii som objavil prvky, ktoré v aplikácii chýbajú a zvýšili by tak jej mieru konkurenčnosti. Jedným z týchto prvkov, ktoré by som chcel do aplikácie v budúcnosti doplniť je možnosť vyhľadávania query dotazov, hlavne hashtagov v tweetoch. Ďalšími funkcionalitami, ktoré by aplikácia mohla do budúcna mať, by mohlo byť prezeranie zoznamu sledovaných užívateľov či zobrazenie časových osí iných užívateľov. Vývoj tejto aplikácie nekončí a plánujem tieto návrhy realizovať.

Literatúra

[1] Alamofire: *Alamofire*. [Online; navštívené 28.04.2017].
URL https://github.com/Alamofire/Alamofire

[2] Apple: *About Search on your Apple TV (4th generation)*. [Online; navštívené 03.05.2017].
URL https://support.apple.com/sk-sk/ht205321

[3] Apple: *About TVML*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/ATV_Template_Guide/

[4] Apple: *Apple TV and tvOS*. [Online; navštívené 06.04.2017].
URL https://developer.apple.com/library/content/documentation/General/Conceptual/AppleTV_PG/index.html#//apple_ref/doc/uid/TP40015241-CH12-SW1

[5] Apple: *Apple TV Markup Language Reference- Templates*. [Online; navštívené 06.04.2017].
URL https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/ATV_Template_Guide/TextboxTemplate.html#//apple_ref/doc/uid/TP40015064-CH2-SW8

[6] Apple: *Controlling the User Interface with the Apple TV Remote*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/library/content/documentation/General/Conceptual/AppleTV_PG/WorkingwiththeAppleTVRemote.html

[7] Apple: *Detecting Gestures and Button Presses*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/library/content/documentation/General/Conceptual/AppleTV_PG/DetectingButtonPressesandGestures.html

[8] Apple: *Dispatch Queues*. [Online; navštívené 19.04.2017].
URL https://developer.apple.com/library/content/documentation/General/Conceptual/ConcurrencyProgrammingGuide/OperationQueues/OperationQueues.html#//apple_ref/doc/uid/TP40008091-CH102-SW1

[9] Apple: *Focus Updates*. [Online; navštívené 06.04.2017].
URL https://developer.apple.com/library/content/documentation/General/Conceptual/AppleTV_PG/WorkingwiththeAppleTVRemote.html#//apple_ref/doc/uid/TP40015241-CH5-SW14

[10] Apple: *iCloud Storage*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/library/content/documentation/General/Conceptual/AppleTV_PG/iCloudStorage.html

[11] Apple: *iOS Security Guid*. [Online; navštívené 29.04.2017].
URL https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[12] Apple: *NSURLSession*. [Online; navštívené 28.04.2017].
URL https://developer.apple.com/reference/foundation/nsurlsession

[13] Apple: *Operation*. [Online; navštívené 28.04.2017].
URL https://developer.apple.com/reference/foundation/operation

[14] Apple: *Quality of Service Classes*. [Online; navštívené 19.04.2017].
URL https://developer.apple.com/library/content/documentation/Performance/Conceptual/EnergyGuide-iOS/PrioritizeWorkWithQoS.html

[15] Apple: *Remote and Controllers*. [Online; navštívené 06.04.2017].
URL https://developer.apple.com/tvos/human-interface-guidelines/remote-and-controllers/

[16] Apple: *Supported Capabilities*. [Online; navštívené 06.04.2017].
URL https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/SupportedCapabilities/SupportedCapabilities.html#//apple_ref/doc/uid/TP40012582-CH38

[17] Apple: *Supporting App Transport Security*. [Online; navštívené 02.05.2017].
URL https://developer.apple.com/news/?id=12212016b

[18] Apple: *Swift*. [Online; navštívené 18.04.2017].
URL https://developer.apple.com/swift/

[19] Apple: *Top Shelf Images*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/tvos/human-interface-guidelines/icons-and-images/#top-shelf-images

[20] Apple: *TVMLKit*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/reference/tvmlkit

[21] Apple: *TVMLKit JS*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/reference/tvmljs

[22] Apple: *tvOS 9.2 to tvOS 10.0 API Diffs*. [Online; navštívené 06.04.2017].
URL https://developer.apple.com/library/content/releasenotes/General/tvOS10APIDiffs/index.html

[23] Apple: *TVServices*. [Online; navštívené 03.05.2017].
URL https://developer.apple.com/reference/tvservices

[24] Apple: *XCode*. [Online; navštívené 18.04.2017].
URL https://developer.apple.com/xcode/

[25] CocoaPods: *CocoaPods- About*. [Online; navštívené 21.04.2017].
URL https://cocoapods.org/about

[26] Fabric: *Fabric Docs*. [Online; navštívené 13.04.2017].
URL https://docs.fabric.io/#

[27] Fabric: *Getting Started*. [Online; navštívené 04.05.2017].
URL https://docs.fabric.io/apple/fabric/overview.html

[28] Forums, A. D.: *tvOS: Temporary storage?* [Online; navštívené 03.05.2017].
URL https://forums.developer.apple.com/thread/19002

[29] Group, H. D. I.: *Object Mapper*. [Online; navštívené 28.04.2017].
URL https://github.com/Hearst-DD/ObjectMapper

[30] Hammer-Lahav, E.: *The OAuth 1.0 Protocol*. [Online; navštívené 14.04.2017].
URL https://tools.ietf.org/html/rfc5849

[31] Hardt, D.: *The OAuth 2.0 Authorization Framework*. [Online; navštívené 14.04.2017].
URL https://tools.ietf.org/html/rfc6749

[32] Ibañez, A.: *Are You Storing Sensitive Data in NSUserDefaults? Stop Doing That!* [Online; navštívené 29.04.2017].
URL https://www.andyibanez.com/nsuserdefaults-not-for-sensitive-data/

[33] Krawczyk, H.; Bellare, M.; Canetti, R.: *HMAC: Keyed-Hashing for Message Authentication*. [Online; navštívené 02.05.2017].
URL http://cseweb.ucsd.edu/~mihir/papers/rfc2104.txt

[34] Mills, A.: *Reachability.swift*. [Online; navštívené 21.04.2017].
URL https://github.com/ashleymills/Reachability.swift

[35] MoPub: *MoPub Documentation*. [Online; navštívené 13.04.2017].
URL https://www.mopub.com/resources/docs/

[36] OAuth: *OAuth community site*. [Online; navštívené 14.04.2017].
URL https://oauth.net/

[37] Olesch, D.: *Comparision of iOS and tvOS frameworks*. [Online; navštívené 06.04.2017].
URL https://twitter.com/DavidOlesch/status/656152648433799168

[38] Statista: *Number of monthly active Twitter users worldwide from 1st quarter 2010 to 1st quarter 2017 (in millions)*. [Online; navštívené 09.06.2017].
URL https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[39] Twitter: *Account Activity API (beta)*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/webhooks/account-activity

[40] Twitter: *API Rate Limits*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/rest/public/rate-limiting

[41] Twitter: *Application Permission Model*. [Online; navštívené 03.05.2017].
URL https://dev.twitter.com/oauth/overview/application-permission-model

[42] Twitter: *Authorizing a request*. [Online; navštívené 02.05.2017].
URL https://dev.twitter.com/oauth/overview/authorizing-requests

[43] Twitter: *Creating a signature*. [Online; navštívené 02.05.2017].
URL https://dev.twitter.com/oauth/overview/creating-signatures

[44] Twitter: *Entities*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/overview/api/entities

[45] Twitter: *Implementing Sign in with Twitter*. [Online; navštívené 02.05.2017].
URL https://dev.twitter.com/web/sign-in/implementing

[46] Twitter: *OAuth*. [Online; navštívené 10.04.2017].
URL https://dev.twitter.com/overview/api

[47] Twitter: *OAuth*. [Online; navštívené 09.04.2017].
URL https://dev.twitter.com/oauth

[48] Twitter: *OAuth overview*. [Online; navštívené 14.04.2017].
URL https://dev.twitter.com/oauth/overview

[49] Twitter: *Percent encoding parameters*. [Online; navštívené 02.05.2017].
URL https://dev.twitter.com/oauth/overview/percent-encoding-parameters

[50] Twitter: *PIN-based authorization*. [Online; navštívené 02.05.2017].
URL https://dev.twitter.com/oauth/pin-based

[51] Twitter: *Places*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/overview/api/places

[52] Twitter: *Profile Images and Banners*. [Online; navštívené 28.04.2017].
URL https://dev.twitter.com/basics/user-profile-images-and-banners

[53] Twitter: *Rate Limits: Chart*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/rest/public/rate-limits

[54] Twitter: *REST API- Reference Documentation*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/rest/reference

[55] Twitter: *REST APIs*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/rest/public

[56] Twitter: *Streaming APIs*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/streaming/overview

[57] Twitter: *Tweets*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/overview/api/tweets

[58] Twitter: *Twitter Ads API*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/ads

[59] Twitter: *Twitter Apps*. [Online; navštívené 19.04.2017].
URL https://apps.twitter.com/

[60] Twitter: *Twitter Developer Documentation*. [Online; navštívené 10.04.2017].
URL https://dev.twitter.com/docs

[61] Twitter: *Users*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/overview/api/users

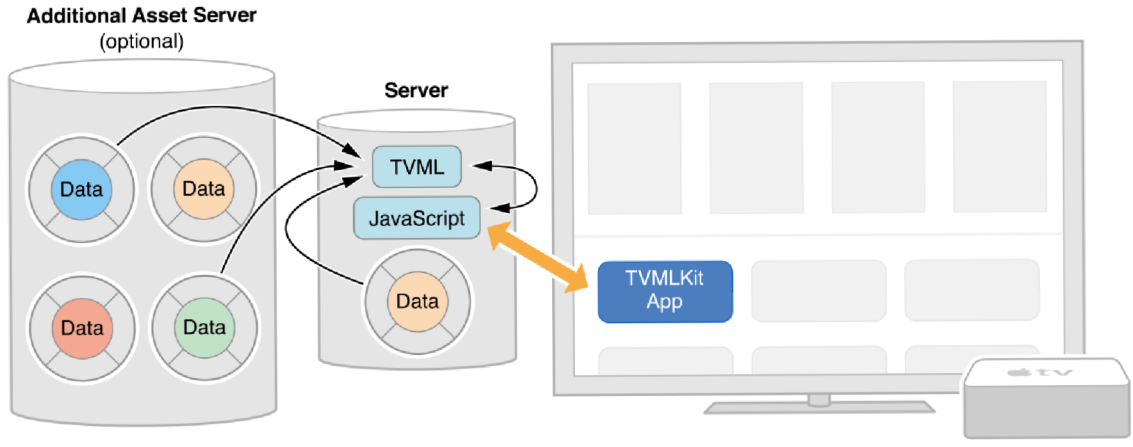
[62] Twitter: *Webhook APIs*. [Online; navštívené 13.04.2017].
URL https://dev.twitter.com/webhooks

[63] ČTK: *Twitter v ČR stále zaostává za Facebookem, uživatelů ale přibývá*. [Online; navštívené 09.06.2017].
URL http://mediahub.cz/media/900591-twitter-v-cr-stale-zaostava-za-facebookem-uzivatelu-ale-pribyva

Prílohy

Príloha A

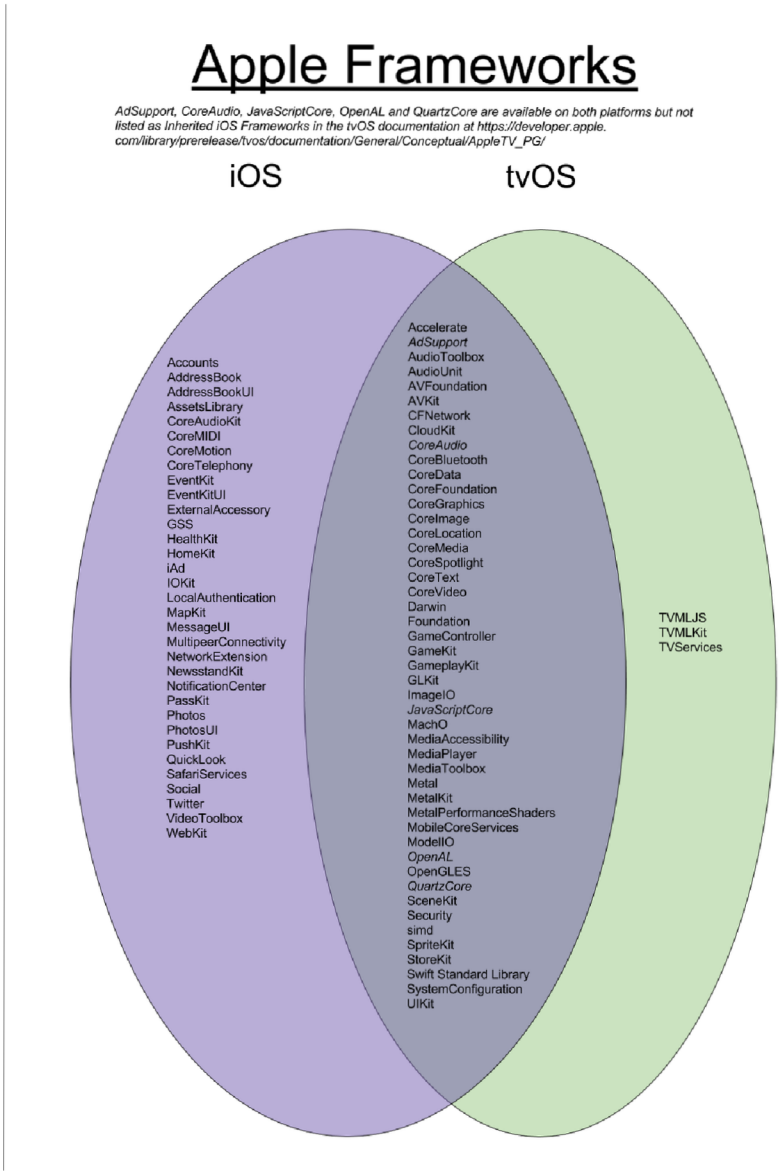
Znázornenie TVML konceptu



Obr. A.1: TVML koncept

Príloha B

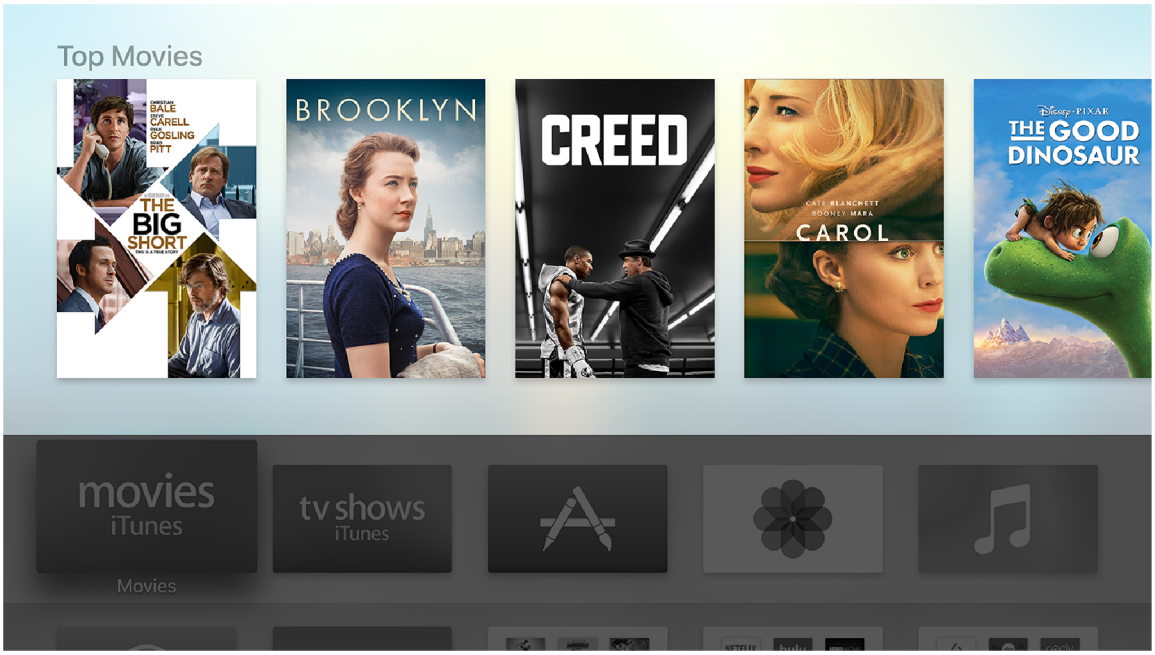
Porovnanie frameworkov



Obr. B.1: Porovnanie frameworkov tvOS a iOS podľa Davida Olescha

Príloha C

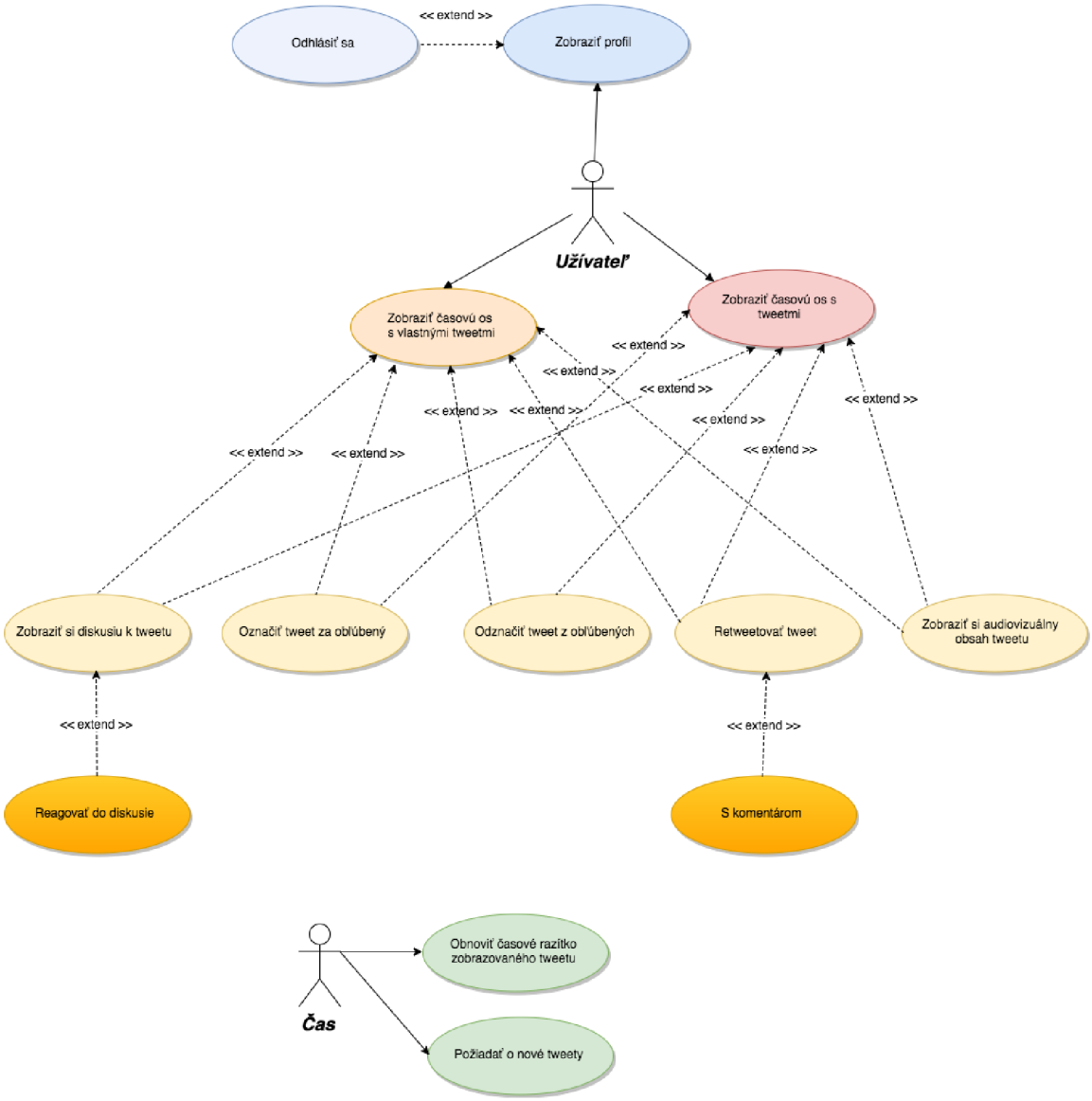
Top Shelf



Obr. C.1: Vo vrchnej polovici sa zobrazí príslušný obsah aplikácie

Príloha D

Diagram prípadov použitia

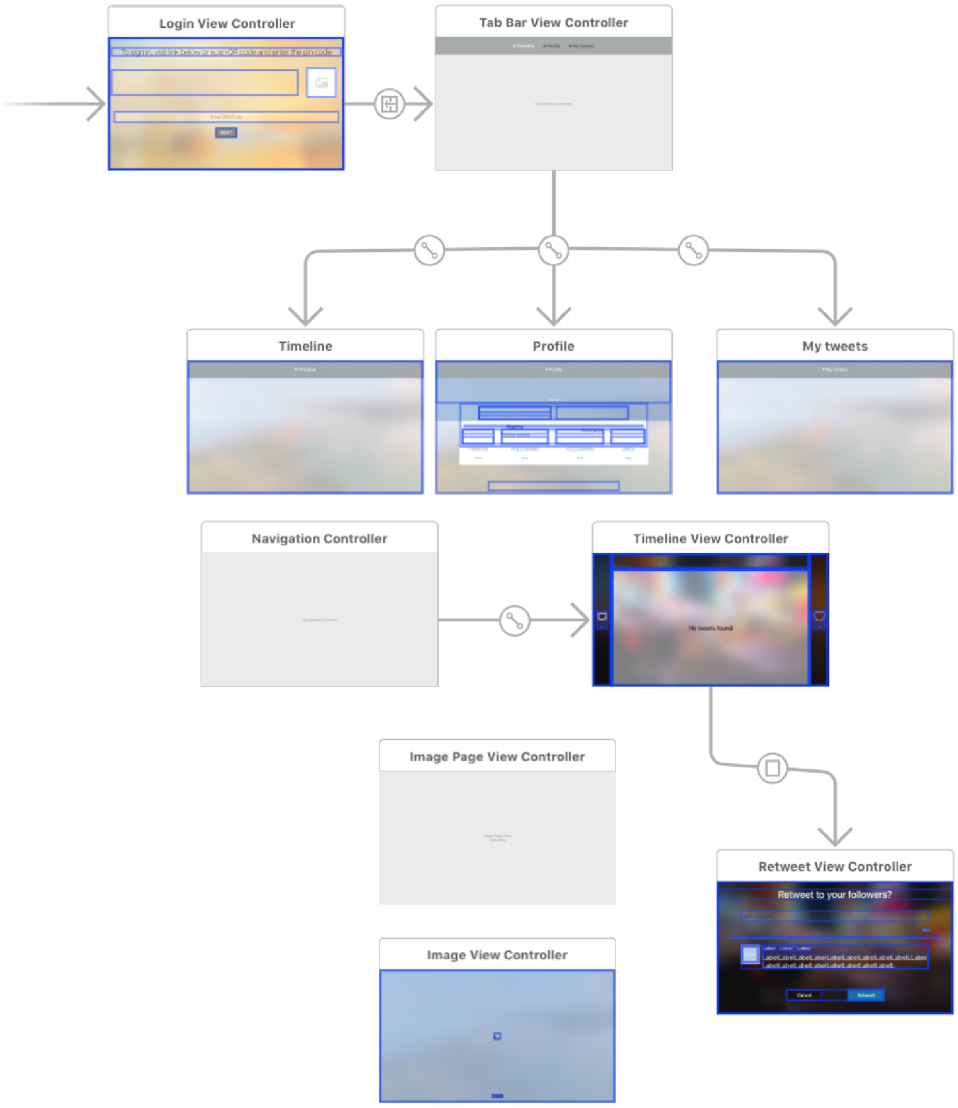


Obr. D.1: Diagram prípadov použitia

Príloha E

Implementačné detaily

E.1 Main.storyboard



Obr. E.1: Zobrazenie náhľadu na Main.storyboard

E.2 Podfile

```
source 'https://github.com/CocoaPods/Specs.git'
platform :tvos, '10.0'
use\_frameworks!

target 'ISP' do
  pod 'KeychainAccess'
  pod 'Alamofire'
  pod 'ObjectMapper'
  pod 'ReachabilitySwift', '~> 3'
  pod 'Fabric'
  pod 'Crashlytics'
end
```