

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Riešenie problémov sieťovej komunikácie v
mobilných aplikáciách**

Bakalárska práca

2018

Lukáš Vavrek

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Riešenie problémov sieťovej komunikácie v
mobilných aplikáciách**

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 9.2.1. Informatika
Školiace pracovisko: Katedra počítačov a informatiky (KPI)
Školiteľ: Ing. Milan Jančár
Konzultant: Mgr. Marcel Majerník
Ing. Peter Stavný

Košice 2018

Lukáš Vavrek

Názov práce: Riešenie problémov sieťovej komunikácie v mobilných aplikáciách

Pracovisko: Katedra počítačov a informatiky, Technická univerzita v Košiciach

Autor: Lukáš Vavrek

Školiteľ: Ing. Milan Jančár

Konzultant: Mgr. Marcel Majerník

Ing. Peter Stavný

Dátum: 22. 5. 2018

Kľúčové slová: sieťová komunikácia, sieťová knižnica, programovanie, mobilné aplikácie

Abstrakt: Táto práca sa venuje najčastejším problémom, s ktorými sa opakovane stretávajú programátori pri implementácii sieťovej komunikácie. Existujúce systémové riešenia a riešenia tretích strán neponúkajú uspokojujúce výsledky pri detekcii sieťových zmien a podpore autentifikácie a autorizácie. Dôvodom je nízka úroveň abstrakcie, na ktorej tieto štandardné knižnice pracujú. Preto sa táto práca venuje návrhu vysokoúrovňovej modulárnej sieťovej knižnice GLNetworking, ktorá tieto problémy eliminuje. Knižnica automatizuje obsluhu autentifikačnej a autorizačnej rutiny a spracovávanie chybových stavov. Obsahuje pokročilú detekciu zmeny sieťového pripojenia, monitorovanie konektivity na internet a dostupnosti aplikačného servera. Použitie knižnice je demonštrované aplikáciou, ktorej úlohou je prezentovať funkcionality a prácu s knižnicou.

Thesis title: Handling Network Communication Problems in Mobile Applications

Department: Department of Computers and Informatics, Technical University of Košice

Author: Lukáš Vavrek

Supervisor: Ing. Milan Jančár

Tutor: Mgr. Marcel Majerník
Ing. Peter Stavný

Date: 22. 5. 2018

Keywords: network communication, networking library, programming, mobile applications

Abstract: This thesis deals with problems, which the developers encounter repeatedly, when implementing a network communication. Available solutions provide unsatisfying results on network changes detection as well as the support of authentication and authorization processes, because they are implemented at the lower level of abstraction. This thesis introduces a high-level modular networking library called GLNetworking. It eliminates above mentioned problems by providing automatization of the authentication and authorization routines, and the error handling. It contains advanced network change detection, which monitors network interface, internet availability and backend server accessibility. Functionalities and usage of the library are illustrated by the demo application.

ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

Riešenie problémov sieťovej komunikácie v mobilných aplikáciách
Handling Network Communication Problems in Mobile Applications

Študent: **Lukáš Vavrek**

Školiteľ: **Ing. Milan Jančár**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce:

Pracovisko konzultanta:

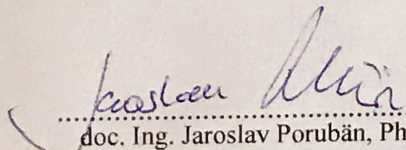
Pokyny na vypracovanie bakalárskej práce:

1. Analyzovať tri často sa opakujúce druhy problémov pri sieťovej komunikácii v mobilných aplikáciách: (1) detegovanie pripojenia na internet, (2) detegovanie a spracovanie chybových návratových kódov s možnosťou opakovania žiadostí a (3) automatická autorizačná a autentifikačná rutina pre opätovné preposielanie požiadavky na server; a taktiež analyzovať existujúce riešenia, ktoré sa dotýkajú tejto problematiky.
2. Navrhnuť knižnicu s modulárnou architektúrou, ktorá by riešila menovaný problém.
3. Implementovať navrhnuté riešenie na platforme iOS.
4. Vyhodnotiť riešenie pomocou testovacej klientskej aplikácie.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

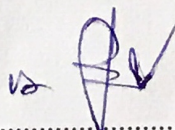
Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 25.05.2018

Dátum zadania bakalárskej práce: 31.10.2017


.....
doc. Ing. Jaroslav Porubän, PhD.
vedúci garantujúceho pracoviska




.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 22.5.2018

.....

Vlastnoručný podpis

Podakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce za jeho čas a odborné vedenie počas písania mojej záverečnej práce.

Rovnako by som rád poďakoval konzultantom za ich rady, nápady a odbornú pomoc.

Na záver, moje najväčšie poďakovanie patrí rodine za ich podporu.

Obsah

Úvod	1
1 Formulácia úlohy	3
1.1 Upresnenie zadania	3
1.2 Čiastkové úlohy a celkový postup	4
2 Analýza problému	5
2.1 Detekcia konektivity na internet a spojenia so serverom	5
2.2 Automatické obsluhovanie autentifikačnej a autorizačnej rutiny . .	6
2.3 Automatická správa chybových stavov	7
3 Analýza súčasného stavu	8
3.1 Voľne dostupné knižnice	8
3.2 API poskytované operačným systémom	10
3.3 Vyhodnotenie analýzy	11
4 Architektúra riešenia	13
4.1 Sieťová komunikácia, sieťové služby	13
4.2 Komunikácia klient - modul	14
4.3 Vytváranie sieťovej požiadavky	17
4.4 Dokumentácia knižnice	18
5 Návrh riešenia - Knižnica GLNetworking	20
5.1 Sieťový manažér (NetworkManager)	21
5.2 Modul dostupnosti (Reachability)	22
5.3 Modul spracovania sieťových odpovedí (Network Response Handler)	24
5.4 Autentifikačný a autorizačný modul (Auth)	25

5.5	Výhody navrhnutého riešenia	26
6	Vyhodnotenie	27
6.1	Testovacia klientská aplikácia	28
6.2	Zhrnutie	31
7	Záver	32
	Literatúra	34
	Zoznam pojmov	36
	Zoznam príloh	39

Zoznam obrázkov

4.1	Rola API v architektúre klient - server	13
4.2	Diagram tried návrhu komunikácie medzi klientom a modulom .	15
4.3	Diagram tried návrhu komunikácie medzi klientom a modulom .	16
4.4	Diagram tried návrhu komunikácie medzi klientom a modulom .	16
4.5	Diagram tried návrhu komunikácie medzi klientom a modulom .	17
5.1	Návrh architektúry knižnice GLNetworking	21
5.2	Diagram aktivity spracovávania odpovede na sieťovú požiadavku	25
6.1	Používateľské rozhranie testovacej aplikácie	29

Úvod

Život ľudí 21. storočia sa točí okolo počítačov. V priebehu posledných 10 rokov sa rozšírili prakticky do všetkých oblastí a stali sa samozrejmom súčasťou nášho života. Obrovskej popularite sa medzi ľuďmi tešia inteligentné telefóny (smartfóny), ktoré spôsobili revolúciu na trhu s mobilnými telefonmi, rovnako ako ich väčší súrodenci - tablety.

Dnes už nie je žiadnym prekvapením, že sa tieto zariadenia výkonnostne dotiahli na úroveň laptopov, či stolových počítačov. Ich najväčšou a nepopierateľnou výhodou je to, že ich môžeme nosiť stále so sebou. Práve tento fakt otvoril nové možnosti využitia a postupne tak vznikla platforma s veľkým potenciálom. Podľa štatistík [1] sa v roku 2017 nachádza vo svete 2,32 miliárd používateľov smartfónov a medziročne sa predpokladá ich nárast o viac ako 8%.

Takto obrovský segment priťahuje aj vývojárov, čo zvyšuje konkurenciu, voči ktorej je veľmi náročné sa presadiť. Je preto potrebné prísť s aplikáciou, ktorá bude nielen originálna, ale taktiež spĺňa určité štandardy a zaistí takú úroveň a kvalitu, na ktorú sú dnešní, čoraz náročnejší, používatelia zvyknutí.

Ďalšou možnosťou je vyvinúť aplikáciu, ktorá prináša riešenie na problémy, ktoré trápia bežných užívateľov. Ja som sa rozhodol ísť touto cestou a zamerať sa na oblasť, ktorej využívanie rastie geometricky. Medzi užívateľmi sa čoraz väčšej obľube teší využívanie mobilného internetu, k čomu dopomohli sociálne siete či cloudové úložiská. Pri využívaní internetu je potrebné využívať komunikáciu so serverovou časťou (tzv. aplikačným backend-om). Aby bol komfort používateľa čo najlepší, s čím súvisí plynulosť aplikácie, je nutné zabezpečiť nasledovné:

- prechody medzi online a offline stavom,
- spracovanie chybových sieťových a serverových hlášok,
- autentifikáciu a autorizáciu.

Existuje viacero možností, ako dané problémy riešiť.

Pri hlbšej analýze problému a po konzultáciách s odborníkmi na danú oblasť (viď kapitolu 2) môžem jednoznačne tvrdiť, že tento problém je komplexný, vyžadujúci si robustné riešenie, avšak také, ktoré bude môcť byť využité a modifikované podľa konkrétnych potrieb programátora.

Podľa dostupných informácií som zistil, že sa doposiaľ nenaimplementovalo všeobecné riešenie. Skôr sa vyvíjali riešenia, ktoré pokrývali veľmi špecifické oblasti pre danú aplikáciu. Tieto neúplné riešenia šité na mieru, zvyčajne obsahujú minimum takej funkcionality, ktorá by mohla byť aplikovaná aj pre iné riešenia. To zabraňuje znovu-použitelnosti kódu, čo predstavuje zbytočné neefektívne využitie času pri vývoji podobnej funkcionality. Namiesto nej sa vývojári mohli venovať programovaniu aplikačného kódu a novej funkcionality.

Z vyššie uvedených dôvodov som sa rozhodol tomuto komplexnému problému venovať vo svojej práci. Sústreďujem sa na problematiku sieťovej komunikácie v mobilných aplikáciách. Znalosť špecifických problémov a ich riešení chcem uplatniť pri vývoji programovej knižnice, ktorá zapuzdrí tieto riešenia a pre konkrétne potreby aplikácií bude poskytovať jednoduché možnosti konfigurácie na mieru.

1 Formulácia úlohy

V tejto práci sa zameriavam na rôzne problematické časti sieťových riešení (networking), najmä pri implementovaní komunikácie medzi serverom a klientom v mobilných aplikáciách.

Úvodnou, dôležitou časťou je investigácia a identifikácia problémov, ktoré sa opakujú v mnohých softvérových projektoch a preto ich nemá zmysel nanovo implementovať.

Hlavným cieľom mojej práce je vyvinúť a naprogramovať všeobecnú knižnicu (framework), ktorá problémy popísané v tejto práci rieši a zapuzdrí ich v komplexný celok. Ten sa bude môcť jednoducho použiť pri rôznych typoch mobilných aplikácií, vyžadujúcich komunikáciu medzi serverom a klientom. Zároveň týmto vznikne otvorený systém, ktorý bude plne modifikovateľný pre konkrétne potreby programátorov.

1.1 Upresnenie zadania

Problémy a potreby, na ktoré poukazujem v tejto práci, sa vyskytujú naprieč všetkými modernými mobilnými platformami. Vzhľadom na to, že mám skúsenosti s programovaním iOS aplikácií, rozhodol som sa v tejto práci venovať práve tejto platforme.

Pre vývoj knižnice som zvolil programovací jazyk Swift. Swift [2], ešte stále mladý programovací jazyk, si medzi vývojármi našiel rýchlo svojich priaznivcov. Má expresívnu syntax, podporujúcu písanie čitateľného a bezpečného kódu. Je robustný, intuitívny a obsahuje modernú funkcionálnosť. V rýchlostnom porovnaní [3] je 2,6 krát rýchlejší než Objective-C a 6,8 krát rýchlejší než Python 2.7.

1.2 Čiastkové úlohy a celkový postup

Na základe praxe sa zistili tri často sa opakujúce druhy problémov pri implementovaní sieťovej komunikácie (networking):

1. detekcia spojenia na internet, ku ktorému dochádza za Wi-Fi prístupovým bodom (access point),
2. detekcia a jej následné spracovanie chybových návratových kódov s možnosťou opakovania žiadostí (requests),
3. automatická autorizačná a autentifikačná rutina (authorization and authentication routine) pre opätovné posielanie požiadavky na server (client request to server).

V kapitole 2 sa zameriavam na podrobnejšie vysvetlenie týchto problémov pre ich lepšie pochopenie. Ukazujem ich dôležitosť pre programátorov pri vývoji mobilných aplikácií.

Následne sa v kapitole 3 venujem prieskumu dostupných knižníc a to najmä na analýzu ich ponúkanej funkcionality v kontexte nastoleného problému.

V kapitole 4 popisujem rozhodnutia, ktoré bolo potrebné urobiť pri navrhovaní celkovej architektúry knižnice. Venujem sa rozboru sieťovej komunikácie (kapitola 4.1), analyzujem použitie vhodných návrhových vzorov v knižnici (kapitola 4.2) a zameriavam sa na návrh vonkajšieho rozhrania, s ktorým bude pracovať aplikácia využívajúca túto knižnicu (kapitola 4.3).

Návrhu knižnice, ktorá zapuzdrí riešenia na popisované problémy do modúlárneho celku, sa venujem v kapitole 5. Popisujem celkovú architektúru a hlavné moduly, ktoré tvoria jej jadro.

V záverečnej kapitole 6 sa zameriavam na vyhodnotenie navrhnutého riešenia (knižnice) pomocou analýzy kódu a funkcionality testovacej klientskej aplikácie.

2 Analýza problému

Pri skúmaní najčastejších problémov týkajúcich sa sieťovej komunikácie v mobilných aplikáciách som vychádzal nielen z vlastného prieskumu, ale v mnohom mi pomohli diskusie s ľuďmi, ktorí majú bohaté skúsenosti z praxe.

Po zhodnotení mojich zistení a poznatkov som identifikoval tri základné problémy, ktorých zvládnutie je kľúčové pre zabezpečenie korektného správania sa aplikácie, ktorá využíva komunikáciu so serverom.

Sú to:

1. detekcia konektivity na internet a spojenia so serverom,
2. automatizované obsluhovanie autentifikačnej a autorizačnej rutiny,
3. automatická správa chybových stavov.

2.1 Detekcia konektivity na internet a spojenia so serverom

Detekcia konektivity na internet a spojenia so serverom je základom pre všetky ďalšie „nastavbové“ funkcionality. Programátor aplikačnej vrstvy potrebuje mať neustále informáciu o tom, či môže komunikovať so vzdialeným API a v akej sieti je dané mobilné zariadenie pripojené.

Pri poskytnutí dostatočného počtu informácií o kvalite sieťového pripojenia môže programátor aplikačnej vrstvy optimalizovať kód tak, aby sa aplikácia adaptovala na zmenu kvality pripojenia. Takéto úpravy zlepšujú používateľský komfort tým, že minimalizujú dobu čakania za prostriedkami, ktoré je nutné načítať z internetu - napr. sťahovanie obrázkov. Pri nekvalitnom, pomalom pripojení (napríklad staršie generácie sietí WWAN), je vhodné zvoliť nižšiu kvalitu (veľkosť)

sťahovaného obrázku, pretože „cena“ dát je podstatne vyššia a rýchlosť pripojenia zvyčajne nižšia ako pri pripojeniach prostredníctvom WiFi.

Ďalším príkladom je stabilita pripojenia. Vzhľadom na to, že sú používatelia (a teda aj ich mobilné zariadenia) neustále v pohybe, dochádza k výpadkom signálu mobilnej siete alebo pripojenia na WiFi. Stáva sa tiež, že aj napriek stabilnému pripojeniu do siete, nie je dostupný internet. Problémy s pripojením a dostupnosťou na internet sa môžu týkať aj samotného servera, preto je nutné počítať aj s takýmito výpadkami. Aplikácia musí už v návrhu počítať s takouto nestabilitou pripojenia. Preto je potrebné tento stav neustále monitorovať a zabezpečiť okamžitú reakciu na zmeny tak, aby sa minimalizoval dopad na stabilitu aplikácie.

2.2 Automatické obsluhovanie autentifikačnej a autorizáčnej rutiny

Súčasťou modernej mobilnej aplikácie by malo byť aj automatické obsluhovanie autentifikačnej a autorizáčnej rutiny [4]. Pod pojmom autentifikácia rozumieme mechanizmus identifikácie a overenia identity. Pojem autorizácia znamená overovanie oprávnenia na prístup či vykonanie akcie.

Aplikácia potrebuje dynamicky reagovať na potrebu autentifikácie, ktorá sa môže vyskytnúť pri ľubovoľnej požiadavke na server.

Z používateľského hľadiska je dôležité vyriešiť túto situáciu tak, aby sa ňou neovplyvnil používateľský tok vzhľadom na pôvodnú požiadavku. Užívateľ tak bude môcť pokračovať v činnosti bez toho, aby musel opätovne zadávať prihlasovacie údaje. Okrem toho je veľmi dôležité je vyvarovať sa narušeniu používateľského toku automatickým odhlásením užívateľa z aplikácie. Takéto prerušenie činnosti môže viesť k nespokojnosti používateľov a celkovému neúspechu aplikácie.

Z programového hľadiska je vhodné, ak bola táto autentifikačná a autorizáčná rutina poskytovaná priamo sieťovou knižnicou. Okrem zjednodušenia a sprehládnenia aplikačného kódu sa tým zabezpečí aj korektná implementácia pre každú odchádzajúcu požiadavku a to bez potreby duplikovania kódu.

2.3 Automatická správa chybových stavov

Poslednou problematickou časťou je detekcia a spracovanie chybových stavov z odpovedí servera. Jedná sa o štandardné stavové kódy protokolu HTTP [5] typu 4xx (chyba na strane klienta) a 5xx (chyba na strane servera). Vzhľadom na to, že potreby každej aplikácie sú rôzne, nie je možné riešenie tohto problému úplne zovšeobecniť. Je preto mojím cieľom nastaviť všeobecné základy a ponechať možnosť konfigurácie na mieru samotným vývojárom.

3 Analýza súčasného stavu

Problém, ktorému sa venujem v tejto práci, je potrebné vyriešiť pri vývoji každej aplikácie, ktorá využíva komunikáciu so serverom. Jeho riešeniu sa venovalo viacero ľudí či tímov, ktoré sú popísané v nasledujúcich častiach tejto kapitoly.

3.1 Voľne dostupné knižnice

Na trhu sa nachádza niekoľko voľne dostupných knižníc ponúkajúce špecifické riešenia na pomenované problémy. Pre platformu iOS medzi najznámejšie a najpoužívanejšie patria:

- AFNetworking,
- Alamofire.

AFNetworking [6] je populárna knižnica pre sieťovú komunikáciu pre platformy iOS a macOS, predstavená v roku 2011. Je napísaná a primárne určená pre použitie s jazykom Objective-C. V prípade hybridných aplikácií, alebo aplikácií písaných v jazyku Swift, je jej použitie možné pomocou tzv. premostovacieho hlavičkového súboru (Objective-C bridging header).

Táto knižnica vznikla ako elegantná a robustná alternatíva k nie veľmi jednoduchému a dnes už zastaranému API v podobe **NSURLConnection** poskytovaného operačným systémom. Programátorom aplikačného kódu umožňovala využívať pokročilé funkcionality, ako napr. zrušenie, pozastavenie a pokračovanie otvorených požiadaviek, monitorovanie stavu sťahovania a nahrávania súboru, či využívať pokročilé cachovanie. V priebehu rokov, sa vďaka napredovaniu systémových knižníc (momentálne **NSURLSession**), tento rozdiel minimalizoval.

Postupom času sa z tejto knižnice stal štandard. AFNetworking obsahuje prídavné moduly, ktoré zjednodušujú proces komunikácie so serverom - od vytvára-

nia sieťových požiadaviek, podpory autentifikácie až po serializáciu a parsovanie dát z odpovede.

Jedným z týchto prídavných modulov je aj **AFNetworkReachabilityManager**. Tento modul slúži na monitorovanie dostupnosti domény, pričom podporuje WWAN a WiFi sieťové rozhrania. Zameriava sa na riešenie problému detekcie konektivity na internet (viď kapitola 2.1). Použitie tohoto modulu je odporúčané hlavne na zisťovanie, či požiadavka, ktorá skončila neúspešne, má byť znovu opakovaná (napr. ak nabehla sieť). Tento odlišný postoj k spomínanému problému je vzhľadom na navrhované riešenie v tejto práci (kapitola 5.2) nedostatočný. Neumožňuje totiž vývojárom spoľahnúť sa na dáta, ktoré ponúka.

AFNetworking si rokmi získala podporu komunity. Avšak, zlepšujúci sa stav knižníc, ktoré poskytuje priamo operačný systém, robia z jej použitia otázku kompromisu. Totiž, rozdiel medzi funkcionalitou, ktorú ponúka je minimálny a zároveň prináša do projektu veľkú závislosť.

Knižnica **Alamofire** [7] sa dá považovať za priamu alternatívu knižnice AFNetworking. Základným rozdielom je fakt, že **Alamofire** je implementovaná v programovacom jazyku Swift, čomu je prispôsobený aj samotný dizajn programového API. Až na niektoré špecifické detaily sú však funkcionálne rovnocenné a odporúča sa vybrať si knižnicu na základe programovacieho jazyka.

Knižnica poskytuje niekoľko zaujímavých funkcií, okrem iného aj reťazenie požiadaviek, sledovanie stavu nahrávania a sťahovania súborov, podporu autentifikácie, kontrolovanie dostupnosti siete a možnosť automatického opakovania požiadaviek.

Posledné dve časti knižnice čiastočne riešia problémy, ktorým sa venuje táto práca.

Kontrolovanie dostupnosti siete (**NetworkReachabilityManager**) rieši čiastočne problém detekcie konektivity na internet a spojenia so serverom (viď kapitola 2). Jeho funkcionalita však nie je úplne postačujúca. Tento modul má primárne informačný charakter a odporúča sa používať na ladenie, prípadne na znovuoslanie neúspešných požiadaviek. Naopak, neodporúča sa využívať na zabránenie odoslania požiadavky, vzhľadom na informáciu o nedostupnosti servera. Samotný modul totiž neiniciuje komunikáciu so serverom a neošetruje tak prípady, kedy práve prvotná požiadavka zabezpečí vytvorenie spojenia a tým pádom aj výsledný kladný stav výsledku testu konektivity.

Funkcionalita možnosti automatického opakovania požiadaviek (**RequestRetrier**) ponúka pomerne robustné, avšak komplikované riešenie. Potrebná funkcionalita sa dosahuje pomocou implementovania protokolu (rozhrania), čím knižnica pre-súva plnú zodpovednosť na aplikačnú vrstvu, ktorá vyhodnocuje, či sa ma daná požiadavka zopakovať. Podľa kapitoly 2 je pre väčšinu prípadov postačujúcim riešením možnosť konfigurácie opakovania požiadaviek na základe stavových kódov protokolu HTTP [5], ktoré definujú výsledok spracovania požiadavky na strane servera.

K riešeniu problémov popísaných v mojej práci pristupujú tieto knižnice odlišne, neúplne a často komplikovane. Ich použitie je teda neuspokojivé. Riešenie problémov nastolených touto prácou by bolo vyvinuté len ako ich nadstavba. Ich funkcionálny prínos je oproti programovým rozhraniám poskytnutých operačným systémom zanedbateľný. Naopak, do projektu prinášajú veľkú závislosť. Preto som sa rozhodol ich z navrhovaného riešenia vynechať (viď kapitola 5).

3.2 API poskytované operačným systémom

Operačný systém iOS ponúka niekoľko API, ktoré môžu byť využité programátormi v rámci implementácie sieťovej komunikácie. Dlhodobý trend používania sieťových knižníc sa postupom času znižuje, nakoľko operačný systém poskytuje čoraz pokročilejšie programové rozhrania. Funkcionálny rozdiel medzi použitím knižníc a natívnych programových rozhraní je v súčasnosti už minimálny.

Pre platformu iOS existujú tri základné vrstvy sieťových knižníc [8] poskytujúcich API na rôznych úrovniach abstrakcie:

1. Vrstva POSIX,
2. Vrstva Core Foundation,
3. Vrstva Foundation.

Vrstva **Foundation** ponúka prístup k API na najvyššej úrovni abstrakcie. Programovanie na nižších vrstvách sa odporúča len pri špeciálnych požiadavkách na API (napríklad pri písaní serverového kódu). Štandardne sa však odporúča použiť čo najvyššiu vrstvu, ktorá poskytuje dostatočnú funkcionalitu.

Vrstvy **POSIX** a **Core Foundation** poskytujú nízko-úrovňový prístup k sieťovej vrstve operačného systému. Vrstva Foundation je jednoduchá na používanie,

vyžaduje najmenšie množstvo kódu na dosiahnutie potrebnej funkcionality a je navrhnutá tak, aby poskytovala dostatočnú funkcionality pre väčšinu klientských aplikácií. Na tejto vrstve sa nachádza súbor tried `URLSession`.

`URLSession` je súbor tried, ktoré spoločne poskytujú vysoko-úrovňové API slúžiace na prístup ku sieťovej komunikácii. V architektúre klient-server sa funkcionality zameriava na podporu implementácie klientskej časti. `URLSession` podporuje protokoly HTTP/1.1, SPDY a HTTP/2, obsahuje tiež mechanizmus pridania podpory pre vlastný protokol. Poskytuje dostatočnú funkcionality na uspokojenie nárokov a požiadaviek väčšiny moderných klientských aplikácií.

Okrem iného podporuje napríklad nahrávanie a sťahovanie súborov zo servera, kontrolu stavu požiadavky (napr. pri sťahovaní súboru), zrušenie aktívnej (prebiehajúcej) požiadavky, prerušenie a pokračovanie požiadavky, s možnosťou pokračovať z bodu prerušenia, podpora cachovacieho mechanizmu.

Ďalším užitočným rozhraním je `SCNetworkReachability` [9], ktoré slúži na zisťovanie stavu aktuálnej systémovej sieťovej konfigurácie a dostupnosti cieľového hostiteľa. Nadstavbou pre tento modul je už spomínaný `AFNetworkReachabilityManager`. `SCNetworkReachability` poskytuje ucelené a detailné informácie o spojení (konektivite). Toto rozhranie považuje cieľového hostiteľa za dostupného vtedy, ak paket s danou cieľovou adresou dokáže opustiť zariadenie. Negaranuje však to, že daný paket bude naozaj doručený. Informácie o dostupnosti preto nemožno považovať za úplné a spoľahlivé, no dajú sa využiť v rámci komplexnejšieho celku a to na zisťovanie internetovej konektivity a spojenia so serverom.

3.3 Vyhodnotenie analýzy

Z analýzy existujúcich riešení vyplýva, že voľne dostupné knižnice a ani programové rozhrania neposkytujú dostatočne uspokojujúce riešenia na problémy, ktorým sa venuje táto práca. Riešenia ktoré ponúkajú, sú iba čiastočné, alebo naopak, až príliš komplexné pre bežné potreby väčšiny aplikácií.

Vzhľadom na vyššie uvedené som presvedčený, že je potrebné prísť s novým a zároveň uceleným riešením. To by malo ponúkať vyššiu úroveň abstrakcie než spomínané alternatívy a malo by byť postavené ako rozšírenie a zapuzdrenie už existujúcej sieťovej funkcionality.

Kvôli minimalizovaniu závislosti som sa rozhodol ako základ použiť API po-

skytované operačným systémom (konkrétne URLSession a SCNetworkReachability, viď kapitola 5).

Ďalším cieľom navrhovaného riešenia je jednoduchosť použitia, ľahká integrácia do projektu a dostatočná flexibilita na konfiguráciu špecifických potrieb konkrétnych aplikácií.

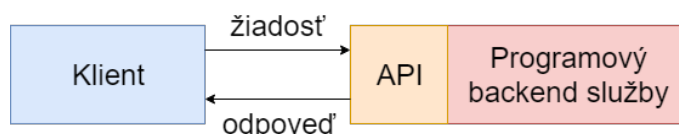
4 Architektúra riešenia

Pri návrhu riešenia knižnice je dôležité rozanalyzovať viaceré architektonické postupy. Pri hľadaní ideálneho riešenia treba brať do úvahy viac možností, pričom každá z nich má svoje pre a proti. Je preto nutné zvážiť všetky argumenty a zvoliť taký prístup, ktorý najlepšie rieši pôvodný zámer.

4.1 Sieťová komunikácia, sieťové služby

Architektúra klient-server [10] je najznámejšia a najpoužívanejšia forma sieťovej architektúry. Túto architektúru pre svoje fungovanie využíva aj Internet (World Wide Web).

Webové služby plnia v tejto architektúre úlohu serverov, ktorých funkciou je zastrešovať klientské potreby aplikácie. Klientský program využíva na komunikáciu s webovou službou aplikačno-programové rozhranie (API). API poskytuje prístup k dátam a funkcionalitám a umožňuje interakciu a výmenu dát medzi klientom a serverom. Rola API je znázornená na obrázku 4.1.



Obr. 4.1: Rola API v architektúre klient - server

Pri návrhu moderných API rozhraní je populárna REST architektúra (z anglického „representational state transfer“). Je to súbor pravidiel a štandardov pre dizajn distribuovanej architektúry. Obsahuje odporúčania pre návrh funkčných a elegantných API (z pohľadu programátora aplikačného kódu využívajúceho dané rozhranie). Takéto rozhrania označujeme ako **REST API**.

REST API [11] je postavené na protokoloch aplikačnej vrstvy (HTTP / HTTPS) a využíva spojové protokoly transportnej vrstvy (zvyčajne TCP). To má niekoľko výhod, okrem iného aj:

- spoľahlivosť (TCP),
- jednoduchosť (HTTP, textový protokol),
- bezpečnosť (HTTPS),
- efektívnosť (cache).

Tieto koncepty sú známe pre väčšinu programátorov, čo znamená jednoduchú integráciu REST služieb do navrhovaných riešení.

Ďalšou výhodou je oddelenie klientskej a serverovej časti. Obe tieto časti sú na sebe nezávislé a jediné čo musia splniť, je dodržanie formálnej štruktúry komunikačného protokolu. Na výbere konkrétneho programovacieho jazyka či technológie, v ktorej budú tieto časti implementované, nezáleží. Zjednodušený popis fungovania je popísaný v ďalších riadkoch.

Klient smeruje požiadavky na serverové zdroje, ktoré sú reprezentované identifikátorom URI (jednotný identifikátor zdroja, z anglického „Uniform Resource Identifier“). Platí, že jednotlivé požiadavky sú bez-stavové (stateless). Tieto URI reprezentujú tzv. CRUD operácie (vytvorenie, čítanie, aktualizácia, mazanie, z anglického Create, Read, Update, Delete). Dátový formát využívaný pri komunikácii môže byť ľubovoľný - zvyčajne sa používa JSON, prípadne XML. Údaj o použitom dátovom formáte sa nachádza v HTTP hlavičke s názvom Content-Type.

API, webový server, aplikačná biznis logika a databázová časť sa zvyknú nazývať tiež ako programový (aplikačný) backend. Pomocou aplikačného backendu je možné prepájať jednotlivých klientov a rozširovať možnosti aplikácie.

4.2 Komunikácia klient - modul

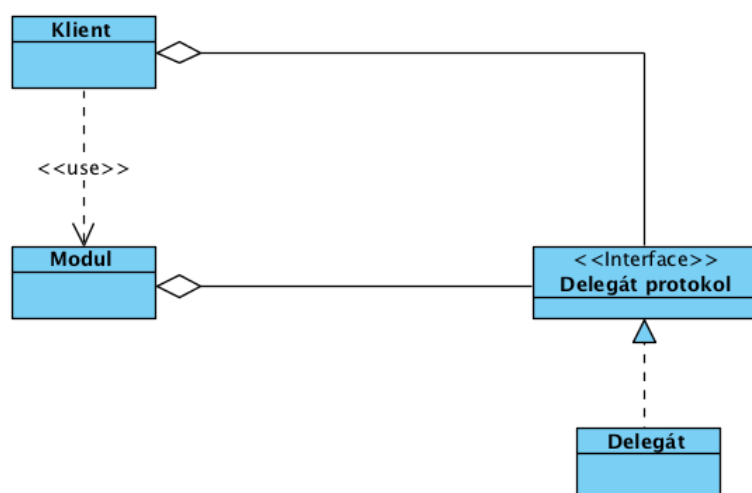
Pre modul (knižnicu), ktorú budeme vyvíjať, potrebujeme zabezpečiť obojsmernú komunikáciu s klientom (aplikáciou). Existuje niekoľko možností ako to dosiahnuť.

Ak by sme chceli túto funkcionálnosť implementovať pomocou obojsmerného prepojenia objektov klienta a modulu, narazili by sme na niekoľko problémov.

Takéto rozhodnutie má výrazný negatívny dopad na samostatnosť jednotlivých tried a komponentov a vytvára medzi nimi silné previazanie. V prípade knižnice je takýto postup dokonca nereálny, vzhľadom na to, že pri jej vývoji je klient neznámy.

Na riešenie tohto problému sa v praxi zvykne využívať princíp delegátov [12]. Pri takomto prístupe je klient schopný volať metódy modulu priamo a poskytnúť mu delegáta, pomocou ktorého môže modul nepriamo komunikovať s klientskou časťou. Delegátom v tomto prípade môže byť nielen klient, ale aj samostatná trieda.

Návrh zobrazený na obrázku 4.2 uvažuje o probléme tak, že klient (konkrétna trieda komunikujúca s modulom) nie je priamo zodpovedný za inšanciovanie konkrétneho delegáta a môže túto zodpovednosť ponechať na niekoho iného, napr. factory (továreň), prípadne využívať jedného zdieľaného delegáta.



Obr. 4.2: Diagram tried návrhu komunikácie medzi klientom a modulom

Diagram navrhnutý na obrázku 4.3 je presnejší, bližšie kopírujúci reálny kód. Klient má priamu zodpovednosť za poskytnutie konkrétneho delegáta, pričom samotný klient môže zastupovať túto rolu. V praxi toto použitie vyzerá nasledovne:

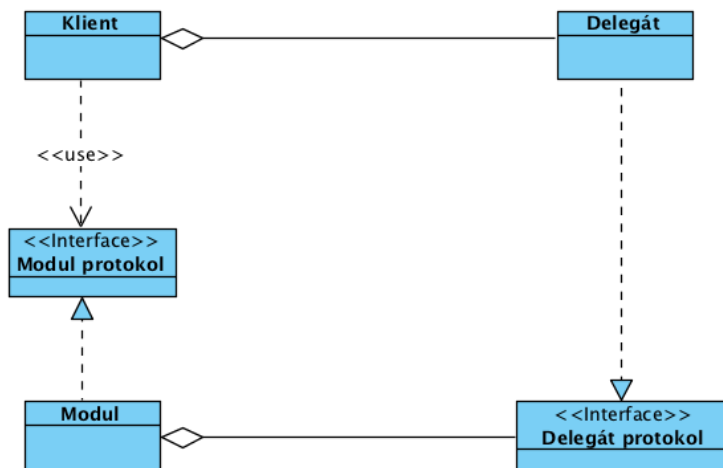
```
self.tableView.delegate = self
```

Diagram na obrázku 4.4 ukazuje modulárnejšie rozšírenie diagramu z obrázku 4.3. Výhodou takéhoto prístupu je to, že „zavesenie“ modulu za rozhranie zlepšuje modularitu a priamo upozorňuje na možnosť rozšírenia modulu, napríklad



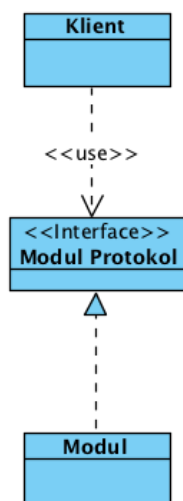
Obr. 4.3: Diagram tried návrhu komunikácie medzi klientom a modulom

pomocou návrhového vzoru decorator (dekorátor). Nevýhodou spomínaného návrhu je to, že modul môže alebo nemusí implementovať rozhranie, keďže vo väčšine prípadov je samotný modul navrhnutý tak, aby sa využíval priamo. Vynechanie implementovania rozhrania nedodržiava princíp „dependency inversion“ [13] („D“ v SOLID [14]). Je preto otázkou kompromisu a záleží či je nutná jednoduchá rozšíriteľnosť tohto modulu.



Obr. 4.4: Diagram tried návrhu komunikácie medzi klientom a modulom

Oveľa jednoduchší prístup je načrtnutý v diagrame 4.5. Problémom takéhoto návrhu je to, že priamo neupozorňuje na možnosť obojsmernej komunikácie. Tú by bolo v takomto prípade nutné implementovať napr. pomocou návrhového vzoru observer, čiže registráciou spätných volaní pre potrebné funkcionality.



Obr. 4.5: Diagram tried návrhu komunikácie medzi klientom a modulom

Pri implementovaní komunikácie medzi klientom a modulom využíva knižnica návrhy, zobrazené na obrázkoch 4.3 a 4.4. Tento spôsob komunikácie podporuje znovu-použiteľnosť jednotlivých častí, znižuje previazanie a zlepšuje celkovú kvalitu kódu. Umožňuje dosiahnuť nezávislosť knižnice tak, že ponecháva aplikačno-špecifické rozhodnutia na delegátovi, ktorého poskytuje aplikačný kód.

V rámci aplikácie sa tiež viac krát vyskytuje použitie tzv. Multicast delegátov. Jedná sa o návrhový vzor observer, ktorý slúži na oboznámenie zmeny stavu (pozorovaného objektu) všetkým, ktorí o to požiadajú (pozorovatelia).

4.3 Vytváranie sieťovej požiadavky

Najčastejšie vyskytujúcou sa časťou knižnice GLNetworking je vytváranie sieťovej požiadavky. Preto bolo mojím cieľom navrhnúť jednoduché verejné API, ktoré bude túto funkcionality vykonávať.

Aby som dosiahol čo najprehľadnejšiu konfiguráciu vytváranej požiadavky, rozhodol som sa využiť návrhový vzor builder (staviteľ) [15]. Tento návrhový vzor umožňuje zrefaziť jednotlivé volania metód, pomocou ktorých sa konfiguruje výsledná požiadavka. Využitie tohto návrhového vzoru má pozitívny vplyv na celkovú čitateľnosť výsledného kódu.

Reálne použitie v klientskej aplikácii môže vyzeráť napríklad takto:

```
GLRequest
```

```

    .create(url: URLConstants.loginURL)!
    .httpMethod("post")
    .jsonBody([
        "username": "name",
        "password": "password"
    ])
    .success(loginSuccessful)
    .error { _ in self.logout() }

```

Pridávanie autentifikačného tokenu do hlavičky už existujúcej požiadavky je taktiež dostatočne prehľadné:

```
request.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
```

4.4 Dokumentácia knižnice

Softvérová dokumentácia je dôležitou časťou knižnice, ktorá slúži v prvom rade jej používateľom, programátorom mobilných aplikácií. Tí sa na ňu môžu obrátiť v prípade nejasností, ktoré sa môžu vyskytnúť počas vývoja, či pri vyhľadávaní detailnejších informácií pre lepšie pochopenie fungovania použitého modulu či funkcie.

Knižnica GLNetworking, ktorá je praktickým výstupom tejto práce, obsahuje dokumentačnú časť integrovanú priamo v kóde. Pomocou špeciálnych komentárov, písaných vo formáte Markdown [16], dokumentuje jednotlivé softvérové komponenty priamo v mieste ich deklarácie. Pokročilé funkcie formátovania a linkovania obsahu umožňujú napísať dokumentáciu, ktorá umožní čitateľovi interaktívne prechádzať kód a v krátkom čase dohľadať relevantné informácie. Výhodou takejto tvorby dokumentácie je možnosť využitia softvérových nástrojov určených na generáciu dokumentov (pre jazyk Swift napr. Jazzy [17]). Pomocou generátorov je možné exportovať softvérovú dokumentáciu do štandardných formátov (PDF, HTML), ktorú následne možno použiť na propagáciu knižnice, napríklad na webových stránkach projektu.

Metóda obsahujúca dokumentačný komentár môže vyzeráť napríklad takto:

```
/**
```

Creates new GLNetworkManager instance.

- Parameter `networkResponseHandler`: Used to process HTTP responses based on the configured policies.
- Parameter `authenticationHandler`: Used for authenticating outgoing requests.

```
*/  
public init(networkResponseHandler: GLNetworkResponseHandler,  
            requestAuthenticator: GLRequestAuthenticator) {  
    self.networkResponseHandler = networkResponseHandler  
    self.requestAuthenticator = requestAuthenticator  
}
```

5 Návrh riešenia - Knižnica GLNetworking

Pri riešení problémov vyskytujúcich sa pri implementovaní komunikácie v architektúre klient - server je vhodné výsledné riešenie zapuzdriť do samostatného celku (v podobe knižnice; library, framework). To zabezpečí jednoduchú použiteľnosť v klientských aplikáciách a minimalizuje čas a úsilie potrebné na integráciu riešenia do projektu.

Pri návrhu knižnice som dbal na verejné rozhranie knižnice (public API). Toto rozhranie definuje spôsob komunikácie medzi klientskou aplikáciou a knižnicou. Návrh takéhoto rozhrania je dôležitý a v ideálnom prípade by sa nemal meniť, keďže zmena v public API môže spôsobiť nekompatibilitu medzi verziami a nutnosť prispôbiť kód klientskej aplikácie. Samozrejmosťou je API dokumentácia.

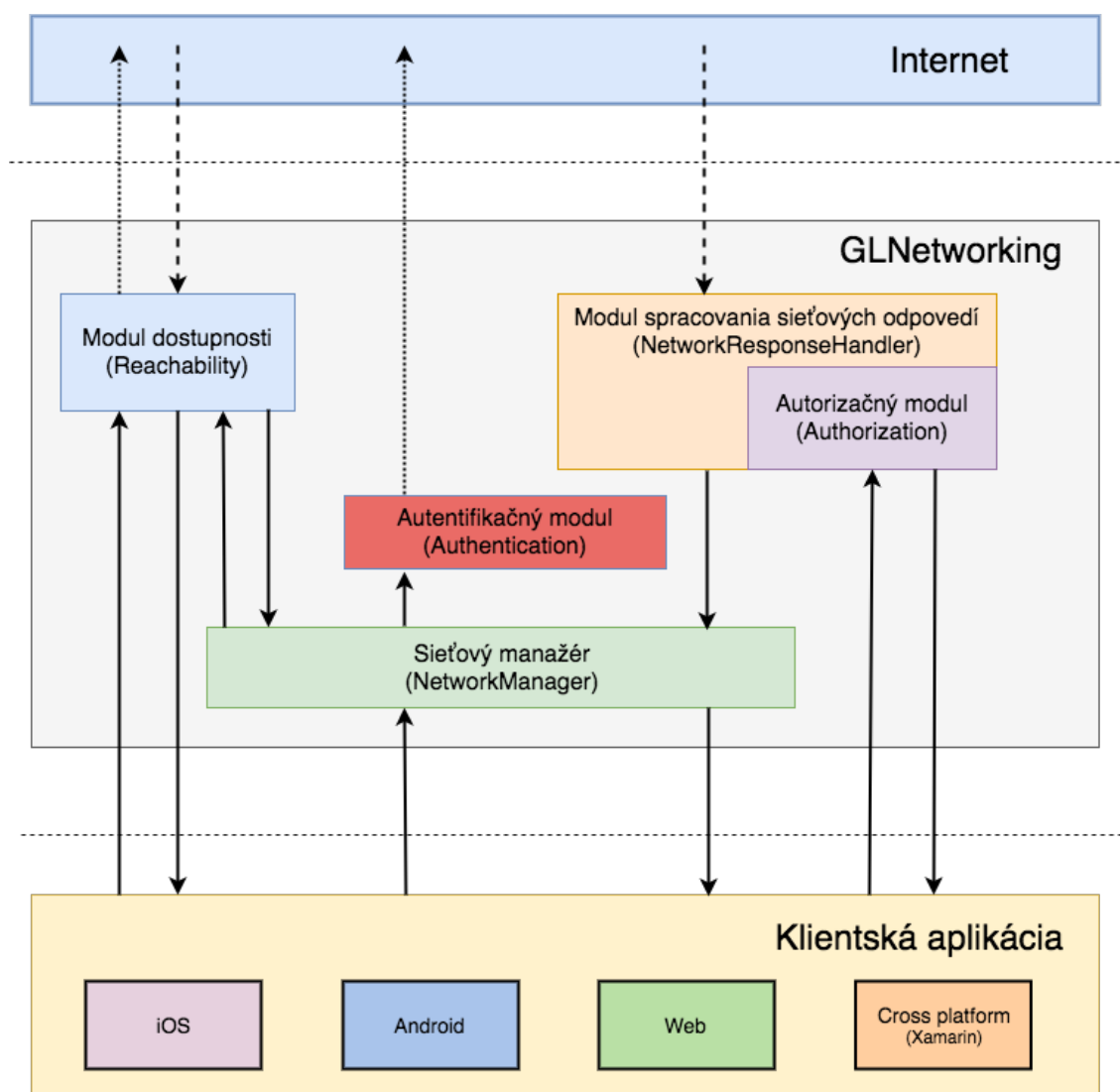
Sieťová komunikácia je komplikovaný proces, ktorý je definovaný množstvom protokolov a štandardov. Sieťové knižnice tieto nízkoúrovňové problémy a definície implementujú - zabezpečujú tak podporu komunikácie prostredníctvom sieťového rozhrania. Pri vývoji pokročilej funkcionality je preto vhodné navrhnúť toto riešenie ako nadstavbu rozširujúcu túto knižnicu. Obalením dosiahneme vyššiu úroveň abstrakcie.

Jadro knižnice GLNetworking je postavené na objektoch **URLSession** [18] (opísaných v kapitole 3.2). Obsahuje podporu pre protokoly HTTP/1.1, SPDY, HTTP/2 a možnosť rozšírenia o vlastný, špecifický protokol. Toto API je poskytované priamo operačným systémom iOS a teda je predpokladaná dlhá životnosť navrhovanej knižnice, vzhľadom k tomu, že budúce verzie operačného systému prinesú patričnú aktualizáciu tohto API.

Návrh architektúry knižnice GLNetworking je znázornený na obrázku 5.1:

- sieťový manažér (Network Manager),

- modul dostupnosti (Reachability),
- modul spracovania sieťových odpovedí (Network Response Handler),
- autentifikačný a autorizačný modul (Auth).



Obr. 5.1: Návrh architektúry knižnice GLNetworking

5.1 Sieťový manažér (NetworkManager)

Sieťový manažér je najdôležitejšou časťou knižnice. Jeho úlohou je sprostredkovanie komunikácie medzi aplikačným kódom a zvyškom knižnice. Všetky sieťové

požiadavky odoslané aplikačnou vrstvou sú smerované cez tento modul. Sieťový manažér zabezpečuje správne odoslanie požiadavky a korektné spracovanie následnej odpovede, vzhľadom na špecifickú konfiguráciu, ktorá je modulu poskytnutá pri jeho inštanciovaní.

Pri návrhu celkovej architektúry som si kládol otázku, či je vhodné zakomponovať takýto centrálny bod. Na prvý pohľad to predstavuje problém. Avšak, decentralizácia by pri reálnom využívaní knižnice priniesla to, že by bolo potrebné riešiť tieto veci v aplikačnom kóde. Preto je vhodné túto zodpovednosť presunúť na úroveň knižnice.

Cieľom sieťového manažéra je sprostredkovať komunikáciu medzi aplikačným kódom a knižnicou a zabezpečiť jednoduchosť implementácie odoslania požiadaviek, pri zachovaní riadenia a poriadku. Aplikačný kód nemusí konfigurovať každú požiadavku samostatne. Požiadavku stačí zdefinovať a o zvyšok sa postará manažér. Takéto riešenie má mnoho výhod, okrem iného aj redukovanie aplikačného kódu, zlepšenie prehľadnosti a jednoduché ladenie.

Jednoduchosť práce so sieťovým manažérom znázorňuje jeho verejné API, ktoré je reprezentované jedinou metódou:

```
public func execute(request: GLRequest)
```

Ďalšou užitočnou funkcionalitou, ktorú podporuje tento modul, je obmedzovanie maximálneho počtu paralelne otvorených sieťových požiadaviek. Toto opatrenie slúži na predídenie zahltenia sieťového rozhrania mobilného zariadenia, čo by mohlo mať negatívny vplyv na jeho výkon. Obmedzenie tiež zjednodušuje ladenie kódu a zvyšuje celkový prehľad a kontrolu. Pri naplnení stanoveného maximálneho počtu sú zvyšné požiadavky postupne zaraďované do fronty (queue), kde čakajú na uvoľnenie priestoru a následné odoslanie. Jedinú výnimku majú priame požiadavky (direct request), ktoré sú generované samotnou knižnicou. Keďže zabezpečujú správnu funkcionalitu niektorých modulov, ich priorita je vyššia.

5.2 Modul dostupnosti (Reachability)

Úlohou modulu dostupnosti je monitorovanie stavu spojenia so serverom. Tento primárny cieľ spĺňa tým, že deteguje zmeny v:

- pripojení na sieť,
- dostupnosti internetu,
- dostupnosti aplikačného servera.

Všetky tieto informácie sú dôležité pre aplikačnú vrstvu. Na optimalizáciu niektorých procesov ich využíva aj samotná knižnica. Napríklad, ak modul nedetektuje žiadne pripojenie do siete alebo je nedostupný internet, knižnica vyhodnotí všetky požiadavky na odoslanie na server chybou.

Pre potreby aplikačnej vrstvy sú tieto informácie primárne určené na zabezpečenie prechodu aplikácie medzi online a offline stavmi. Zaistenie plynulých prechodov je náročná úloha, ktorá pri správnej implementácii dokáže odlišiť aplikáciu od konkurencie. Pozitívne vplyva na stabilitu aplikácie, nakoľko sa eliminujú nepredvídateľné pády pri strate pripojenia a používateľ tak môže pokračovať v práci bez prerušenia. Pri opätovnom získaní konektivity sa aplikácia zosynchronizuje so serverom.

Preto sa odporúča vyvíjať mobilné aplikácie štýlom nazývaným „offline first“ [19], čo v preklade znamená, že aplikácia má byť v prvom rade pripravená na fungovanie bez internetu. Pri komunikácii so serverom sa musí myslieť na to, že toto spojenie môže byť nespoľahlivé a že sa dá predpokladať jeho zlyhanie.

Pre zlepšenie fungovania aplikácie je dôležité upovedomiť užívateľa o akýchkoľvek problémoch s pripojením. Ten ich môže v prípade potreby začať riešiť. Aplikácia sa tým okrem iného vyhne aj tomu, že za zdroj týchto problémov bude užívateľ nesprávne implicitne identifikovať samotnú aplikáciu.

Pri návrhu architektúry tohoto modulu sa naskytla otázka použitia návrhového vzoru singleton (jedináčik). Jeho použitie by bolo opodstatnené, keďže na Reachability modul sa dá pozrieť ako na jednu autoritu, podávajúcu informácie o mobilnom zariadení.

Pre potreby knižnice je lepšia možnosť klasického inšanciovania. Z pohľadu knižnice sa tým znižuje previazanosť jednotlivých modulov, čo zlepšuje kvalitu a testovateľnosť kódu. Namiesto jednej autority sa presúva možnosť voľby počtu inštancií na úroveň aplikačnej vrstvy. V určitých špecifických prípadoch je tiež výhodou to, že možnosť vytvorenia viacerých inštancií Reachability modulu umožňuje monitorovať dostupnosť viacerých serverov naraz. Tu je však nutné zohľadniť

energetický dopad takéhoto rozhodnutia a prispôbiť tomu konfiguráciu jednotlivých inštancií.

Na dosiahnutie žiadanej funkcionality využíva Reachability modul niekoľko častí. Pre monitorovanie sieťových nastavení mobilného zariadenia sa využíva **SC-NetworkReachability** API (opísané v kapitole 3.2) poskytovaného priamo operačným systémom. Na zistenie dostupnosti internetu a servera sa využívajú HTTP požiadavky, ktoré poskytuje aplikačná vrstva prostredníctvom svojho delegáta.

Na notifikovanie okolia pri zmenách stavu dostupnosti využíva Reachability modul princíp multicast delegátov (viď kapitola 4.2).

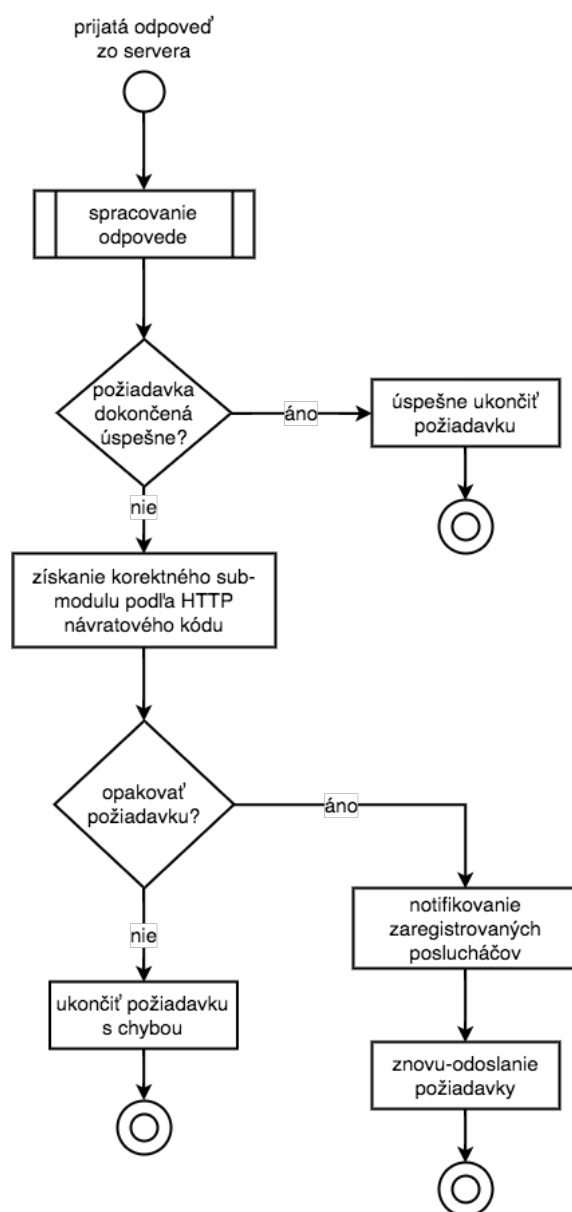
5.3 Modul spracovania sieťových odpovedí (Network Response Handler)

Odpovede všetkých sieťových požiadaviek, odoslaných cez GLNetworking knižnicu, sú automaticky spracovávané modulom spracovania sieťových odpovedí.

Tento modul slúži ako kontajner pre ďalšie sub-moduly (moduly spracovávajúce odpovede pre konkrétne HTTP návratové kódy), ktoré je možné nakonfigurovať z aplikačnej časti. Každý z týchto sub-modulov sa zameriava na spracovanie určitej časti HTTP návratových kódov. Programátor aplikačnej vrstvy si napríklad môže nakonfigurovať hlavný sub-modul, ktorým bude spracovávať všetky HTTP návratové kódy reprezentujúce chybu na strane klienta (4xx). Následne došpecifikuje konkrétny sub-modul, ktorým bude samostatne odchytať a spracovávať návratový kód 404, reprezentujúci chybu, kde žiadané prostriedky neboli nájdené. Tento postup možno využiť napríklad pri logovaní. Vzhľadom na poskytované možnosti konfigurácie je jednoduché nakonfigurovať tento modul tak, aby sa prispôbil špecifickým potrebám aplikácie.

Pre každý z týchto sub-modulov je možné nakonfigurovať počet opakovaní pôvodnej požiadavky predtým, než knižnica upozorní aplikačnú časť o úspešnom či neúspešnom dokončení požiadavky. Týmto postupom sa zabezpečí odfiltrovanie požiadaviek, ktoré mohli zlyhať napríklad z dôvodu krátkodobého výpadku servera.

Postup, akým sú spracované jednotlivé odpovede prijaté zo servera, je načrtnutý na obrázku 5.2.



Obr. 5.2: Diagram aktivity spracovávania odpovede na sieťovú požiadavku

5.4 Autentifikačný a autorizačný modul (Auth)

Úlohou tohto modulu je zabezpečiť autentifikáciu odchádzajúcich požiadaviek a reagovať na potrebu autorizácie voči serveru.

Autentifikovanie odchádzajúcich požiadaviek je riešené formou delegáta implementovaného na strane klientskej aplikácie. Delegovaním zodpovednosti sa zabezpečí podpora rôznych, aj neštandardných druhov autentifikácie. Odstráne-

nie tejto závislosti pomáha k jednoduchému rozšíreniu tejto knižnice a zlepšuje kvalitu kódu, keďže klientský kód nemusí riešiť autentifikáciu každej požiadavky zvlášť.

Autorizačný modul je špecifickým druhom obslužnej rutiny, ktorá spracováva odpoveď s HTTP návratovou hodnotou 401 - Unauthorized.

V prípade požiadavky, ktorá skončila neúspešne vzhľadom na potrebu autorizácie, vykoná tento modul priamu požiadavku (bez potreby čakať na uvoľnenie fronty) na overenie totožnosti. V prípade úspešného overenia je pôvodná požiadavka automaticky odoslaná opakovane. Aplikačný kód, ktorý iniciuje pôvodnú požiadavku, tak zaznamená kladné vybavenie požiadavky.

Požiadavku na autentifikáciu získava modul pomocou delegáta poskytovaného aplikačnou vrstvou. Aplikácia má tak priestor získať potrebné údaje a vyskladať potrebnú požiadavku.

5.5 Výhody navrhnutého riešenia

Navrhnuté riešenie, ktoré prezentujem v tejto práci, má oproti existujúcim dostupným riešeniam niekoľko výhod. Knižnica GLNetworking pracuje na vyššej úrovni abstrakcie, rieši komplexné problémy aplikačnej vrstvy.

Modul GLReachability získava informácie o stave pripojenia nielen na základe konektivity mobilného zariadenia, ale aj pomocou opakovaného dotazovania sa na server. Tieto dáta pomôžu vývojárom mobilných aplikácií pri implementovaní prechodov medzi online a offline stavom.

Vysoko-úrovňová možnosť konfigurácie spracovávania sieťových odpovedí umožňuje naprogramovať spracovanie chýb na úrovni aplikačnej vrstvy. To pozitívne vplýva na zrozumiteľnosť a prehľadnosť výsledného kódu.

Ďalšou možnosťou automatizácie je opakovanie pôvodnej neúspešnej požiadavky jednak bez narušenia toku odoslania požiadavky a zároveň spracovania odpovede na úrovni aplikačnej vrstvy. Táto funkcionálna umožňuje vývojárom zakomponovať do svojich aplikácií jednoduché spracovanie chýb, bez nutnosti písania kódu navyše.

Automatické spracovanie požiadavky na autentifikáciu zo strany servera a automatická autorizácia odchádzajúcich sieťových požiadaviek umožňujú programátorom využiť tieto služby a sústrediť sa na vývoj funkcionality aplikácie.

6 Vyhodnotenie

Knižnica tvorí celok, ktorý je ľahko integrovateľný či už do existujúcej alebo do novo vytváranej aplikácie. Pri jej tvorbe som sa držal požiadaviek, ktoré pomáhajú riešiť stanovené problémy.

Je napísaná ako rozšírenie, ktoré je postavené nad sieťovou knižnicou **URL-Session** (kapitola 3.1). Týmto dizajnovým rozhodnutím sa zabezpečila kompatibilita so spomínaným systémovým sieťovým rozhraním. Programátorom aplikačnej vrstvy to umožňuje zakomponovať knižnicu do ich existujúceho riešenia bez toho, aby narušili kód, ktorý bol napísaný pred jej vložením do projektu. Takéto preportovanie kódu si z ich strany bude vyžadovať minimálne úsilie.

Ďalšou výhodou je možnosť využívať pokročilú funkcionálnu poskytnutú systémovým API bez toho, aby sa narušila kompatibilita tohto kódu s knižnicou. V prípade potreby tak knižnica neobmedzuje programátorov, naopak, dáva im možnosť voľby rozhodnúť sa, či použijú vysoko-úrovňový prístup knižnice, alebo bude ich riešenie špecifickejšie a pracujúce na nižšej vrstve.

Toto rozhodnutie má vplyv aj na životnosť knižnice. Vzhľadom na to, že na svoju funkcionálnu používa natívne systémové rozhrania sa dá predpokladať, že tieto rozhrania budú podporované a aktualizované aj v nasledujúcich verziách operačného systému. Riziko budúcej kompatibility je preto minimálne a nie je nutné sa obávať výraznejších zmien či nutnosti prepisovať knižnicu.

Pozitívne vplýva toto rozhodnutie aj na optimalizáciu „krivky učenia sa“ používania tejto knižnice. Programátori oboznámení s aktuálnymi rozhraniami operačného systému nebudú mať žiaden problém pochopiť a začať využívať knižnicu od prvého momentu.

Knižnica má rozdielny pohľad na riešenie najčastejších problémov, ktoré sa vyskytujú v moderných mobilných aplikáciách pri implementácií sieťovej komunikácie. Na základe analýzy (kapitola 2), ktorá vzišla nielen z mojich poznatkov a

pozorovaní, ale hlavne z komunikácie s ľuďmi s dlhoročnou praxou v tomto obore si myslím, že je tento odlišný pohľad výhodou. Poskytovanou funkcionalitou šetrí knižnica programátorom čas, ktorý môže byť strávený produktívnejšou prácou na konkrétnom zadaní. Vývojári mobilných aplikácií majú možnosť nakonfigurovať správanie sa knižnice podľa ich špecifických potrieb a napísať len minimum kódu - na odoslanie a spracovanie sieťovej požiadavky. Nemusia sa pri tom obávať toho, že ich implementácia bude nedostatočná, alebo že nebude dosahovať úroveň, ktorú je potrebné splniť pri vývoji „offline first“ [19] aplikácie.

6.1 Testovacia klientská aplikácia

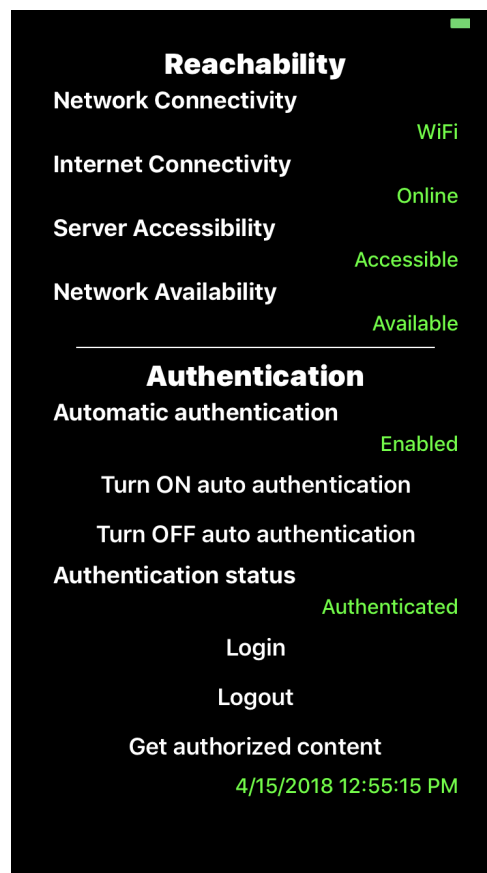
Pre potreby demonštrácie fungovania navrhutej knižnice som vytvoril testovaciu demo aplikáciu, ktorej používateľské rozhranie je zobrazené na obrázku 6.1. Pri jej návrhu a tvorbe som sa zameriaval na funkcionalitu a nekládol som dôraz na používateľské rozhranie. Úlohou tejto aplikácie je overiť funkčnosť, otestovať kód verejného rozhrania (public API) a získať reálny pohľad na použitie knižnice v praxi. Používateľské rozhranie je navrhnuté minimalisticky, s dôrazom na jednoduchosť a prehľadnosť poskytovaných informácií.

Pre potreby testovania využívam Microsoft Azure server. Ten je napísaný v jazyku C# a využíva technológiu ASP.NET Core MVC [20].

Funkcionalita umožňuje otestovať dostupnosť a komunikáciu so serverom a implementovať autentifikáciu. Implementácia autentifikácie využíva metódu JSON Web Tokenov [21] (JWT), kedy je potrebné bezpečne prezentovať nároky (claims) medzi dvoma stranami.

Testovacia aplikácia obsahuje funkcionalitu, ktorá:

1. zobrazuje stav modulu dostupnosti v reálnom čase,
2. umožňuje povoliť, či zakázať možnosť automatickej autentifikácie voči serveru,
3. poskytuje možnosť manuálne vyvolať prihlásenie a odhlásenie aplikácie voči serveru,
4. umožňuje manuálne vyvolať odoslanie požiadavky na získanie autorizovaného obsahu zo servera.



Obr. 6.1: Používateľské rozhranie testovacej aplikácie

Pomocou tejto funkcionality vieme otestovať všetky funkcie a vlastnosti, ktoré knižnica ponúka.

Práca s modulom dostupnosti je jednoduchá a priamočiara. Pri inicializácii tohto modulu je nutné poskytnúť delegáta, ktorý zabezpečí všetky potrebné informácie pre korektné fungovanie tohto modulu a určí interval obnovy týchto informácií. Delegátom môže byť aj samotný kontrolér:

```
reachability = GLReachability(delegate: self, updateInterval: 1)!

extension ViewController: GLReachabilityDelegate {
    func dnsCheckRequest(sender: GLReachability) -> GLRequest {
        return GLRequest.create(url: URLConstants.networkAvailabilityURL)!
    }
    func serviceCheckRequest(sender: GLReachability) -> GLRequest {
        return GLRequest.create(url: URLConstants.availabilityURL)!
    }
}
```

```

    }
}

```

Pre nakonfigurovanie modulu spracovania sieťových odpovedí je potrebné poskytnúť delegáta, ktorý zabezpečí komunikáciu iniciovanú z modulu:

```

networkResponseHandler = GLNetworkResponseHandler()

let authHandler = GLAuthHandler(
    delegate: self,
    requestAuthenticator: self
)
authHandler.setRepetition(3)
networkResponseHandler.add(handler: authHandler)

extension ViewController: GLAuthDelegate {
    func authenticationRequest(sender: GLAuthHandler) -> GLRequest {
        return getLoginRequest()
    }
}

```

Posledným krokom je vytvoriť sieťového manažéra, ktorý slúži ako centrálny bod na odosielanie sieťovej komunikácie. Ten na svoju minimálnu prevádzku potrebuje inštanciu modulu spracovávanie sieťových odpovedí. V prípade nutnosti autentifikácie odosielajúcich požiadaviek je potrebné poskytnúť delegáta, ktorý bude túto funkcionálnu vykonávať:

```

networkManager = GLNetworkManager(
    networkResponseHandler: networkResponseHandler,
    requestAuthenticator: self
)
extension ViewController: GLRequestAuthenticator {
    func authenticate(request: GLRequest) {
        if let token = authToken {
            request.setValue("Bearer \(token)",
                forHTTPHeaderField: "Authorization")
        }
    }
}

```



```
    }  
  }  
}
```

6.2 Zhrnutie

V analytickej časti tejto práci som pomenoval tri často sa opakujúce problémy, spojené s implementáciou sieťovej komunikácie v architektúre klient - server:

1. detekcia konektivity na internet a spojenia so serverom,
2. automatizované obsluhovanie autentifikačnej a autorizačnej rutiny,
3. automatická správa chybových stavov.

Tieto problémy som vyriešil nasledovne:

1. nepretržité monitorovanie pripojenia na sieť, dostupnosti internetu a overenie možnosti komunikácie so serverom,
2. konfigurácia plne automatického reagovania na potrebu autentifikácie voči serveru a automatické autorizovanie odosielajúcich požiadaviek,
3. konfigurácia spracovania sieťových odpovedí na základe HTTP stavových kódov a možnosť automatického opakovania neúspešných požiadaviek.

7 Záver

Táto práca sa venuje oblasti sieťovej komunikácie a komplikáciám, ktoré sú spojené s jej implementáciou v mobilných aplikáciách, konkrétne v architektúre klient-server. Analyzuje tri často sa opakujúce druhy problémov a vysvetľuje dôležitosť a výhody správneho riešenia. Práca tiež skúma a porovnáva existujúce sieťové knižnice, ktoré sú dostupné pre platformu iOS a zisťuje, či tieto knižnice ponúkajú riešenia na popisované problémy.

Vzhľadom na to, že sa existujúce knižnice týmito problémami buď vôbec nezaobierajú, alebo ponúkajú len čiastočné a nedokonalé riešenia, sa práca venuje návrhu nového riešenia, ktoré bude postačujúce pre zabezpečenie potrieb väčšiny bežných mobilných aplikácií.

Ako riešenie na popisované problémy navrhujem architektúru knižnice a popisujem jej funkcionality, ktorej úlohou je minimalizovať úsilie potrebné pre korektnú implementáciu komunikácie z pohľadu klienta. Knižnica je navrhnutá modulárne, s dôrazom na jej konfigurovateľnosť a prispôsobiteľnosť podľa špecifických, individuálnych potrieb aplikácie. Na základe týchto požiadaviek je architektúra knižnice navrhnutá s dôrazom na jednoduchosť a prehľadnosť.

Pre zabezpečenie čo najväčšej compatibility a predĺženie jej životnosti je knižnica napísaná ako rozšírenie štandardných programových rozhraní poskytovaných operačným systémom. Výrazne sa tým zjednodušuje postupná integrácia knižnice do existujúceho projektu.

Funkcionalita a prínos navrhovaného riešenia boli otestované pomocou demo aplikácie. Tá ukazuje reálny pohľad na využívanie knižnice v aplikácii. Kód je prehľadný, expresívny. Na konfiguráciu je potrebné minimálne množstvo kódu, pomocou ktorého aplikácia určí špecifické a očakávané správanie sa knižnice. Po úspešnej konfigurácii sa programátor môže plne venovať svojmu prioritnému cieľu, implementácii funkcionality vyvíjanej aplikácie. Odosielanie a spracovanie

sieťových požiadaviek je jednoduché. Všetky potrebné nastavenia a opatrenia vykonáva knižnica automaticky. Súčasťou projektu je aj testovacia klientská aplikácia, slúžiaca na demonštráciu práce s knižnicou a na otestovanie ponúkanej funkcionality. Jednotlivé moduly a triedy, ktoré sa nachádzajú v projekte, sú zdokumentované štandardným spôsobom priamo v zdrojovom kóde, tým pádom je možné vygenerovať dokumentáciu v ľubovoľnom formáte podľa potrieb.

Literatúra

- [1] Statista Ltd. *Štatistika počtu používateľov smartfónov na svete*. 2016. URL: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/> (cit. 02.11.2017).
- [2] Apple Inc. *Webová stránka prezentujúca programovací jazyk Swift pre vývojárov*. 2018. URL: <https://developer.apple.com/swift/> (cit. 14.04.2018).
- [3] Apple Inc. *Webová stránka prezentujúca programovací jazyk Swift*. 2018. URL: <https://www.apple.com/swift/> (cit. 14.04.2018).
- [4] Jaroslav Janáček. *Prezentácia - Riadenie prístupu*. 2013. URL: https://www.csirt.gov.sk/doc/MFSRVzdelavanie/03Vzdelavanie2013/Prezentacie_specialisti_na_informacne_technologie_a_specialisti_na_informacnu_bezpecnost/PrezIT_2013_06_krit_Riadenie_pristupu.pdf (cit. 10.04.2018).
- [5] RFC 2616 Fielding, et al. *Špecifikácia HTTP/1.1 protokolu - stavové kódy*. 1999. URL: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1.1> (cit. 02.11.2017).
- [6] Alamofire Software Foundation. *Domovská stránka projektu AFNetworking*. 2016. URL: <https://github.com/AFNetworking/AFNetworking> (cit. 02.11.2017).
- [7] Alamofire Software Foundation. *Domovská stránka projektu Alamofire*. 2016. URL: <https://github.com/Alamofire/Alamofire> (cit. 02.11.2017).
- [8] Apple Inc. *Oficiálna dokumentácia odporúčaných postupov pri implementovaní sieťovania*. 2017. URL: <https://developer.apple.com/library/content/documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/ChoosingTheRightNetworkingAPI/ChoosingTheRightNetworkingAPI.html> (cit. 01.01.2018).

-
- [9] Apple Inc. *Oficiálna dokumentácia SCNetworkReachability API*. 2017. URL: <https://developer.apple.com/documentation/systemconfiguration/scnetworkreachability-g7d> (cit. 02. 11. 2017).
- [10] Gilbert Held. *Server management*. Boca Raton, FL: Auerbach, 2000. ISBN: 9781420031065.
- [11] Mark Massé. *REST API design rulebook*. Sebastopol, CA: O'Reilly, 2012. ISBN: 978-1-449-31050-9.
- [12] tutsplus.com. *Článok zaoberajúci sa použitím návrhového vzoru Delegát*. 2015. URL: <https://code.tutsplus.com/articles/design-patterns-delegation--cms-23901> (cit. 18. 02. 2018).
- [13] oodesign.com. *Článok zaoberajúci sa princípom inverzie závislosti (dependency inversion)*. 2018. URL: <http://www.oodesign.com/dependency-inversion-principle.html> (cit. 18. 02. 2018).
- [14] dzone.com. *Článok zaoberajúci sa princípmi S.O.L.I.D.* 2017. URL: <https://dzone.com/articles/the-5-solid-principles-explained> (cit. 18. 02. 2018).
- [15] J. M. Bishop. *C# návrhové vzory*. Brno: Zoner Press, 2010. ISBN: 978-80-7413-076-2.
- [16] Matt Cone. *Webová stránka venujúca sa značkovaciemu jazyku Markdown*. 2018. URL: <https://www.markdownguide.org/getting-started> (cit. 22. 04. 2018).
- [17] Realm Inc. *Oficiálna dokumentácia a repozitár softvérového nástroja Jazzy*. 2018. URL: <https://github.com/realm/jazzy> (cit. 22. 04. 2018).
- [18] Apple Inc. *Oficiálna dokumentácia NSURLSession API*. 2016. URL: <https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/URLLoadingSystem/Articles/UsingNSURLSession.html> (cit. 02. 11. 2017).
- [19] offlinefirst.org. *Webová stránka zaoberajúca sa problematikou offline first návrhu*. 2017. URL: <http://offlinefirst.org/> (cit. 24. 12. 2017).
- [20] Microsoft. *Webová stránka opisujúca prehľad technológie ASP.NET Core MVC*. 2018. URL: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.1> (cit. 15. 04. 2018).
- [21] M. Jones, J. Bradley a N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. <http://www.rfc-editor.org/rfc/rfc7519.txt>. RFC Editor, máj 2015. URL: <http://www.rfc-editor.org/rfc/rfc7519.txt>.

Zoznam pojmov

API Aplikačné programové rozhranie, z anglického Application Programming Interface. Jedná sa o zbierku knižníc, funkcií, tried a pod., ktoré určujú, akým spôsobom sa majú funkcie knižníc volať zo zdrojového kódu programu. Vývojári môžu prostredníctvom komunikácie s API využívať funkcie, ktoré poskytuje.

Autentifikácia Preukázanie skutočnosti, že identifikácia používateľa je pravdivá. Na overenie totožnosti sa využívajú napríklad heslá, biometrické vlastnosti alebo overenie SMS kódom.

Autorizácia Rozhodovanie, či daný subjekt (používateľ) má oprávnenie vykonať požadovanú operáciu s daným objektom (súbor, informácia, funkcia systému).

Decorator Návrhový vzor, ktorý sa využíva vtedy, ak je potrebné poskytnúť spôsob pre dynamické pripojenie nového stavu a chovania k objektu, pričom dekorovaný objekt o tejto skutočnosti nevie.

Factory Návrhový vzor, pomocou ktorého je možné zabezpečiť tvorbu nových objektov bez potreby tvorby konkrétnych rozhodnutí. Logika zabezpečujúca rozhodnutia pri tvorbe týchto objektov je extrahovaná v samostatnej triede. Tento koncept využívajú návrhové vzory Factory Method (továrenská metóda) a Abstract Factory (abstraktná továreň).

HTTP Hypertextový prenosový protokol, z anglického Hypertext Transfer Protocol. Protokol na prenos HTML dokumentov medzi servermi a klientmi služby WWW. Pôsobí ako primárna metóda prepravy informácií na internete.

- HTTPS** Zabezpečený hypertextový prenosový protokol, z anglického Hypertext Transfer Protocol Secure. Jedná sa o zabezpečenú verziu protokolu HTTP. Prenos dát je šifrovaný pomocou protokolu SSL alebo TLS.
- JSON** JavaScriptový objektový zápis, z anglického JavaScript Object Notation. Je to spôsob zápisu dát, ktorý je nezávislý na počítačovej platforme. Využíva sa na prenos dát.
- JWT** Z anglického JSON Web Token. Otvorený štandard pre tvorbu prístupových tokenov obsahujúcich informácie o oprávneniach vlastníka tokenu.
- POSIX** Prenosné rozhranie pre operačné systémy, z anglického Portable Operating System Interface. Rozhranie určujúce štruktúru a detaily POSIX kompatibilných systémov.
- REST** Z anglického Representational State Transfer. Súbor pravidiel a štandardov pre dizajn distribuovanej architektúry. Obsahuje odporúčania pre návrh funkčných a elegantných API.
- Singleton** Návrhový vzor, ktorého úlohou je zaistiť, aby existovala iba jedna inštancia triedy. Zároveň k tomuto objektu poskytuje globálny prístupový bod.
- SPDY** Zastaraný protokol vyvinutý pre prenos dát na WWW. Cieľom protokolu bolo minimalizovať čakaciu dobu potrebnú na načítanie stránky a zvýšenie bezpečnosti.
- TCP** Protokol riadenia prenosu, z anglického Transmission Control Protocol. Spojový protokol, ktorý zabezpečuje, že dáta odoslané z jedného konca spojenia budú prijaté na druhej strane spojenia v rovnakom poradí a bez chýbajúcich častí.
- URI** Jednotný identifikátor prostriedku, z anglického Uniform Resource Identifier. Kompaktný reťazec znakov používaný na identifikáciu alebo pomenovanie zdroja.

WWAN Veľká bezdrôtová sieť, z anglického Wireless Wide Area Network. Je to sieť pokrývajúca veľké oblasti. Využíva infraštruktúru siete mobilných operátorov, prostredníctvom ktorej ponúka prístup na internet. Patria tu napríklad GSM (2G), UMTS (3G) či LTE (4G).

XML Rozšíriteľný značkovací jazyk, z anglického Extensible Markup Language. Jazyk, ktorý umožňuje jednoduché vytváranie konkrétnych značkovacích jazykov na rôzne účely a široké spektrum rôznych typov údajov. Využíva sa predovšetkým na výmenu údajov medzi aplikáciami a na zverejňovanie dokumentov.

Zoznam príloh

Príloha A Používateľská príručka

Príloha B Systémová príručka

Príloha C CD médium – záverečná práca v elektronickej podobe

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Riešenie problémov sieťovej komunikácie v
mobilných aplikáciách**

**Príloha A
Používateľská príručka**

2018

Lukáš Vavrek

Obsah

1	Popis knižnice	1
2	Používanie knižnice	2
2.1	Inicializácia a konfigurácia knižnice	2
2.2	Konfigurácia sub-modulov pre spracovanie sieťových odpovedí . .	4
2.3	Tvorba a odoslanie sieťovej požiadavky, Spracovanie odpovede . . .	5

1 Popis knižnice

Knižnica GLNetworking je sieťová knižnica napísaná v jazyku Swift, určená pre platformu iOS. Je napísaná ako rozšírenie systémového API URLSession.

Jej úlohou je poskytnúť programátorom mobilných aplikácií jednoduché programové rozhranie na odosielanie a prijímanie sieťových požiadaviek a zabezpečiť korektnú implementáciu tejto komunikácie.

Sekundárnym cieľom knižnice je minimalizovať počet potrebných riadkov kódu pre dosiahnutie požadovanej funkcionality, podporiť princípy DRY (don't repeat yourself) a poskytovať API, ktoré sa používa jednoducho a intuitívne.

Knižnica zapuzdruje tri hlavné riešenia problémov, ktoré sa často vyskytujú pri implementovaní sieťovej komunikácie. Sú to:

- detekcia spojenia na internet, ku ktorému dochádza za Wi-Fi prístupovým bodom a detekcia spojenia so serverom,
- detekcia a následné spracovanie chybových návratových kódov s možnosťou opakovania žiadostí,
- automatická autorizačná a autentifikačná rutina pre opätovné posielanie požiadavky na server.

2 Používanie knižnice

Knižnica podporuje komunikáciu s aplikačnou vrstvou dvoma spôsobmi:

- volaním metód verejného API,
- poskytnutím delegáta.

Pomocou volania metód verejného API sa iniciuje komunikácia z aplikačnej vrstvy. Tento spôsob zvyčajne slúži na konfiguráciu alebo volanie určitej funkcionality knižnice.

Delegáti naopak poskytujú možnosť iniciovať komunikáciu z knižnice, čo sa využíva na získavanie informácií potrebných pre prácu knižnice.

Nasledujúce kapitoly bližšie popisujú spôsob používania knižnice.

2.1 Inicializácia a konfigurácia knižnice

GLNetworkManager potrebuje pre svoju prácu inštanciu GLNetworkResponseHandler, aby vedel, ako sa má rozhodovať pri spracovávaní odpovede na sieťovú požiadavku. GLRequestAuthenticator a GLReachability sú voliteľné.

Konfigurácia GLNetworkManagera.

```
reachability = GLReachability(delegate: self, updateInterval: 1)!  
networkResponseHandler = GLNetworkResponseHandler()
```

```
networkManager = GLNetworkManager(  
    networkResponseHandler: networkResponseHandler,  
    requestAuthenticator: self,  
    reachability: reachability  
)
```

Pri inštanciovaní modulu `GLReachability` je potrebné poskytnuť dvojicu argumentov:

- `updateInterval` (v sekundách) určuje, ako často sa bude vykonávať overovanie dostupnosti,
- `GLReachabilityDelegate` slúži na spätnú komunikáciu z knižnice, ktorá potrebuje získať dodatočné informácie na vykonanie svojej činnosti. Delegát obsahuje metódy, ktorých návratová hodnota je typu `GLRequest`. Tieto požiadavky budú použité pri vykonávaní overovania dostupnosti.

```
extension ViewController: GLReachabilityDelegate {
    func dnsCheckRequest(sender: GLReachability) -> GLRequest {
        return GLRequest.create(url: URLConstants.networkAvailabilityURL)!
    }

    func serviceCheckRequest(sender: GLReachability) -> GLRequest {
        return GLRequest.create(url: URLConstants.availabilityURL)!
    }
}
```

Nepovinným parametrom pre inštanciovanie `GLNetworkManagera` je aj objekt, ktorý implementuje rozhranie `GLRequestAuthenticator`. Jeho úlohou je autentifikovať sieťové požiadavky odchádzajúce zo zariadenia.

Príkladom implementácie `GLRequestAuthenticator` môže byť autentifikácia pomocou tzv. Bearer tokenov (napr. JWT), ktoré fungujú na princípe pridania HTTP hlavičky do odchádzajúcej požiadavky.

```
extension ViewController: GLRequestAuthenticator {
    func authenticate(request: GLRequest) {
        if let token = authToken {
            request.setValue("Bearer \(token)",
                             forHTTPHeaderField: "Authorization")
        }
    }
}
```

2.2 Konfigurácia sub-modulov pre spracovanie sieťových odpovedí

`GLNetworkResponseHandler` funguje ako kontajner obsahujúci jednotlivé sub-moduly pre spracovanie sieťových odpovedí, ktoré sú určené pre špecifické návratové HTTP stavové kódy.

Príkladom sub-modulu môže byť logovanie klientských a serverových chýb (HTTP statusy 4XX a 5XX).

```
networkResponseHandler = GLNetworkResponseHandler()

let serverErrorHandler = GLAsyncResponseStatusCodeHandler(
    httpStatusCodes: Array(400..<600))

serverErrorHandler.setRepetition(3)
serverErrorHandler.addDelegate(GLResponseStatusCodeHandlerDelegate {
    print("[ ERROR ]")
})
networkResponseHandler.add(handler: serverErrorHandler)
```

Špecifickým typom sub-modulu je aj `GLAuthHandler`, ktorý spracováva sieťové odpovede s HTTP stavovým kódom 401(unauthorized). Pre správne fungovanie vyžaduje v konštruktoe dva parametre:

- delegát `GLAuthDelegate`, ktorý modulu poskytne požiadavku použitú na autentifikáciu,
- `GLRequestAuthenticator`, ktorý zabezpečí automatické autentifikovanie pôvodnej (zlyhanej) požiadavky na jej opätovné odoslanie.

```
let authHandler = GLAuthHandler(
    delegate: self,
    requestAuthenticator: self
)
authHandler.setRepetition(3)
networkResponseHandler.add(handler: authHandler)
```

```

extension ViewController: GLAuthDelegate {
    func authenticationRequest(sender: GLAuthHandler) -> GLRequest {
        return GLRequest
            .create(url: URLConstants.loginURL)!
            .httpMethod("post")
            .jsonBody([
                "username": "",
                "password": ""
            ])
            .success(loginSuccessful)
            .error { _ in self.logout() }
    }
}

```

2.3 Tvorba a odoslanie sieťovej požiadavky, Spracovanie odpovede

Vytváranie sieťovej požiadavky využíva návrhový vzor Builder. Trieda `GLRequest` obsahuje verejné API, pomocou ktorého je možné poskladať výslednú požiadavku.

Odpoveď môže byť úspešná (`success`) alebo neúspešná (`error`). Pomocou týchto metód je možné reagovať na odpoveď zo servera a pracovať s prijatými dátami.

```

let request = GLRequest
    .create(url: URLConstants.serverTimeURL)?
    .success { response in
        DispatchQueue.main.async {
            label.text = String(bytes: response.data!, encoding: .utf8)!
            label.textColor = UIColor.green
        }
    }
    .error { response in
        self.setFetchFailed(forLabel: self.fetchAuthContentLbl)
    }
}

```


Vytvorené požiadavky môžeme následne odoslať prostredníctvom metódy `execute`, ktorú obsahuje `GLNetworkManager`.

```
if let request = request {  
    networkManager.execute(request: request)  
}
```

**Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky**

**Riešenie problémov sieťovej komunikácie v
mobilných aplikáciách**

**Príloha B
Systémová príručka**

2018

Lukáš Vavrek

Obsah

1	Popis knižnice	1
2	Architektúra knižnice	2
3	Štruktúra projektu	3
3.1	GLNetworking projekt	3
3.2	ReachabilityExample projekt	4
3.3	Pods projekt	4
3.4	Externé závislosti	5
3.5	Zostavenie projektu	5

Zoznam obrázkov

2.1	Architektúra knižnice GLNetworking	2
-----	--	---

1 Popis knižnice

Knižnica GLNetworking je sieťová knižnica napísaná v jazyku Swift a je určená pre platformu iOS. Je napísaná ako rozšírenie systémového API URLSession.

Jej úlohou je poskytnúť programátorom mobilných aplikácií jednoduché programové rozhranie na odosielanie a prijímanie sieťových požiadaviek a zabezpečiť korektnú implementáciu tejto komunikácie.

Sekundárnym cieľom knižnice je minimalizovať počet potrebných riadkov kódu pre dosiahnutie požadovanej funkcionality, podporiť princípy DRY (don't repeat yourself) a poskytovať API, ktoré sa používa jednoducho a intuitívne.

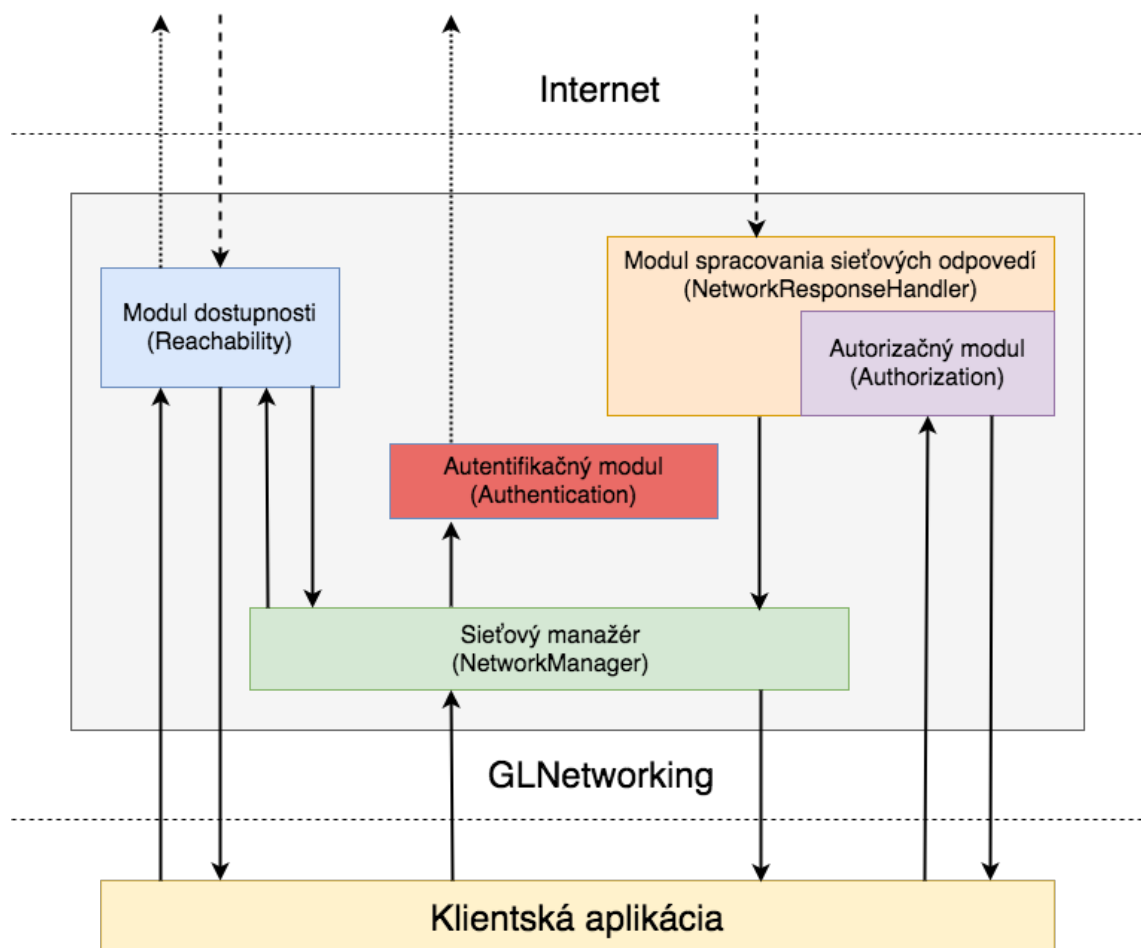
Knižnica zapuzdruje tri hlavné riešenia problémov, ktoré sa často vyskytujú pri implementovaní sieťovej komunikácie. Sú to:

- detekcia spojenia na internet, ku ktorému dochádza za Wi-Fi prístupovým bodom a detekcia spojenia so serverom,
- detekcia a následné spracovanie chybových návratových kódov s možnosťou opakovania žiadostí,
- automatická autorizačná a autentifikačná rutina pre opätovné posielanie požiadavky na server.

Táto systémová príručka je napísaná pre programátorov, ktorí majú záujem študovať internú štruktúru a fungovanie knižnice a pokračovať vo vývoji ďalšej funkcionality, prípadne pracovať na odstránení nájdených chýb.

2 Architektúra knižnice

Architektúra knižnice (obrázok 2.1) je navrhnutá modulárne, so snahou podporiť rozširovanie knižnice o ďalšie moduly implementujúce nové funkcionality.



Obr. 2.1: Architektúra knižnice GLNetworking

3 Štruktúra projektu

Workspace GLNetworking obsahuje tri fyzické projekty. Sú to:

- GLNetworking,
- ReachabilityExample,
- Pods.

3.1 GLNetworking projekt

Projekt GLNetworking je najdôležitejšou časťou celkového workspace-u. Nachádzajú sa v ňom súbory, ktorých zdrojové kódy implementujú funkcionality knižnice. Obsahuje referenciu na projekt Pods. Výstupom kompilácie a zostavenia projektu je framework, nachádzajúci sa v priečinku Products.

Fyzické rozdelenie jednotlivých zdrojových kódov knižnice zodpovedá logickej štruktúre knižnice.

- **AuthHandler** - obsahuje triedy a protokoly, ktoré sa zaoberajú autentifikáciou a autorizáciou.
- **GLNetworkManager** - obsahuje triedu sieťového manažéra, centrálného bodu komunikácie medzi knižnicou a klientskou časťou.
- **HTTP Helper** - obsahuje pomocné triedy, ktoré implementujú proces tvorby sieťovej požiadavky a reprezentáciu odpovede.
- **Helpers** - obsahuje pomocné triedy a dátové štruktúry, ktoré sú využívané pri implementácii konkrétnych častí funkcionality jednotlivých modulov.

- **Reachability** - obsahuje triedy, protokoly a dátové štruktúry, ktoré spoločne tvoria samostatný modul overovania dostupnosti.
- **NetworkResponseHandler** - obsahuje triedy, protokoly a dátové štruktúry, ktoré sa týkajú modulu spracovania odpovedí na sieťové požiadavky. GLNetworkResponseHandler tvorí kontajner pre ďalšie sub-moduly orientované na špecifické HTTP stavové kódy. GLResponseStatusCodeHandler je trieda, pomocou ktorej je možné implementovať vlastný typ sub-modulu.

3.2 ReachabilityExample projekt

Projekt ReachabilityExample obsahuje implementáciu demo aplikácie, ktorá demonštruje prácu s knižnicou a dáva príklad na jej využitie v reálnom projekte. Referencuje projekty Pods a GLNetworking.framework, ktorý je generovaný projektom GLNetworking. Výstupom kompilácie a zostavenia projektu je aplikácia pre platformu iOS.

Zdrojové kódy sa nachádzajú v priečinku ReachabilityExample.

- **Models** - obsahuje dátové štruktúry využívané v aplikácii.
- **CustomComponents** - obsahuje programové komponenty, ktoré sú nakonfigurované tak, aby ich bolo možné opakovane používať pri vývoji používateľského rozhrania.
- **ViewController.swift** - kontrolér obsahujúci metódy na obsluhu a interakciu používateľa s používateľským rozhraním. Zapuzdruje kód sieťovej komunikácie, za využitia knižnice GLNetworking.
- **Main.storyboard** - obsahuje dizajn používateľského rozhrania demo aplikácie.

3.3 Pods projekt

Projekt Pods obsahuje všetky externé komponenty, na ktorých sú závislé zvyšné projekty nachádzajúce sa vo workspace-e. Tento projekt je auto-generovaný manažérom závislostí CocoaPods.

- **MulticastDelegate** - implementácia multicast delegátov na princípe hašovacej tabuľky.

3.4 Externé závislosti

Knižnica GLNetworking je navrhnutá tak, aby využívala knižnice a sieťové rozhrania poskytované priamo operačným systémom.

Je závislá iba na jednom externom module a to `MulticastDelegate.swift`. Tento modul je do projektu integrovaný prostredníctvom manažéra závislostí `CocoaPods`.

3.5 Zostavenie projektu

Pri tvorbe knižnice bolo snahou minimalizovať úsilie pre zostavenie projektu a vytvorenie frameworku. Tento proces je možné zvládnuť pomocou jedného kliku.

Pre zostavenie projektu je potrebné mať nainštalované programy `CocoaPods` a `XCode`.

Najprv je potrebné vyriešiť chýbajúce závislosti knižnice. V termináli, v koreňovom priečinku `GLNetworking`, ktorý obsahuje súbor `Podfile`, je nutné stiahnuť a nainštalovať závislosti pomocou príkazu `pod install`. Zostavenie a spustenie projektu knižnice je potrebné vykonať vo vývojovom prostredí `XCode`, prostredníctvom tlačidla `spustiť`, prípadne pomocou dvoj-krokového postupu `Product > Build For > Running` a následne `Product > Run`.