VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE PRO NOSITELNOU ELEKTRONIKU SE SYSTÉMEM APPLE WATCHOS
APPLICATION FOR WEARABLES RUNNING APPLE WATCHOS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Kevin Singh

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Pavel Mašek

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Kevin Singh ***ID:*** 162255
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Aplikace pro nositelnou elektroniku se systémem Apple WatchOS

POKYNY PRO VYPRACOVÁNÍ:

Specifickou oblast IoT představují v dnešní době takzvané nositelnosti (wearables), které jsou velmi často reprezentovány chytrými náramky nebo hodinkami. V rámci této bakalářské práce bude pozornost soustředěna na platformou Apple Watch a vývoj aplikace pro zjištění základní informací o spárovaného mobilního telefonu (zjišťování informací o parametrech WiFi připojení, zobrazení aktuální polohy s využitím GPS, notifikace o stavu baterie, atd.). Zjištěné hodnoty budou následně uloženy do vytvořené databáze pro pozdější vyhodnocení.

DOPORUČENÁ LITERATURA:

[1] DANIEL, Steven F. 2016. Apple Watch App Development: Build real-world applications for the Apple Watch platform using the WatchKit framework and Swift 2.0. ISBN: 978-1-78588-636-2.

[2] BUTTFIELD-ADDISON, Paris, Jon MANNING a Tim NUGENT. 2016. Learning Swift: Building Apps for OS X and IOS. O'Reilly Media, Inc. ISBN: 978-1-49194-074-7.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. Pavel Mašek
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Bakalářská práce „Aplikace pro nositelnou elektroniku se systémem Apple WatchOS“ se zabývá teoretickým rozborem komunikací M2M (Machine-to-Machine), H2H (Human-to-Human), D2D (Device-to-Device) a nositelnou elektronikou. V teoretické části jsou dále popsány chytré hodinky Apple Watch, jenž obsahují požadovaný systém WatchOS. V rámci bakalářské práce vznikly dvě aplikace, přičemž první byla vytvořena pro mobilní zařízení iPhone a druhá pro chytré hodinky Apple watch. Aplikace zobrazují informace o připojení k Internetu a poloze uživatele na mapě. Aplikace pro zařízení iPhone navíc umožňuje získaná data ukládat do lokální, ale i serverové databáze.

KLÍČOVÁ SLOVA

Machine-to-Machine, Human-to-Human, Device-to-Device, WatchOS

ABSTRACT

Bachelor work „Application for wearable electronics with Apple WatchOS system“ deals with theoretical analysis of M2M (Machine-to-Machine), H2H (Human-to-Human) and D2D (Device-to-Device) communication and wearable electronics. Later in the theoretical part, smart watch Apple Watch is described, which contains required WatchOS system. During compilation of this bachelor work were created two applications. One was made for iPhone and a second for Apple Watch. Both applications are able to show information about Internet connection and the user´s location. The application for iPhone is also able to store data in the local and server´s database.

KEYWORDS

Machine-to-Machine, Human-to-Human, Device-to-Device, WatchOS

PROHLÁŠENÍ

Prohlašuji, že svůj semestrální projekt na téma „Aplikace pro chytré hodinky se systémem WatchOS“ jsem vypracoval(a) samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedeného semestrálního projektu dále prohlašuji, že v souvislosti s vytvořením tohoto semestrálního projektu jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Pavlu Maškovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych chtěl poděkovat Ing. Rastislavovi Červenákovi za cenné rady a své rodině za podporu.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v tomto semestrálním projektu byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

ÚVOD

Počet komunikujících zařízení neustále roste, přičemž rozsáhlou skupinu z těchto zařízení představuje nositelná elektronika. Ta komunikuje s připojeným mobilním telefonem a je schopna posílat data například o tepové frekvenci, počtu spálených kalorií atd. Dle analytiků společnosti IDC bude v roce 2019 prodáno přes 200 milionů kusů zmíněné technologie [1]. Z nositelné elektroniky se zejména prosazují chytré hodinky a náramky, ale vývoj je směrován i na brýle či oblečení.

Nositelná elektronika je vedena jako podskupina komunikace D2D (Device-to-Device), která je podskupinou M2M (Machine-to-Machine). Z tohoto důvodu je zde uvedena definice jednotlivých komunikací, přičemž u komunikace M2M je navíc provedeno srovnání s komunikací H2H (Human-to-Human).

Většina chytrých hodinek obsahuje velké množství funkcí. Na trhu se nejvíce prosazují společnosti jako jsou Fitbit, Xiaomi a Apple [18]. Poslední zmíněná společnost nabízí hodinky, jenž se nazývají Apple Watch a obsahují systém WatchOS. Cílem této práce je tedy vyvinout aplikaci pro toto zařízení, která bude zpřístupňovat informace o poloze uživatele a připojené bezdrátové síti Wi-Fi. Tyto informace budou obsahovat nejen zeměpisnou délku a šířku, ale také zobrazení polohy na mapě.

Z informací o síti bude k dispozici IP adresa, SSID (Service Set Identifier), Gateway (brána) a maska sítě. Celkově se informace budou ukládat do databáze, kde bude možnost nahlédnout na podrobnosti internetového připojení, které budou uloženy na základě polohy. Získaná data budou uložena na mobilním zařízením. Aplikace bude vyvíjena v jazyce Swift a využito bude vývojové prostředí Xcode.

1 POPIS TYPŮ KOMUNIKACÍ V MOBILNÍCH SÍTÍCH

V této kapitole jsou stručně popsány typy komunikací. Konkrétněji se jedná o M2M (Machine-to-Machine), H2H (Human-to-Human) a D2D (Device-to-Device), která je uvedena jako podskupina komunikace M2M. Ke každému uvedenému druhu je uvedená stručná teorie a příklady využití. V závěru této kapitoly je provedeno srovnání technologií H2H a M2M.

1.1 Machine-to-Machine

M2M neboli Machine-to-Machine je typ komunikace, kde není potřeba zásahu člověka. Obecně můžeme říct, že slouží ke stanovení podmínek pro výměnu informací mezi dvěma a více zařízeními přes komunikační síť. Ideálním příkladem může být zařízení, jenž si vyměňuje data s určitou business aplikací právě přes zmíněnou síť. Nejdůležitější pozice patří síti, neboť bez ní by nemohla vzniknout tato komunikace. M2M je však více než konektivita, neboť zahrnuje mnoho různých komponent, které umožňují vzdálenou i automatickou komunikaci se vzdáleným zařízením. V souhrnu se tedy také jedná o senzory v přístroji ovládacího softwaru, komunikačního modulu, centrálního systému, apod. V praxi se s tím můžeme například setkat u automobilů (navigace, zabezpečení,...). V dnešní době se M2M také prosazuje ve zdravotnictví, kdy může například doktor získat informace do svého počítače z chytrého zařízení pacienta, které sbírá jeho zdravotní hodnoty (např. tepová frekvence, krevní tlak, atd.). Využití tohoto typu komunikace je tedy velmi rozšířené a to zejména u provozovatelů mobilních sítí, kteří se odkazují na možnosti bezdrátové komunikace na bázi SIM karet. Dle odhadu firmy Cisco bude do roku 2020 počet zařízení využívající M2M přes 3,1 miliardy kusů [2].

1.1.1 Machine-to-Machine a Internet of Things

Definice M2M a IoT (Internet of Things) bývá často matoucí, neboť se může zdát, že tyto způsoby komunikace jsou stejné. Liší se však v mnoha ohledech.

Oproti M2M, které lze také znázornit jako komunikaci One-to-One, vytvoří IoT síť, umožňující komplexnější komunikační prostředí. Navíc se u komunikace M2M jedná o určitý typ zařízení, které je monitorováno a ovládáno centrálním systémem. Naopak o Internetu věcí lze tvrdit, že prostřednictvím technologií a sítí umožňuje propojení různých typů zařízení a integraci dalších informačních zdrojů, které spolu dokáží komunikovat [3].

1.1.2 Machine-to-Machine s bezdrátovou technologií

Zatímco se s použitím bezdrátové M2M technologie spoléhalo na ZigBee bezdrátovou komunikační technologii amobilní sítě 2G/3G, stále více se prosazují technologie, jako jsou Wi-Fi a Bluetooth. Mezi další používané komunikační technologie patří WLAN, Wireless, MBUS, NB-IoT, LoRa, SigFox a mobilní sítě čtvrté generace LTE (Long Term Evolution). Srovnání technologií ZigBee a Wi-Fi je uvedeno níže:

Tab. 1.1: Porovnání ZigBee a Wi-Fi [4].

Typ technologie	ZigBee	Wi-Fi
Šířka pásma	1 MHz	2 MHz
Dosah signálu	od 10 m do 30 m	od 30 m do 100 m
Rychlost přenosu	250 kbps	11 mbps
Životnost baterie	několik měsíců	jednotky hodin

- Technologie ZigBee je navrhnuta k přenesení malého množství dat na krátkou vzdálenost, přičemž oproti Wi-Fi spotřebuje velmi málo energie [4].
- Srovnání baterie u Wi-Fi je v případě využití telefonu, který používá uživatel denně. Výdrž je tedy potom přibližně 10 hodin [4].
- Rychlost přenosu dat u technologie Wi-Fi je znázorněna u standardu IEEE 802.11b [4].

Obě technologie mají své pozitvní, ale i negativní stránky. S použitím Wi-Fi je možné získat lepší využití pásma a přenosové rychlosti, ale také podstatně nižší výdrž baterie. Naopak u použití technologie ZigBee je výdrž zařízení na baterii mnohem delší, ale také patrná ztrátovost především v rozsahu signálu [4].

1.1.3 Zařízení bez obsluhy

Několik miliónů různých zařízení posílá a přijímá data pomocí Internetu, mezi kterými jsou zahrnuty i senzory, aktuátory apod. Jelikož komunikace M2M probíhá pouze mezi dvěma zařízeními, jedná se o komunikaci bez lidského zásahu. Z tohoto důvodu vznikají obrovské nároky na správu velkého množství zařízení, jenž propouští pouze zlomek dat v celé řadě rozvíjejících se aplikací. Řešením se tak nabízí opírání se o specificky vzdálený subjekt infrastruktury s podporou automatického zabezpečení a vzdálenou konfiguraci M2M zařízení, rekonfiguraci a dalších aktualizacích zabezpečení operací [5].

1.2 Human-to-Human

Jak již ze zkratky H2H (Human-to-Human) vyplývá, jedná se o komunikaci mezi dvěma lidmi, která probíhá přes určité zařízení. Jako způsob takové komunikace lze uvést telefonování, posílání SMS zpráv, e-mailů apod. Nutnost tedy spočívá v tom, že na obou koncích komunikace musí být člověk.

Požadavky u komunikace H2H jsou [6]:

- Vysoká rychlost přenosu dat.
- Mobilita.
- Kvalita služby QoS (Quality of Service) a kvalita zážitků QoE (Quality of Experience).

1.3 Device-to-Device

Díky zavádění nových mobilních komunikačních technologií, jako jsou LTE či Wi-MAX systémy, je trh se službami doslova nasycen. Kvůli tomuto problému bylo zapotřebí přijít s novým obchodním modelem mobilních služeb. Od těchto nových modelů se očekává nejen zlepšení ziskovosti poskytovatelů služeb a dodavetelů, ale také rostoucí poptávka po mobilním obsahu multimediálních služeb [7].

Device-to-Device komunikace je považována za slibnou budoucnost mobilních komunikačních služeb. Mezi komunikacemi D2D a M2M lze vidět určitou podobnost. Komunikace D2D je taková komunikace, která umožňuje přímou komunikaci mezi dvěma blízkými zařízeními v rámci mobilní sítě, Bluetooth či BLE a komunikace M2M je na druhou stranu taková, u které není zapotřebí k plnění funkce člověk [7].

Mezi nejrozšířenější komunikační technologie u D2D patří především BLE a Wi-Fi Direct. Dále je to LTE (které ale bývá použito spíše jako záložní řešení pro sestavení datového spojení a to z důvodu energetické náročnosti), Wi-Fi, WiGig (20/60/72 GHz), Bluetooth a LTE Direct. Přímá komunikace LTE je však zatím dostupná pouze na území Spojených států.

Využití přímé komunikace mezi mobilními zařízeními zlepší využití rádiového spektra, celkovou propustnost a energetickou účinnost. V případě komunikace D2D s LTE Direct je využití frekvenčního spektra velmi důležité a má potenciál stát se konkurenceschopnou záložní veřejné bezpečnostní sítě, které musí fungovat v případě, kdy nejsou k dispozici mobilní sítě [7].

Jako příklad využití D2D můžeme uvést komunikaci mobilního telefonu s chytrými zařízeními (chytré hodinky, náramky apod.).

1.4 Srovnání komunikací H2H a M2M

V tabulce 1.2 uvedené níže jsou srovnány důležité parametry komunikací H2H a M2M. Z tabulky je zřejmé, že nároky na provoz komunikací H2H a M2M jsou zcela odlišné [8].

Tab. 1.2: Porovnání H2H a M2M [8].

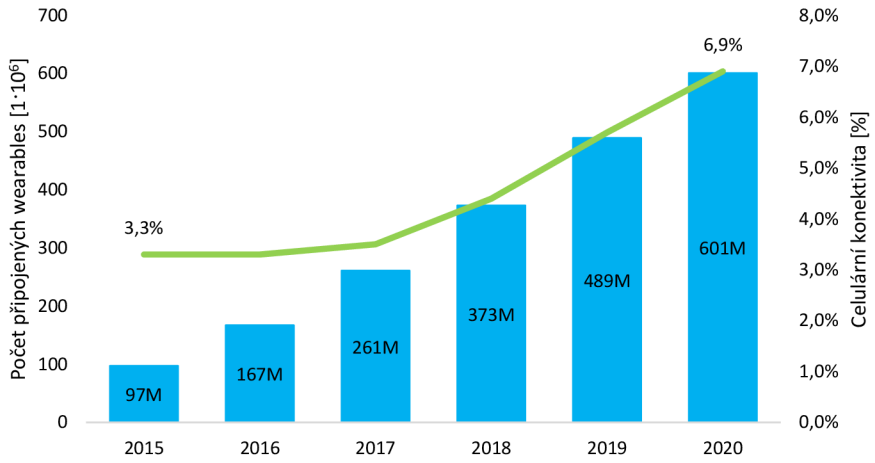
Typ parametru	H2H	M2M
Počet zařízení	Neustále se zvýšující. Ne však tak, jako u M2M.	Převýšení lidských koncových uživatelů. Jeden z důvodů zavedení IPv6.
Objem dat	Velká většina provozu představuje stahování (download) a vyžaduje významné množství šířky pásma.	Provoz je tvořen především z nahrávání (upload). Využita malá šířka pásma.
Napájecí baterie	V případě potřeby je možné koupit baterii novou. Je zde tedy možnost výměny.	Musí být soběstačné po dlouhou dobu.
Zpoždění	Určitá tolerance i v případě hlasové komunikace.	Některé z aplikací jsou určeny především pro řízení v reálném čase. Nouzové akce musí mít malou toleranci (Delay, jitter).
Zabezpečení	Lze poznat, že zařízení je odcizeno.	K dispozici je robustní zabezpečení i utajení.
Příjem	Dobrý.	Nízká ARPD (Average Relative Percentage Deviation).
Dimenzování	Nepotřebuje speciální přístup.	Potřebné nadimenzování, aby se předcházelo přetížení.

2 NOSITELNÁ ELEKTRONIKA

Tato kapitola obsahuje popis nositelné elektroniky, která je zde uvedena jako podskupina D2D. Dále jsou zde uvedeny informace ohledně typu chytrých hodinek Apple Watch, které byly vybrány v rámci bakalářské práce. Nakonec je vložena ukázka z vývojového prostředí Xcode.

2.1 Teoretický popis nositelné elektroniky

Elektronická zařízení, která mohou být nošena člověkem, nesou název Wearables (nositelná elektronika). Tímto označením však nejsou myšlena zařízení jako mobilní telefony či MP3 přehrávače, ale různé náramky, hodinky, brýle, kamery a oblečení, které mají možnost propojení s chytrým telefonem či tabletem. Díky svým možnostem a rozměrům se stávají čím dál tím více populární. Dle odhadů společnosti Cisco bude v roce 2020 počet chytrých zařízení větší než 601 milionů, což by bylo pětkrát více než v roce 2015 [9].



Obr. 2.1: Odhad společnosti Cisco pro počet chytrých zařízení [9].

2.1.1 Možnosti nositelné elektroniky

V dnešní době obsahují chytré hodinky či náramky plnou řadu senzorů, které dokáží poskytnout uživatelům nejrůznější informace. Díky těmto senzorům můžeme například získat počet našich kroků, spálené kalorie, ale i ušlou vzdálenost či srdeční puls. Pokročilejší přístroje toho umí však mnohem více. Dokážou například

reagovat na telefonní hovory, odpovídat na zprávy a e-maily. S možností propojení s dalším chytrým zařízením se otevírají dveře i aplikacím třetích stran. Ty mohou práci usnadnit už jen tím, že uživatel nemusí svůj mobilní telefon tak často používat a vystačí si v rámci možností jen s nositelnou elektronikou.

Obr. 2.2: Možnosti nositelné elektroniky [10].

2.2 Popis typů nositelné elektroniky

Nositelná elektronika je dnes prezentována v mnoha formách. Mezi nejčastější zařízení patří náramky a hodinky, pozadu však ale nezůstávají například brýle či dokonce oblečení.

2.2.1 Chytré náramky

Chytré náramky patří mezi nejrozšířenější nositelnou elektroniku. To je způsobeno zejména tím, že zpravidla mají mnoho funkcí za příznivou cenu. Většina modelů je též vodotěsná a jsou tedy vhodné i pro plavce. Náramky je možné pořídit i s displejem, který umožňuje nejen přehled o naměřených hodnotách, ale také o času.

Příklady chytrých náramků

- **Fitbit Flex2** - Zmíněný náramek patří k tomu nejlepšímu, co je možné na trhu dostat. Uživateli poskytuje informace o spálených kalorií, ušlé vzdálenosti, ale také počet minut, po jaké byl uživatel aktivní. Oproti většině konkurence dále nabízí měřič kvality spánku a voděodolnost. S náramkem je tedy možné se potápět a měřit plavecké výkony. Zákazníky především láká na dlouhou výdrž

baterie (až 5 dní) a tracker, který je možný z náramku vyjmout a nosit jej například jako přívesek na řetízku [11].

- **Xiaomi MiBand 2** - MiBand 2 vychází z modelu MiBand 1S, oproti kterému má navíc OLED displej. I přes toto vylepšení je výdrž baterie s kapacitou 70 mAh přibližně 20 dní. Stejně jako Flex 2 umožňuje sledovat uživatelův pohyb a spánek. Není však vodotěsný [12].

2.2.2 Chytré hodinky

Na první okamžik je možné nabít dojmu, že se od chytrých nárámků s displeji nijak neliší. Rozdíl však spočívá v tom, že slouží jako plnohodnotný doplněk chytrého telefonu, který dokáže zrcadlit mnoho jeho funkcí. Je tedy možné telefonovat, posílat textové zprávy, ale také ovládat chytrou domácnost apod. Oproti náramkům je tedy displej dotykový, což umožňuje daleko větší využití.

Příklady chytrých hodinek

- **Samsung Gear S3** - Jedná se o hodinky s tělem z nerezové oceli, které umožňují výměnu pásku hodinek a změnu ciferníku. Spárování s mobilními telefony je možné pro smartphony značky Samsung, které mají alespoň 1,5 GB RAM a systém minimálně Android 4.3. Hodinky obsahují bezkontaktní technologii NFC (Near Field Communication), která umožňuje provést platbu pomocí hodinek [13].
- **Garmin fenix 3** - Hodinky jsou zaměřené na náročné uživatele a jejich největší potenciál spočívá v GPS díky vysoce citlivému GPS/GLONASS přijímači s ocelovou anténou. Hodinky jsou plně voděodolné a umožňují tak provádět například plavecké tréninky. Mimo jiné poskytují SWOLF (Swim Golf) hodnotu, počet záběrů a dokonce rozpoznají i plavecký styl. S aktivním GPS je výdrž hodinek v rozmezí 16 až 50 hodin. Model podporuje Bluetooth, Wi-Fi a ANT+ [14].

2.2.3 Oblečení

Již dnes je možné pořídit různé typy oblečení, které umožňuje uživateli např. provést masáž, která slouží k eliminaci stresu a upozornění na špatné držení těla.

Příklad chytrého oblečení

- **AiraWear** - Jedná se o masážní oblečení, které také pomáhá ke správnému držení těla. Oblečení je tvořené z bavlny a polyesteru. Masážní program je možné nastavit pomocí aplikace pro chytrý telefon. Počítá se s podporou iOS

a Android. Plánované spuštění prodeje chytrého oblečení je plánované ke konci roku 2017 [15].

2.2.4 Budoucnost

Nositelná elektronika je spjata s mnoha možnostmi využití. Vzniká mnoho zajímavých projektů a to i v případě zdravotnictví. Jeden z těchto projektů jsou kontaktní čočky, které budou kromě opravy očí hlídat stav pacienta postíženého cukrovkou [16].

2.3 Apple Watch

V současné době existují 2 verze, přičemž jsou obě k dostání ve dvou typech rozměrů displeje (38 mm a 42 mm). V důsledku propojení s mobilním telefonem obsahují hodinky širokou paletu funkcí a podporu aplikací třetích stran. Z tohoto důvodu je tedy možné reagovat na telefonní hovory, textové zprávy, ale i ovládat hudební přehrávač či sledovat sociální sítě. Mezi další užitečné funkce je možné zahrnout navigování pomocí GPS, které se však nachází v mobilním zařízení (platí pouze pro Apple Wach Series 1). Mimo jiné obsahuje „základní“ funkce, jako jsou počítání kalorií, kroků atd. Propojení s mobilním telefonem je možné pomocí technologie Bluetooth či Wi-Fi.

Obr. 2.3: Hodinky Apple Watch [17].

2.3.1 Podíl na trhu

Při příchodu na trh se staly prvními chytrými hodinkami, které se začaly prodávat na globálním trhu v obrovském množství a ovládly tak velkou část trhu. V aktuální době však jejich podíl na trhu klesá. To je dáno především tím, že se na trh dostává masové množství levnějších a jednodušších chytrých hodinek. Jako jednodušší jsou označovány podle toho, zda podporují aplikace třetích stran. Jejich rozšíření dokazuje fakt, že za druhý kvartál roku 2016 představují přibližně 86% všech prodaných chytrých hodinek. V tabulce 2.1 je znázorněn podíl pěti velkých značek na trhu v letech 2015 a 2016. Jak je z tabulky patrné, prodej hodinek Apple klesl meziročně o 56,7% [18].

Tab. 2.1: Podíl na trhu [18].

Prodejce	2Q16	2Q2016 Market	2Q2015	2Q2015	2Q2015
Fitbit	5.7	25.4%	4.4	24.9%	28.7%
Xiaomi	3.1	14.0%	3.1	17.2%	2.5%
Apple	1.6	7.0%	3.6	20.3%	-56.7%
Garmin	1.6	6.9%	0.8%	4.2%	106.7%
Lifesense	1.0	4.6%	0.0	0.0%	N/A
Ostatní	9.5	42.1%	5.9	33.3%	59.3%
Celkem	**22.5**	**100%**	**17.8%**	**100%**	**26.1%**

2.3.2 WatchOS

Hodinky fungují na svém vlastním operačním systému, jenž se nazývá WatchOS. Nejnovější verze tohoto systému se nazývá WatchOS 3 a oproti předchozím verzím nabízí několik vylepšení.

WatchOS 1

Již první verze poskytovala bohaté množství vlastností. Mezi ty nejzásadnější patří funkce Siri (umělá inteligence systému, jenž uživateli znatelně usnadňuje práci) a podpora aplikací třetích stran. Aplikace však znatelně zpomalovaly chod systému [19].

WatchOS 2

Druhá verze přináší mnoho nových funkcí, dále však vylepšuje funkce u verze první. Nyní již není podmínkou mít hodinky připojené s mobilním telefonem pomocí Bluetooth, ale stačí je mít propojené pomocí připojení na stejnou síť Wi-Fi. Mezi další

přidané funkce patří možnost propojení hodinek s větším počtem mobilních zařízení [20].

Novinky:

- Přímá instalace aplikací do hodinek.
- Animované ciferníky.
- Noční režim.

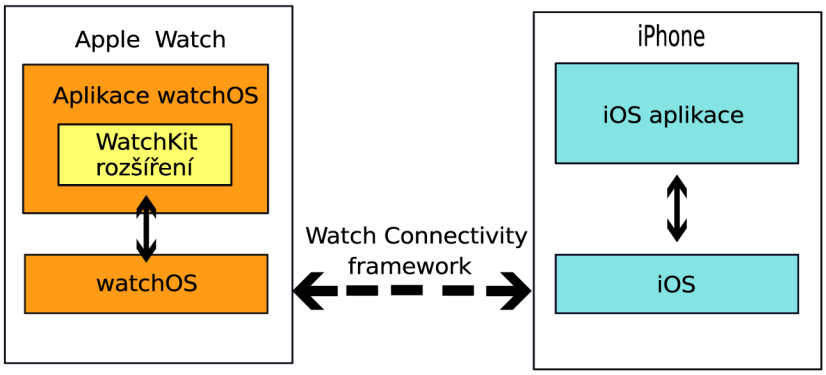
WatchOS 3

Aktuální verze zrychluje chod aplikací. Dále přichází s novými možnostmi a aplikacemi, které jsou především zaměřené na zdraví uživatele a podporu pro chytrou domácnost [21]. Novinky [21]:

- SOS tlačítko.
- Běh aplikací na pozadí.
- Pět nových předinstalovaných aplikací.
- Nové ciferníky.

Komunikace WatchOS s iOS

Při spuštění aplikace na hodinkách ji WatchOS uvádí do popředí, dokud ji uživatel nezruší. Pokud aplikace není v popředí, běží periodicky na pozadí. V tomto případě je aplikace v doku a je schopna posílat notifikace. Mimo to zůstávají funkce aplikace v paměti, což umožňuje rychlejší spuštění stejné aplikace [22].
Komunikace aplikace s telefonem zahrnuje dvě důležité funkce, svazek WatchApp a framework WatchKit. Průběh komunikace hodinek s mobilním zařízením je zobrazen na obrázku níže [22].



Obr. 2.4: Architektura komunikace [22].

V době startu přejde aplikace ze stavu neaktivního do stavu aktivního. Pokud uživatel ukončí aplikaci, dojde opět k přechodu do neaktivního stavu, ale poběží na pozadí a funguje jako aplikace pozastavená. Nastane-li situace, že je aplikace v dokovací stanici či jedna z jejich funkcí je zobrazena na aktivním ciferníku, nastane též pozastavený stav [22].

2.3.3 Apple Watch Series 2

Po dvou letech po vydání první série přichází nový model chytrých hodinek společnosti Apple nesoucí název Apple Watch Series 2. Po vzhledové stránce je zařízení stejné, ale po stránce hardwarové došlo k mnoha vylepšení. Osazeno je procesorem S2, který umožňuje vykreslit 60 snímků za sekundu, a je tak o 50% rychlejší než procesor S1. GPU nové verze má dvakrát vyšší výkon a tím je umožněn rychlejší chod systému. Procesor navíc obsahuje GPS modul a tím se hodinky stávají méně závislé na mobilním telefonu. Dalšího vylepšení se dočkala baterie, která má o 32% delší výdrž [23].

2.3.4 Porovnání verzí Apple Watch

V následné tabulce je znázorněné srovnání obou verzí. Rozměry i vzhled zůstávají stejné.

Tab. 2.2: Porovnání verzí Apple Watch [24].

Typ modelu	Apple Watch Series 1	Apple Watch Series 2
GPS modul	Ne	Ano
Procesor	S1P	S2
CPU	520MHz	Dvoujádrový
Wi-Fi modul	802.11b/g/n	802.11b/g/n
Bluetooth	Bluetooth 4.0	Bluetooth 4.0
Voděodolnost	Ne	Ano
Displej	OLED retina	OLED retina
Rozlišení (38mm)	272 x 340	272 x 340
Rozlišení (42mm)	390 x 312	390 x 312
Váha	50g	70g

2.4 Vývojové prostředí Xcode

Jedná se o vývojové prostředí specializované na vývoj softwarových aplikací pro produkty značky Apple. Podporované platformy pro vývoj jsou Mac OS X, iOS (iPhone, iPad), WatchOS a nově i tvOS. Od jeho počátku již vzniklo několik verzí, přičemž každá přinesla značná a důležitá vylepšení. Momentálně nějnovější verze nese označení 8. Především je používáno pro vývoj aplikací prostřednictvím programovacího jazyka Swift, mimo něj však podporuje i další jazyky, jakou jsou C, C++, Objective-C, Objective-C++, Java, AppleScript, Python a Ruby. V rámci Xcode je také možné pracovat s Cocoa a Cocoa Touch frameworky [25].

2.4.1 Interface builder

Jednoduchou cestou umožňuje uživateli rozvrhnout funkce své aplikace bez nutnosti psaní kódu. Je tak možné přidat různá tlačítka, textová pole a další objekty, ke kterým lze poté přiřadit jednotlivé třídy. K jednotlivým obrazovkám a funkcím lze také přiřadit design a animace [25].

2.4.2 Swift

Jedná se o moderní programovací jazyk určený pro vývoj na platformy macOS, iOS, WatchOS a tvOS. Vznikl v roce 2014 a akutální verze nese označení Swift 3. Syntaxí je velmi podobný jazyku C. Původně byl zamýšlen jako náhrada za programovací jazyk Objective-C, který by měl být mnohem jednodušší a pro uživatele vhodnější. Od mnoha jazyků se liší tím, že nevyžaduje použití středníků u příkazů a závorek. Od verze 2.0 je navíc open-source, což umožňuje upravení a využití jazyka na jiných platformách.

2.4.3 iOS simulátor

Xcode umožňuje uživateli spouštění a testování vyvíjených aplikací na reálném zařízení. Pokud jej však uživatel nevlastní, lze využít aplikaci iOS simulator. Ten je součástí Xcode a dovoluje počítači simulovat iPhone, iPad, Apple Watch, iMac apod. Uživateli při testování dovoluje simulovat doteky či fyzické otočení jako u normálního zařízení [25].

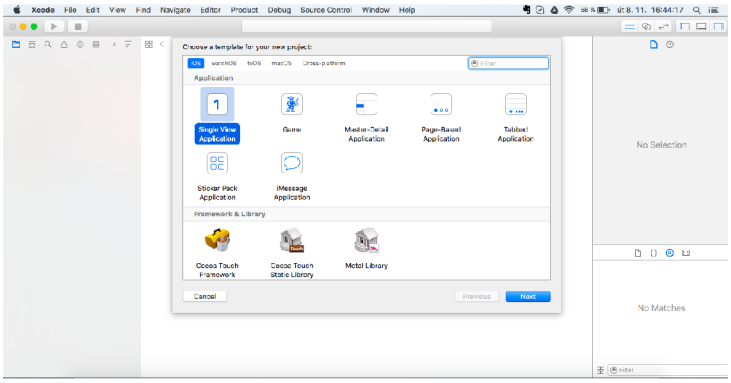
2.5 Ukázka vývojového prostředí

Níže je provedena jednoduchá ukázka vývojového prostředí Xcode a iOS simulátoru. Jedná se o stručný postup, jehož cílem je zobrazit text “Hello World!” na chytrých

hodinkách Apple Watch. Demonstrace je znázorněna pomocí čtyř kroků.

Krok první

Pokud se jedná o vývoj aplikace pro Apple Watch, je nutné udělat aplikaci i pro mobilní zařízení. V prvním kroce je potřeba otevřít vývojové prostředí Xcode a vytvořit nový projekt. V následném okně je důležité vybrat objekt nesoucí název *Single View Application*. V dalším okně je doporučeno projekt vhodně pojmenovat, například *Hello World!*
file -> new -> project -> Single View Application.



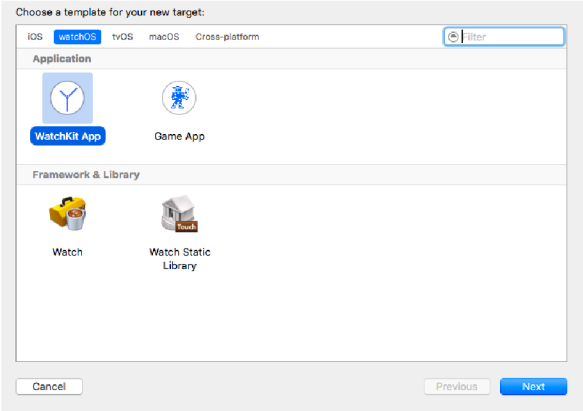
Obr. 2.5: Založení projektu.

Krok druhý

Tímto krokem je zhotoveno vývojové prostředí pro mobilní zařízení. Nyní je potřeba přidat vývojové prostředí pro hodinky. V tomto případě už není nutno přidávat *Single View Application*, ale *WatchKit App*, které slouží pro vývoj a nastavení aplikace pro hodinky. U pojmenování je třeba na jiný název, aby bylo možné odlišit, co k čemu patří. Pojmenování může být například *Hello*.
file -> new -> target -> WatchKit App.

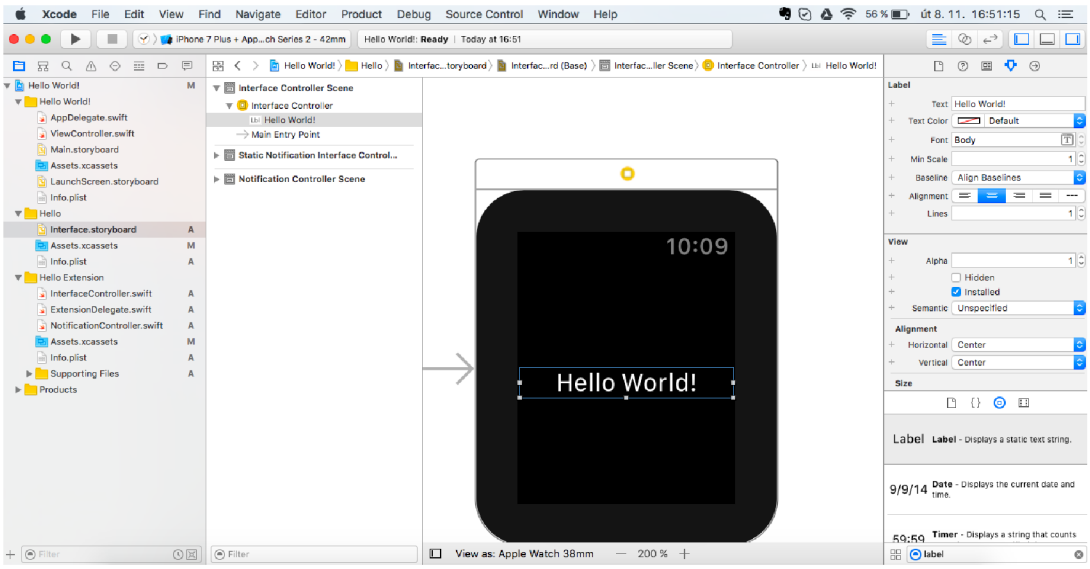
Krok třetí

Nyní je vše připraveno pro samotný vývoj aplikace. Ve struktuře aplikace je potřebné vybrat složku *Hello* a v ní soubor se jménem *Interface.storyboard*. Následně se zobrazí obrazovka hodinek, která poskytuje rozvržení aplikace.
Ve spodní části programu lze nalézt výběr volby displeje hodinek, kde je možnost nastavení velikosti 42 mm. Z knihovny objektů (vpravo dole) je nutné vybrat objekt *label* a vložit jej do obrazovky, tedy do prostředí. Narozdíl od prostředí na mobilním



Obr. 2.6: Přidání Watchkit App.

zařízení není možné objekt vložit kamkoliv, ale pouze na určitá místa. Nyní je ale možné objekt vložit kamkoliv, neboť jeho umístění se upraví manuálně.
Ve vlastnostech objektu lze upravit text na požadované „*Hello World!*“. Pro přehlednější vzhled je ještě možné nastavit zarovnání objektu s textem na střed, což lze provést ve spodní části vlastností objektu.



Obr. 2.7: Umístění objektu *label*.

Krok čtvrtý

Nyní přichází na řadu simulátor. Důležité je vybrat jako zařízení hodinky, které lze najít pod označením *Hello.extension*. Výběr je možné provést v horní části programu,

kde je mimo jiné volba simulátoru i pro telefon, či obě zařízení dohromady.
Poté už jen stačí stisknout v levé horní části tlačítko (*run*) ke spuštění simulátoru a dojde k jeho spuštění. Jakmile proběhne instalace aplikace na simulátoru, zobrazí se dvě nová okna. Jedno okno bude zobrazovat simulovanou obrazovku mobilního zařízení a druhá obrazovku hodinek, kde dojde k zobrazení požadovaného obsahu. Na obrazovce telefonu bude zobrazena jen hlavní nabídka, neboť zde se nic nevytvářelo.

Obr. 2.8: Zobrazení textu „Hello World!“.

3 DEFINICE PLÁNOVANÉ APLIKACE

V této kapitole je zahrnuta definice plánované aplikace, popis programovacího jazyka Swift a mobilního zařízení iPhone. Dále je zde znázorněn její prvotní design.

3.1 Popis aplikace

Výstupem bakalářské práce by měla být aplikace na chytré hodinky Apple Watch, která bude poskytovat tyto funkce:

- Poskytování informací ohledně připojené Wi-Fi.
- Poskytování polohy pomocí GPS.

Z důvodu absence GPS modulu je potřeba vytvořit další aplikaci pro mobilní zařízení, která umožní požadované informace získat. Uvedené hodinky dokáží komunikovat pouze s takovým mobilním zařízením, které pracuje na systému iOS. V současné době existuje pouze jeden typ takového telefonu, jenž nese název *iPhone*.

Komunikace mezi hodinkami a telefonem tedy bude probíhat na základě žádostí a odpovědí. Chytré hodinky pošlou žádost do telefonu, který zpětně odešle požadovanou odpověď.

3.1.1 Design

Protože je na hodinkách relativně malý displej, mělo by být menu co nejpřehlednější. Z tohoto důvodu je doporučené umístit do hlavní nabídky dvě tlačítka s dostatečnou velikostí, načež se kliknutím na určité tlačítko zobrazí jeho vlastní menu.

Na **první pozici** se bude nacházet tlačítko se jménem *Wi-Fi info*. V nabídce je umístěno 8 *labelů*, přičemž hodnoty vypisují pouze čtyři z nich. Zbylé *labely* budou mít tedy pevné pojmenování. Dále se ve spodní části nachází tlačítko, které slouží ke zobrazení požadovaných hodnot.

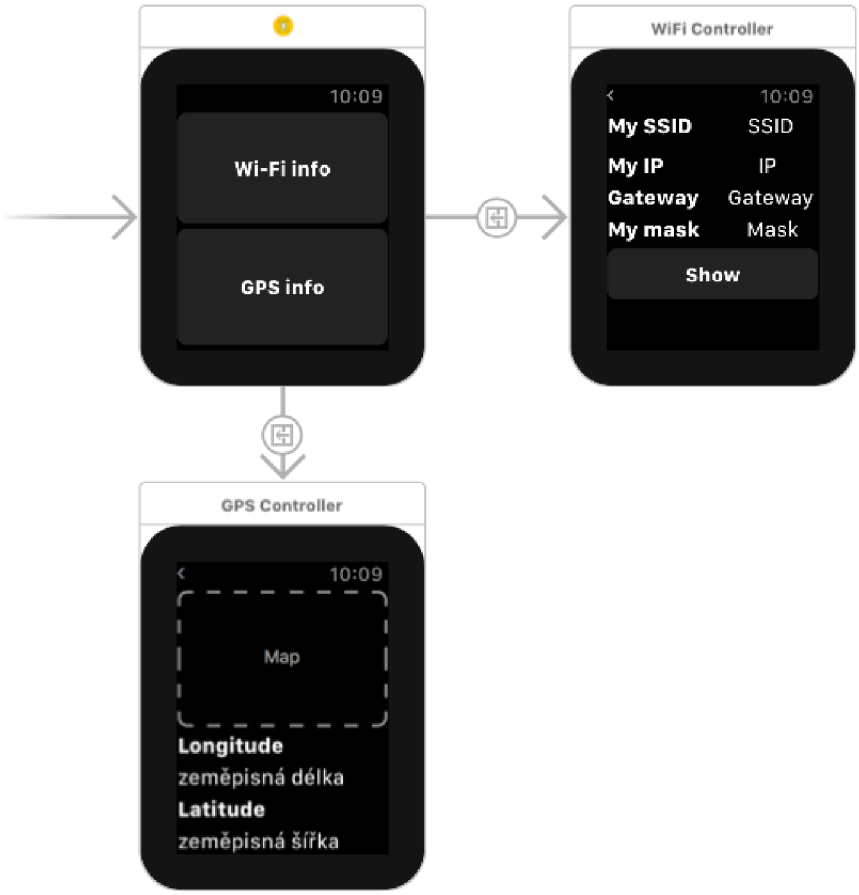
Druhá pozice ponese jméno *GPS info*. Zde bude zobrazena mapa, která znázorní lokaci uživatele a pod mapou budou opět *labely*, které vypíší zeměpisnou šířku a výšku.

Předběžný design aplikace, který byl vytvořen pouze v Xcode, je znázorněn níže na vývojovém diagramu. Návrh dále poukazuje na jednotlivé propojení obrazovek (obr. 3.1).

Sketch

Jedná se o vektorový editor určený pouze pro platformu MAC OS, který je určen pro vývoj webového a mobilního designu. Jeho síla spočívá v efektivním využití

ovládacích prvků. Aktuální verze nese označení *Sketch 3*, ale novější verze je již očekávána [26]. Tento program bude použit pro finální provedení designu aplikace.



Obr. 3.1: Prvotní design aplikace.

4 REALIZOVANÁ APLIKACE

Tato část je věnována samotnému vývoji aplikace. Jsou zde uvedeny a popsány ukázky zdrojového kódu. Dále je zde uvedena tvorba designu aplikace.

4.1 Seznámení s objekty v projektu

Nejprve je potřeba založit nový projekt ve vývojovém prostředí Xcode. Projekt je potřeba založit způsobem, jako v ukázce vývojového prostředí v teoretické části. Pro mobilní zařízení:

File -> new -> project -> iOS -> Single View Application.

Po založení projektu se ve složce zobrazí třídy, které slouží pro základní vývoj aplikace. Definice těchto tříd je uvedena níže:

- AppDelegate.swift - jedná se o třídu, která se spustí jako první. Obsahuje tedy funkce a frameworky, které jsou zapotřebí k běhu celé aplikace.
- Main.storyboard - zde dochází k navrhnutí a vývoji designu aplikace. Je zde znázorněna plocha zvoleného zařízení (v případě bakalářské práce iPhone 7 plus), do které je možné implementovat různá tlačítka, textová pole, atd.
- ViewController.swift - aby bylo možné přidělit funkce tlačítkům a dalším objektům v Main.storyboard, je potřeba provést jejich deklaraci v určité třídě. Při implementování těchto objektů do třídy je důležité myslet na funkci objektu. Při „natažení“ objektu je povinné zvolit, zda bude přidán jako funkce (tlačítko) či jen zobrazovací pole (label/textové pole).
  Je potřeba mít na mysli, že každý storyboard má přiřazenou pouze jednu třídu. Není tedy možné používat pro všechny plochy jen jeden Controller.
- Assets.xcassets - jedná se o složku, do které se ukládají veškeré obrázky, jenž budou v aplikaci použity (pro mobilní zařízení). Tyto obrázky je ale také možné napsat pomocí programovacího jazyku Swift. Výhoda spočívá ve změně velikosti, aniž by byla narušena grafika určitého objektu, či ve snažší změně barev.
- LaunchScreen.storyboard - Zastává stejnou funkci jako Main.storyboard. Rozdíl ale spočívá v tom, že tato obrazovka se zobrazí pouze při načítání aplikace. Nejpoužívanější využití tedy spočívá k zobrazení loga společnosti či právě spouštějící aplikace.
- info.plist - obsahuje technické informace o vyvíjené aplikaci.

Nyní je potřeba přidat vývojové prostředí pro chytré hodinky Apple Watch:

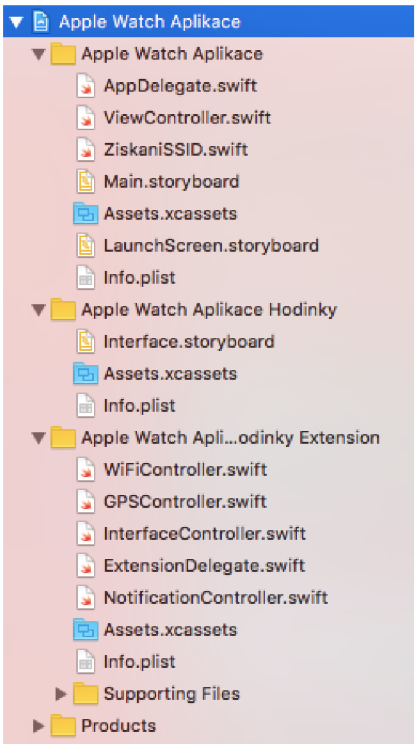
Editor -> Add target... -> watchOS -> WatchKit App.

V projektu vznikly dvě další složky - *Apple Watch Aplikace Hodinky* a *Apple Watch Aplikace Hodinky Extension.*

Ve složce `AppleWatch_Aplikace_Hodinky` je k nalezení Interface_storyboard a Assets.xcassets. Funkce jsou zcela stejné jako pro mobilní telefon.

Složka `AppleWatch_Aplikace_Hodinky_Extension` obsahuje *ViewController.swift*, *ExtensionDelegate.swift*, *NotificationController.swift*, *Assets.xcassets*, *info.plist* a složky *Supporting Files* a *Products.*

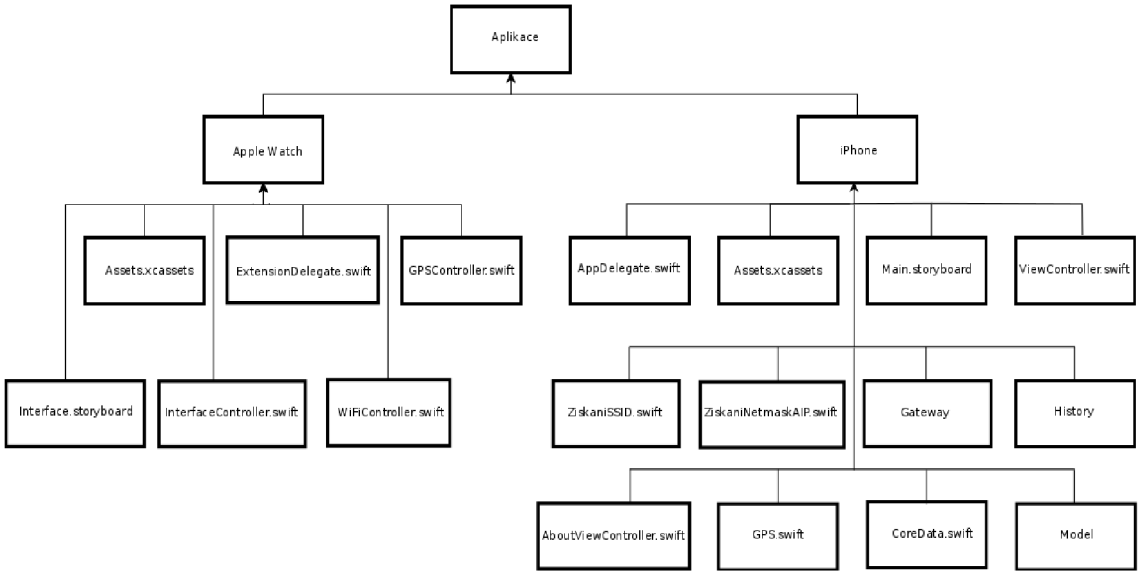
- ExtensionDelegate.swift - jedná se o obdobu třídy AppDelegate.swift.
- NotificationController.swift - slouží pro konfiguraci notifikací.
- Supporting Files - složka obsahuje dokument **PushNotificationPayload.apns** a obsahuje podporující soubory.
- Products - jak už název vypovídá, jedná se o složku zobrazující vyvíjené produkty. K nalezení je zde tedy aplikace pro iPhone a Apple Watch. Složka obsahuje ještě třetí aplikaci, která slouží k zobrazení notifikací.



Obr. 4.1: Seznam objektů v projektu.

4.2 Struktura aplikace

Pro lepší orientaci ve zdrojovém kódu je doporučené hlavní funkce aplikace rozdělit do jednotlivých tříd. Vznikne tak přehledný systém, kde je řešení problémů mnohem efektivnější a rychlejší. Struktura využitých tříd této aplikace je zobrazena níže na obr. 4.2:



Obr. 4.2: Stromová struktura aplikace.

iPhone:

- IP.swift - Konfigurace získání IP adresy.
- SSID.swift - Konfigurace získání SSID.
- Gateway.swift - Konfigurace získání brány.
- Mask.swift - Konfigurace získání masky sítě.
- GPS.swift - Konfigurace mapy, zeměpisné délky a šířky.

Apple Watch:

- WiFiDetailController.swift - Konfigurace zobrazení informací ohledně Wi-Fi.
- GPSDetailController.swift - Konfigurace zobrazení informací ohledně GPS.

4.3 Nastavení komunikace WCSession

Aby bylo možné přenášet data mezi mobilním zařízením a chytrými hodinkami, je důležité nakonfigurovat jejich vzájemnou komunikaci. To je možné uskutečnit pomocí frameworku WatchConnectivity, který umožňuje součinnost mezi iOS a aplikací

na hodinkách. Po implementování a následném nastavení bude možné z hodinek posílat žádosti na mobilní zařízení a to zpětně odesílat dat do hodinek.
Framework je potřeba nastavit do všech tříd, které budou pracovat s požadovanými daty. Přidání samotného frameworku ale nestačí. Dále je potřeba nastavit dědičnost ze třídy *WCSessionDelegate*, aby bylo možné komunikaci nastavit.

Výpis 4.1: Nastavení dědičnosti

```
@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate,
WCSessionDelegate {
}
```

Z této přidané třídy je poté potřeba vložit do **AppDelegate.swift** funkce, které jsou potřebné pro komunikaci. Přesněji se jedná o tyto funkce:

- activationDidCompleteWithactivationState,
- sessionDidBecomeInactive,
- sessionDidDeactivate,
- didReceiveMessage.

Velmi důležitá funkce je *didReceiveMessage*, neboť zde budou klíče (žádosti z chytrých hodinek) zobrazovány. Podle přijatého klíče mobilní telefon pošle požadovanou odpověď zpět na hodinky (replyHandler).

Výpis 4.2: Potřebné funkce k aktivaci komunikace.

```
public static var wcSession : WCSession!

public func session(_ session: WCSession,
activationDidCompleteWithactivationState:
 WCSessionActivationState, error: Error?) {
}
public func sessionDidBecomeInactive(_ session: WCSession)
{
}
public func sessionDidDeactivate(_ session: WCSession){
}
public func session(_ session: WCSession,didReceiveMessage
message: [String : Any], replyHandler :@escaping
([String : Any]) -> Void) {
}
```

V další části je potřebné nastavit jen samotnou komunikaci. To je možné provést zavedením podmínky do funkce, která se aktivuje při spuštění aplikace. Tato funkce se nazývá *didFinishLaunchingWithOptions* a je opět k nalezení ve třídě *AppDelegate.swift*. Celkové nastavení komunikace je potřeba provést pro každou třídu, která bude přijímat či odesílat data. Stejným způsobem je nutné nastavit komunikaci i na Apple Watch. Na straně hodinek tuto funkci lze přidat do funkce *awake*, jenž slouží k inicializaci dat.

Výpis 4.3: Nastavení komunikace.

```
if (WCSession.isSupported()) {
AppDelegate.wcSession = WCSession.default()
AppDelegate.wcSession.delegate = self
AppDelegate.wcSession.activate()
}
return true
```

4.4 Zobrazení SSID

Název Wi-Fi sítě se nazývá SSID (Service Set Identifier). Každá WLAN síť má své vlastní, které slouží k rozlišení jednotlivých sítí.

Aby bylo možné získat SSID a následně jej zobrazit na hodinkách, je v prvním kroku nutné funkci naprogramovat na mobilním zařízení. Poté bude možné tuto hodnotu poslat a zobrazit na hodinkách. Zdrojový kód bude uložen ve třídě *SSID.swift*. Při vyžádání tohoto klíče dojde k odeslání požadavku SSID (ve formě klíče) na telefonní zařízení, které po přijetí klíče pošle odpověď, která tomuto klíči náleží. Podoba funkce odeslání klíče ze strany chytrých hodinek na mobilní zařízení je uvedena níže.

Výpis 4.4: Klíč SSID.

```
InterfaceController.wcSession.sendMessage
(["PoslatSSID" : "Povedlo se"], replyHandler: {
            (response) -> Void in
     if let ssid = response["zobrazeno"] as? String {
                self.poslaneSSID.setText(ssid)
            }
        }, errorHandler: nil)
```

Lze říct, že klíč *PoslatSSID* dle konfigurace očekává, že odpověď bude též typu String (*zobrazeno*). Podle uvedené funkce *sendMessage* je vidět, že pokud klíč dorazí do mobilního zařízení, měl by jako odpověď poslat klíč s názvem *zobrazeno*

a následně změnit hodnotu pole *poslaneSSID* na požadovanou hodnotu na straně chytrých hodinek. Po realizaci kódu je tedy důležité implementovat tyto klíče.

Z důvodu zachování přehlednosti je potřebné založit novou třídu, která bude věnována pouze SSID. Třídu je možné vytvořit následujícím způsobem:

File -> New -> Swift file.

Dle stromové struktury bude tato třída pojmenována *ZjistiSSID*. Pro vývoj této funkce je zapotřebí vložit framework, který umožňuje získání informací o síti. Jeho název je *SystemConfiguration.CaptiveNetwork*. Během vývoje jsou použity dvě funkce, které se nazývají *CNCopySupportedInterfaces* a *CNCopyCurrentNetworkInfo*. Funkcionalita z obou zmíněných tříd byla převedena do nově vytvořené třídy **getSSID**, která bude při získání SSID volána.

- CNCopySupportedInterfaces - Poskytuje jména síťových rozhraní.
- CNCopyCurrentNetworkInfo - Vrací aktuální síťovou informaci pro dané síťové rozhraní.

Funkci CNCopySupportedInterfaces je možné uložit jako hodnotu určité konstanty, která bude využita k zobrazení síťových rozhraní.

Výpis 4.5: Vytvoření konstanty pro zobrazení rozhraní.

```
let rozhrani = CNCopySupportedInterfaces()
        if rozhrani == nil {
            return nil
        }
```

Další krok je věnovaný definování konkrétního rozhraní, které je požadováno k zobrazení. V případě této aplikace je potřebné SSID pouze připojené Wi-Fi.

Výpis 4.6: Výběr požadovaného rozhraní.

```
let radaRozhrani = rozhrani as! [String]
        if radaRozhrani.count <= 0 {
            return nil
        }
```

Nyní přichází na řadu funkce CNCopyCurrentNetworkInfo, která již zobrazí zvolené síťové rozhraní. Funkce je opět tvořena podmínkou.

Výpis 4.7: Zobrazení konkrétního rozhraní.

```
 let jmenoRozhrani = interfacesArray[0] as String
let nezabezpecenaDataRozhrani =
CNCopyCurrentNetworkInfo(interfaceName as CFString)

        if nezapezpecenaDataRozhrani == nil {
            return nil
        }
```

Nakonec je potřeba definovat, v jakém typu se má požadovaná hodnota vrátit (String). Aby bylo možné tuto hodnotu odesílat na chytré hodinky, je zde potřeba uvést další klíč (SSID).

Výpis 4.8: Zavedení klíče SSID.

```
let dataRozhrani = nezabezpecenaDataRozhrani as!
 Dictionary
 <String,AnyObject>

        return dataRozhrani["SSID"] as? String
```

Nyní je třeba nastavit odeslání hodnoty SSID na chytré hodinky. Jak již bylo zmíněno výše (výpis 4.4), chytré hodinky pošlou žádost pomocí funkce *sendMessage*, načež bude přijata na mobilním zařízení ve třídě *AppDelegate.swift*. Zde je implementována funkce pro přijetí zpráv *didReceiveMessage*, která podle přijatého klíče posílá příslušnou odpověď.

Výpis 4.9: Přijetí klíče s následnou odpovědí.

```
func session(_ session: WCSession, didReceiveMessage
 message: [String : Any],
 replyHandler: @escaping ([String : Any]) -> Void) {
  if let poslatSSID = message["PoslatSSID"] as? String {

            if let ssid = SSID().getSSIDInfo() {
            replyHandler(["zobrazeno" : ssid])
        }
    }
     }
```

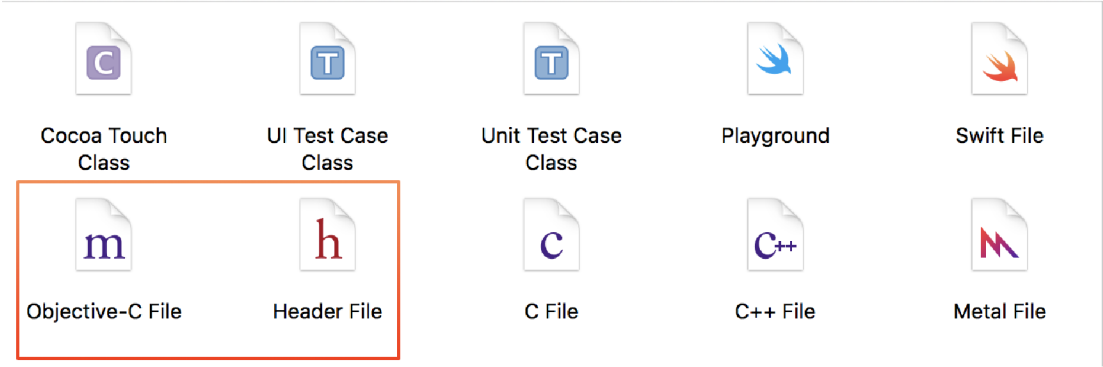
Dle uvedené odpovědi v kódu po přijetí klíče s názvem PoslatSSID, zasílá mobilní zařízení klíč *zobrazeno*, jehož přijetí je již definované ve výpise 4.4. Po získání očekáváného klíče dojde ke zobrazení hodnoty SSID.

4.5 Zobrazení IP adresy a masky sítě

Pro zobrazení konkrétnějších informací ohledně připojení k síti je možné využít framework, který byl původně vytvořen pro programovací jazyk Objective-C. Framework se nazývá *Ifaddrs* a jeho využití je možné pomocí mostu, jenž umožní propojení funkcí zvolených jazyků, v tomto případě Objective-C a Swift.

4.5.1 Vytvoření mostu

Most lze vytvořit poté, co se do projektu přidají soubory, které spadají pod jazyk Objective-C. Jedná se tedy o soubory *Objective-C File* a *Header File*.



Obr. 4.3: Soubory pro jazyk Objective-C.

Do souboru s koncovkou *.m* (*Objective-C File*) je nutné vložit framework, který bude použit i v jazyce Swift.

Výpis 4.10: Konfigurace souboru Objective-C.

```
#include <ifaddrs.h>
```

U souboru s koncovkou *.h* (*Header File*) je důležité zaznamenat frameworky, které budou využívány i jinými třídami. Nutné je také zaznamenat framework *Foundation*, který dokončí vytvoření mostu. Zatím jsou zde tedy k nalezení frameworky *Foundation* a *ifaddrs*.

Výpis 4.11: Konfigurace souboru Header.

```
#import <Foundation/Foundation.h>
#include <ifaddrs.h>
```

4.5.2 Vytvoření struktury a uložení parametrů

Nyní je již možné pracovat se zmíněným frameworkem a využít jej tak k získání požadovaných informací. S jeho pomocí je možné získat nejen IP adresu, ale i masku sítě. Z důvodu přehlednosti a uspořádání lze opět vytvořit novou třídu (jako u SSID), která bude obsahovat pouze zdrojový kód pro zmíněné parametry.

Parametry je možné vložit do vytvořené struktury, ve které se budou ukládat požadovaná data. Samotná struktura bude obsahovat dva parametry, *ip* a *netmask*, které budou sloužit k zobrazení příslušných dat.

Výpis 4.12: Struktura pro získání dat.

```
struct VypisInfo {
let ip : String
let netmask : String
}
```

Získání těchto údajů je znázorněno níže ve výpisu 4.13. Tato implementace zobrazí seznam všech rozhraní na lokálním zařízení.

Výpis 4.13: Získání parametrů.

```
var ifaddr: UnsageMtablePointer<ifaddrs>? = nil
if getifaddrs(&ifaddr) == 0 {

var ptr = ifaddr;
while ptr != nil {
let flasgs = Int32((ptr?.pointee.ifa_flags)!)
var addr = ptr?.pointee.ifa_addr.pointee
//Kontrola, zda se jedná o IPv4 či IPv6.
if (flags & (IFF_UP|IFF_RUNNING|IFF_LOOPBACK)) ==
 (IFF_UP|IFF_RUNNING) {
if addr?.sa_family == UInt8(AF_INET) ||
addr?.sa_family == UInt8(AF_INET6) {
//Převod do lidské řeči.
}
```

Jako nejdůležitější část kódu lze považovat převod adres do typu operátoru *String*, aby jej mohl člověk lehce přečíst. Kód dále zahrnuje výpis výsledků do vytvořené struktury na příslušné pozice (*ip* a *netmask*).

Výpis 4.14: Převod do operátoru String.

```
var hostname = [CChar](repeating: 0, count: Int
(NI_MAXHOST))
if (getnameinfo(&addr!, socklen_t((addr?.sa_len)!),
&hostname, socklen_t(hostname.count),
nil, socklen_t(0), NI_NUMERICHOST) == 0) {
if let address = String.init(validatingUTF8:hostname) {
var net = ptr?.pointee.ifa_netmask.pointee
var netmaskName = [CChar](repeating: 0, count:
Int(NI_MAXHOST))
getnameinfo(&net!, socklen_t((net?.salen)!),
 &netmaskName, socklen_t(netmaskName.count),
nil, socklen_t(0), NI_NUMERICHOST)
if let netmask = String.init(validatingUTF8:netmaskName)
{
//Výpis na příslušné pozice do struktury.
adresses.append(VypisInfo(ip:address, netmask:netmask))
      }
  }
}
```

Důležité je také zmínit, jak lze tyto parametry zobrazit. Při volání těchto funkcí se na polích IP adresy a masky podsítě zobrazí všechny adresy lokálního zařízení a je tedy nutné definovat zobrazení posledních parametrů, které představují požadované hodnoty. Zobrazení těchto hodnot je adresované na konstanty na straně mobilního zařízení. Tento princip je použit především u mobilní aplikace. Na stejném či podobném způsobu budou zobrazovány i další parametry.

Výpis 4.15: Adresování ip adresy a masky sítě.

```
let ip = getAddresses().last!.ip
let netmask = getAddresses().last!.netmask
```

Po implementaci celkového kódu je nutné opět vytvořit dvojici klíčů, jenž bude sloužit k zobrazení dat na chytré hodinky. Podmínka ohledně přijetí klíče a její následné odpovědi se definuje ve stejné funkci a třídě (*AppDelegate.swift*), jako již nadefinované SSID. Pro IP adresu i masku sítě je potřebné vytvořit odlišnou dvojici klíčů. Na straně chytrých hodinek je nutné nastavit, co se má vykonat po přijetí klíče.

Výpis 4.16: Nastavení odpovědi ze strany mobilního telefonu.

```
if (message["PoslatNetmask"] as? String != nil {
let gateway = IPaMaska.getAddresses().last!.netmask
replyHandler(["zobrazenoNetmask" : gateway])
}
```

4.6 Zobrazení Gateway

Výchozí brána (Gateway) je označení pro tzv. směrovač (router). Získání jeho IP adresy v této aplikaci je možné pomocí čisté implementace programovacího jazyka *Objective-C*. Z důvodu funkčního použití je nutné implementovat soubory a frameworky do mostu, aby je šlo použít v jazyce *Swift*.

Výpis 4.17: Přidané soubory a frameworky pro konfiguraci Gateway.

```
#include <stdio.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <sys/sysct1.h>
#include "getgateway.h"
#include "route.h"
#include <net/if.h>
#include <string.h>
```

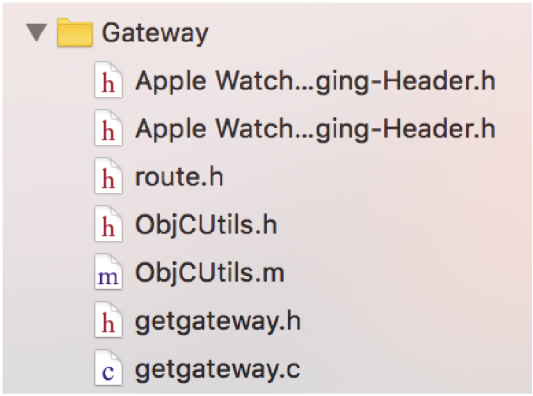
Pro získání této adresy bude také opět použit framework *<ifaddrs.h>*. Důležitou roli zde hraje soubor *route.h*. Zdrojový kód uvnitř souboru využívá většinu potřebných frameworků uvedených výše (výpis 4.17) a používá se především pro simulátor.[1]

Celková konfigurace získání výchozí brány je definována pomocí 5 souborů, které na sebe navzájem navazují a jejich výsledkem je tedy zobrazení požadovaného parametru.

- getgateway.c - použit programovací jazyk C k získání adresy a přístupu na router.
- getgateway.h - oznamuje ostatním souborům a třídám, že taková metoda existuje a udává potřebné parametry.
- ObjUtils.m - definované jednotlivé metody.

[1]Použití tohoto kódu se používalo dříve k získání informací routeru. Později však nebyla tato cesta déle povolena a aplikace s tímto kódem tedy nebylo možné nahrát do obchodu *AppStore*. Nedávno byl ale tento krok odebrán a nyní je opět možné tuto cestu použít. V této aplikaci se nachází kód pod veřejnou licencí firmy Apple, verze 2.0.

- ObjUtils.h - deklarace metod, které budou používány. Následně přidány do *bridge*, aby byl přístup i pro jazyk Swift.
- route.h - popis výše, uvedeno především pro simulátor.



Obr. 4.4: Soubory pro zobrazení adresy výchozí brány.

Samotné zobrazení této adresy je pak nutné nakonfigurovat v souboru *ViewController.swift* na straně mobilního zařízení. Konfigurace je znázorněna níže ve výpisu 4.18, přičemž *mojeGateway* je uživatelem vytvořený *label* ve vývojovém prostředí. Zde se nakonec vypíše požadovaná adresa, pokud bude uživatel připojen k Wi-Fi.

Výpis 4.18: Konfigurace Gateway na straně mobilního zařízení.

```
func zobrazGateway() {
self.mojeGateway.text = ObjCUtils.getGatewayIP()
}
```

Na straně chytrých hodinek se opět data zasílají pomocí klíče. V prvním kroku hodinky zasílají zprávu a mobilní zařízení na ni reaguje. Label *poslaneGateway* zastává stejnou funkci jako label *mojeGateway*, ale na straně chytrých hodinek.

Výpis 4.19: Gateway zpráva ze strany chytrých hodinek.

```
InterfaceController.wcSession.sendMessage
(["PoslatGateway" : "Poslalo se"],
 replyHandler: { (response) -> Void in
if let gateway = response["zobrazenoGateway"]
 as? String { self.poslaneGateway.setText(gateway)
}
}, errorHandler: nil)
}
```

Výpis 4.20: Odpověď na zprávu Gateway ze strany mobilního zařízení.

```
if (message["PoslatGateway"] as? String != nil {
let gateway = ObjCUtils.getGatewayIP()
replyHandler(["zobrazenoGateway" : gateway!]
}
}
```

4.7 Kontrola připojení k Wi-Fi

Protože funkce vytvořené výše jsou závislé na připojení k bezdrátové síti Wi-Fi, je potřeba vytvořit další funkci, která bude toto připojení kontrolovat. Pro lepší přehlednost vznikne opět nově vytvořený soubor se jménem *KontrolaWiFi*, který bude obsahovat otevřenou třídu pojmenovanou *Reachablity*. Otevřená třída znamená, že je přístupná i podtřídám vně definujícího modulu. Vzhledem k tomu, že se jedná o zjištění připojení k Wi-Fi, bude zde opět využit framework *SystemConfiguration*.

Výpis 4.21: třída *Reachability*.

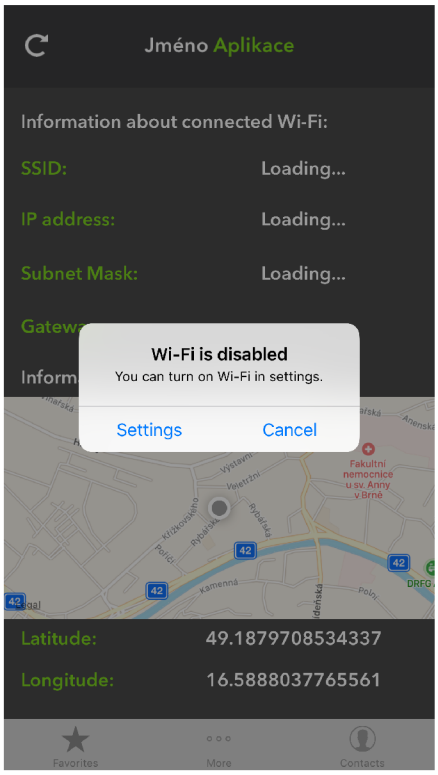
```
class func isConnectedToNetwork() -> Bool {
var zeroAddress = sockaddr_in()
zeroAddress.sin_len =
UInt8(MemoryLayout.size(ofValue: zeroAddress))
zeroAddress.sin_family = sa_family_t(AF_INET)
let defaultTouteReachability = withUnsafePointer
(to: &zeroAddress) {
 \$0.withMemoryRebound(to: sockaddr.self, capaticy: 1)
{zeroSockAddress in
SCNEtworkReachabilityCreateWithAddress (nil, zeroAddress)
}}
var flags = SCNetworkReachabilityFlags()
if !SCNetworkReachabilityGetFlags
(defaultRouteReachability!, &flags) {
return false
}
let isReachable = (flags.rawValue & UInt32
(kSCNetworkFlagsReachable)) != 0
let needsConnection = (flags.rawValue & UInt32
(kSCNetworkFlagsConnectionRequired)) ! =0
return (isReachable &&  !needsConnection) }}
```

Vytvořená funkce také zjišťuje, zda uživatel využívá mobilní data. Při jejich použití dojde k zobrazení IP adresy zařízení a masky sítě. Ostatní parametry nejsou k dispozici. Pro využití této funkce je vytvořený alert (upozornění), kdy není detekované připojení k Internetu. Funkce je dále implementována u zobrazení parametru IP adresy. Pokud je tedy uživatel připojen, dojde k zobrazení. Pokud ne, dojde ke zobrazení alertu. Ten bude uživatele informovat, že není připojen k síti a bude mít možnost přejít do nastavení aplikace. Zobrazení alertu je uvedeno na obr. 4.5.

Výpis 4.22: Zdrojový kód alertu.

```
func alertWifi() {
if (Reachability.isConnectedToNetwork() == true) {
let ip = IPaMaska.getAddresses().last?.ip
guard let ipAdresa = ip else {
print("IP adresa není k dispozici")
return
}
print("Moje IP adresa je: \(ipAdresa)")
self.mojeIP.text = ipAdresa} else {
 print("Připojení k internetu se nezdařilo")
let alertController = UIAlertController
(title: "Wi-Fi is disabled", message:
"You can turn on Wi-Fi in settings",
prefferedStyle: .alert)
let settingsAction = UIAlertAction
(title: "Settings", style: .default)
{
(_) -> Void in
guard let settingsUrl = URL(string:
UIApplicationOpenSettingsURLString) else {
return
}
if UIApplicatoin.shared.canOpenURL
(settingsUrl) {
UIApplication.shared.open(settingsUrl,
completionHandler: { (success) in
print("Settings opened: \(success)")
          }
        )
      }
```

```
}
alertController.addAction(settingsAction)
let cancelAction = UIAlertAction(title:
"Cancel", style: .default, handler: nil)
alertController.addAction(cancelAction)
present(alertController, animated: true,
completion: nil)
    }
}
```



Obr. 4.5: Zobrazení alertu.

4.8 Polohové služby

Další funkcionalitou aplikace je zobrazení informací ohledně polohy uživatele. Informace zahrnují zeměpisnou délku a šířku, vertikální a horizontální přesnost a zobrazení polohy uživatele na mapě. Jelikož chytré hodinky Apple Watch Series 1 neobsahují GPS modul, získávají zeměpisnou délku a šířku uživatele pomocí dat z mobilního zařízení. Získání těchto dat však již neprobíhá formou přeposílání klíče, ale

konfigurací a využitím frameworku *Core Location*[2]. Na straně mobilního zařízení se jedná o použití frameworku *MapKit*. Zařízení mají tedy pro polohové služby odlišnou konfiguraci.

4.8.1 Konfigurace na straně iPhone

Jak již bylo zmíněno, konfigurace je realizována prostřednictvím frameworku *MapKit*[3]. Díky němu je tedy možné získat potřebné informace a využít náhled mapy. Aby ale mohlo dojít k zobrauení požadovaných údajů, je nutní povolit polohové služby. Bez tohoto povolení tedy není možné získat informace o poloze a zobrazit polohu uživatele.

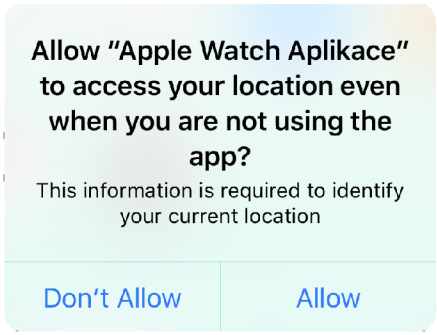
Pro realizaci tohoto povolení je nutné přidat v sekci *Information Property List*, jenž se nachází v souboru *info.plist*. Zde je potřeba přidat *Privacy - Location Always Usage Description* a k němu přiřadit hodnotu. Hodnotou je zde na mysli text, který uživateli oznámí, proč je například potřeba povolit polohové služby. Dále je možné definovat, pro které zařízení je možné aplikaci stáhnout. Potřebná funkce telefonu bude jistě GPS čip a polohové služby. Tyto a jiné požadavky lze definovat v záznamu *Required device capabilities*. Pokud se nyní přidá do *main.storyboard* položka *MapKit*

Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
Localization native development region	String	en
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application Category	Boolean	NO
Application requires iPhone environment	Boolean	YES
Privacy - Location Always Usage Description	String	This information is required to identify your current location
Launch screen interface file base name	String	LaunchScreen
Main storyboard file base name	String	Main
▼ Required device capabilities	Array	(3 items)
Item 0	String	location-services
Item 1	String	gps
Item 2	String	armv7
Status bar is initially hidden	Boolean	YES
▶ Supported interface orientations	Array	(3 items)
View controller-based status bar appearance	Boolean	NO

Obr. 4.6: Záznam v info.plist

[2]Reference na fr. *Core Location* -`https://developer.apple.com/reference/corelocation`.
[3]Reference na framework *MapKit* - `https://developer.apple.com/reference/mapkit`.

View(a také framework *MapKit* do třídy *ViewController*), mělo by dojít k zobrazení žádosti o povolení polohových služeb, která je zobrazena na obrázku 4.7. Nyní



Obr. 4.7: Zobrazení žádosti o povolení polohových služeb.

je možné pracovat na získání požadovaných informací. Celá konfigurace proběhne ve třídě s názvem *ViewController*. Pro přidání zmíněného frameworku je potřeba také přidat podtřídu *CLLocationManagerDelegate*, se kterou se bude pracovat. Pro úspěšný chod této potřídy je nutné také definovat její metody. Pro jednodušší přehlednost je doporučené si předem vytvořit konstantu, jenž bude obsahovat funkci *CLLocationManager()*. Vznikne tak přehlednější a uspořádanější kód.

Metody:

- **didChangeAuthorization status** - definuje, co se stane či nestane při povolení/nepovolení polohových služeb. Pokud je povolení uděleno, dojde k zobrazení potřebných údajů. Pokud ne, zobrazí se notifikace. Ta informuje o nemožnosti provedení funkce. V této metodě se také nastavuje vykreslení polohy uživatele na přidanou mapu (*zobrazenimapy.showUserLocation = true*).
- **didUpdateLocations** - Definice postupu po aktualizaci polohy uživatele. Definice postupu po aktualizaci polohy uživatele.
- **didFailWithError** - pokyny pro aplikaci, pokud nastane chyba.

Další nutností je definovat kroky, které se vykonají při načtení stránky. V případě využití polohových služeb v této aplikaci se jedná o nastavení delegátu, zobrazení žádosti povolení a aktualizování polohy uživatele. Pole *zobrazeniMapy* slouží k zobrazení polohy na mapě v *main.storyboard*. Po spuštění následně dojde k animaci na mapě, jenž zobrazí polohu uživatele a neustále ji aktualizuje.

Výpis 4.23: Nastavení delegátu.

```
locationManager.delegate = self
locationManager.requestWhenInUseAuthorization()
location.startinUpdatingLocation()
zobrazeniMapy.setUserTrackingMode MKUserTrackingMode
```

```
follow, animated:true)
```

Nyní by tedy nemělo nic bránit tomu, aby došlo k zobrazení aktuální polohy uživatele. Díky výše definované konstantě *locationManager* je možné získat také zbývající parametry. Zdrojové kódy pro tyto informace jsou znázorněny níže ve výpisu 4.24.

Výpis 4.24: Výpis parametrů.

```
let zemDelka =
locationManager.location?.coordinate.longitude
let zemSirka =
locationManager.location?.coordinate.latitude
let verPresnost =
locationManager.location?.verticalAccuracy
let horPresnost =
locationManager.location?.horizontalAccuracy
```

4.8.2 Konfigurace na straně Apple Watch

Konfigurace na straně chytrých hodinek je velmi podobná. Velký rozdíl však spočívá v tom, že není zapotřebí přidávat žádný záznam do *info.plist* na straně hodinek. Ten byl již proveden na straně mobilního zařízení. Stejné je použití podtřídy *CLLocationManagerDelegate*, která opět potřebuje být definována pomocí metod, které jsou zde dvě (*didUpdateLocations locations*, *didFailWithError*).

Opět je potřeba také definovat, co se vykoná při načtení stránky. Podobně jako na straně telefonu je potřeba nastavit delegát a žádost o lokaci.

Výpis 4.25: Nastavení delegátu a žádosti lokace.

```
locationManager.requestAlwaysAuthorization()
locationManager.delegate = self
locationManager.requestLocation()
```

Zbytek konfigurace je implementován v metodě *didUpdateLocations location*. Souřadnice jsou zde zjišťovány pomocí modulu *CLLocationCoordinate2DMake*. Region je nastavený pomocí rozsahu a souřadnic. Po zobrazení mapy dojde k zobrazení polohy uživatele na mapě. Zobrazení zeměpisné délky a šířky lze nastavit stejným způsobem, jako na straně telefonu. Celková konfigurace je zobrazena ve výpisu 4.26.

Výpis 4.26: Odpověď na zprávu Gateway ze strany mobilního zařízení.

```
let location = locations.first
let coordinate = CLLocationCoordinate2Make(
location!.coordinate.latitude,
location!.coordinate.longitude)
let span = MKCoordinateSpanMake(0,005, 0,005)
let region = MKCoordinateRegionMake(coordinate, span)
zobrazenaPoloha.addAnnotation(coordinate, with: .red)
zobrzenaPoloha.setRegion(region)
```

4.9 Zobrazení parametrů v aplikaci

Po implementaci všech funkcí, které jsou potřeba pro chod aplikace, je nyní možné přejít k jejich zobrazení na mobilním zobrazení a chytrých hodinkách. Nastavení zobrazení lze provést dvěma způsoby - kódem či nastavováním polí v *main.storyboard*. Pro tuto aplikaci bude zvolen druhý způsob.

Každá obrazovka bude znázorněna pomocí kontrolérů, v nichž je možné vložit pole, mapy a další objekty. Objekty je možné přidat přeháthnutím do kontroléru z nabídky *Objects*.

Aby bylo možné přiřadit objektům nějakou funkci, je nutné jim přiřadit určitou třídu. Do třídy bude potřebné navázat spojení s objekty a následně je konfigurovat tak, aby provedly požadovanou práci.

Výpis 4.27: Propojený objekt a přiřazená funkce.

```
@IBOutlet wear var zobrazeneSSIDLabel: UILabel!
//připojený objekt do kontroléru
self.zobrazeneSSIDLabel.text = ssid.getSSIDInfo()
//přiřazená funkce objektu
```

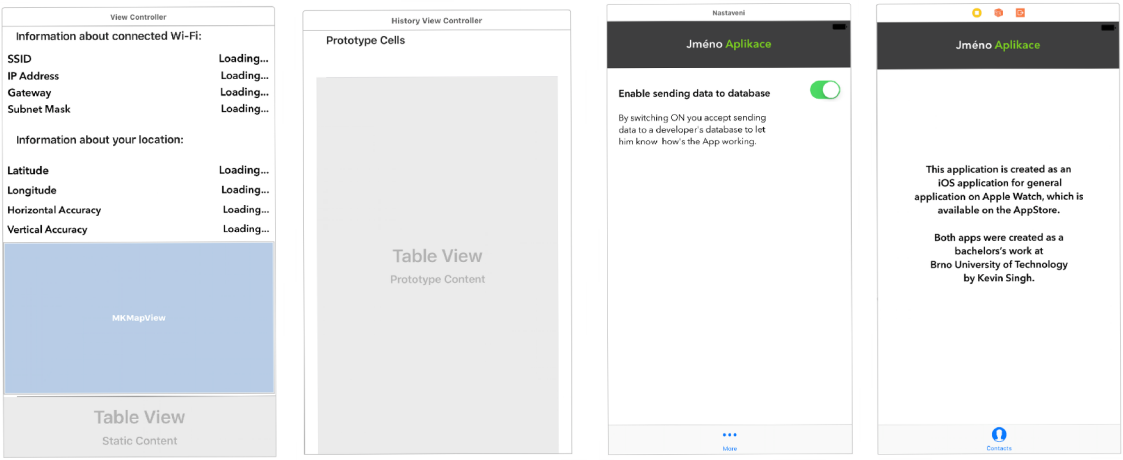
4.9.1 Konfigurace mobilní aplikace

Mobilní aplikace bude celkově obsahovat čtyři obrazovky viz obr. 4.8. První obrazovka bude sloužit k vypsání požadovaných parametrů a zobrazení uživatele na mapě. Dále také poskytuje aktualizaci těchto údajů. Druhá obrazovka prezentuje databázi, jenž zaznamená naměřené parametry a zobrazí je v tabulce. Konfigurace databáze proběhne po nastavení zobrazení parametrů. Třetí obrazovka znázorňuje nastavení aplikace, které umožňuje zasílat data na univerzitní server. Čtvrtá a zároveň poslední obrazovka obsahuje informace ohledně aplikace.

Přiřazené třídy kontrolérům:

- ViewController - ViewController.swift.
- HistoryTableView - HistoryTableViewController.swift.
- Nastaveni - Nastaveni.swift.

Informační kontrolér má přidělenou třídu *AboutViewController.swift*. Kontrolér obsahuje pouze jedno pole *UILabel* a tři objekty typu *Image View*. Pole podává textovou informaci ohledně vzniku aplikace a také o existenci druhé aplikace pro chytré hodinky. Objekty *Image View* slouží k zobrazení log Fakulty elektrotechniky a telekomunikačních technologií, VUT a výzkumného centra Wislab.



Obr. 4.8: Obrazovky.

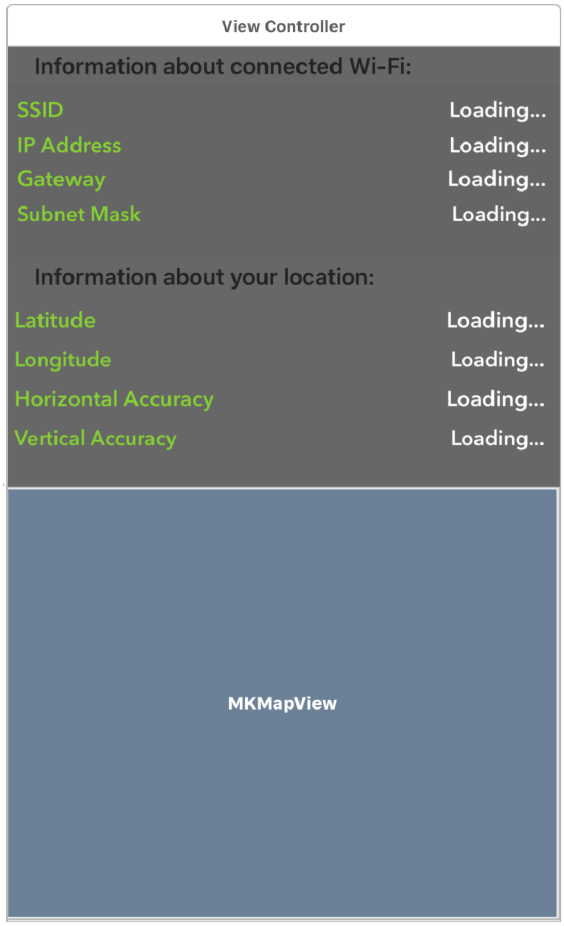
Propojení obrazovek

Aby bylo možné zobrazit i ostatní obrazovky než jen spouštěcí, je nutné obrazovky mezi sebou propojit (obr 4.9). Toho lze opět dosáhnout v *main.storyboard* dalším kontrolérem, jenž je možný k nalezení v objektech pod názvem *Tab Bar Controller*. Ten umožňuje jednoduchou navigaci mezi jednotlivými obrazovkami v podobě menu, které se v aplikaci zobrazí ve spodní liště. Kontrolér je potřeba nastavit jako kontrolér spouštěcí. Toto menu bude poté zobrazeno na každé obrazovce, kterou bude možné identifikovat různým textem či obrazcem, jenž bude v této liště zobrazen. Propojení obrazovek je možné pomocí držení klávesy *control* a najetím na další kontrolér, přičemž začínající kontrolér je vždy *Tab Bar Controller*.

Konfigurace hlavní obrazovky

Hlavní obrazovka slouží k zobrazení zjištěných informací a polohy uživatele na mapě. Celkově obrazovka poskytuje dvě sekce, přičemž první je věnována informa-

cím ohledně připojení k internetu a druhá GPS informacím. První sekce obsahuje celkem osm objektů, které jsou všechny typu *UILabel*. Čtyři z uvedených objektů mají neměnící se hodnotu, neboť uživatele pouze informují, co zobrazené hodnoty znamenají (SSID, IP adresa, výchozí brána, maska sítě). Ke každému zmíněmému objektu je dále přiřazen další objekt, jenž již zobrazuje požadovanou hodnotu. Druhá sekce obsahuje stejný počet objektů stejného typu. Čtyři objekty opět slouží k informačnímu sdělení(zeměpisná šířka, zeměpisná délka, horizontální přesnost, vertikální přesnost) a zbývající objekty opět zobrazují jejich hodnoty. Dále je zde k nalezení objekt typu *MapKit View*, jenž slouží pro zobrazení aktuální polohy.

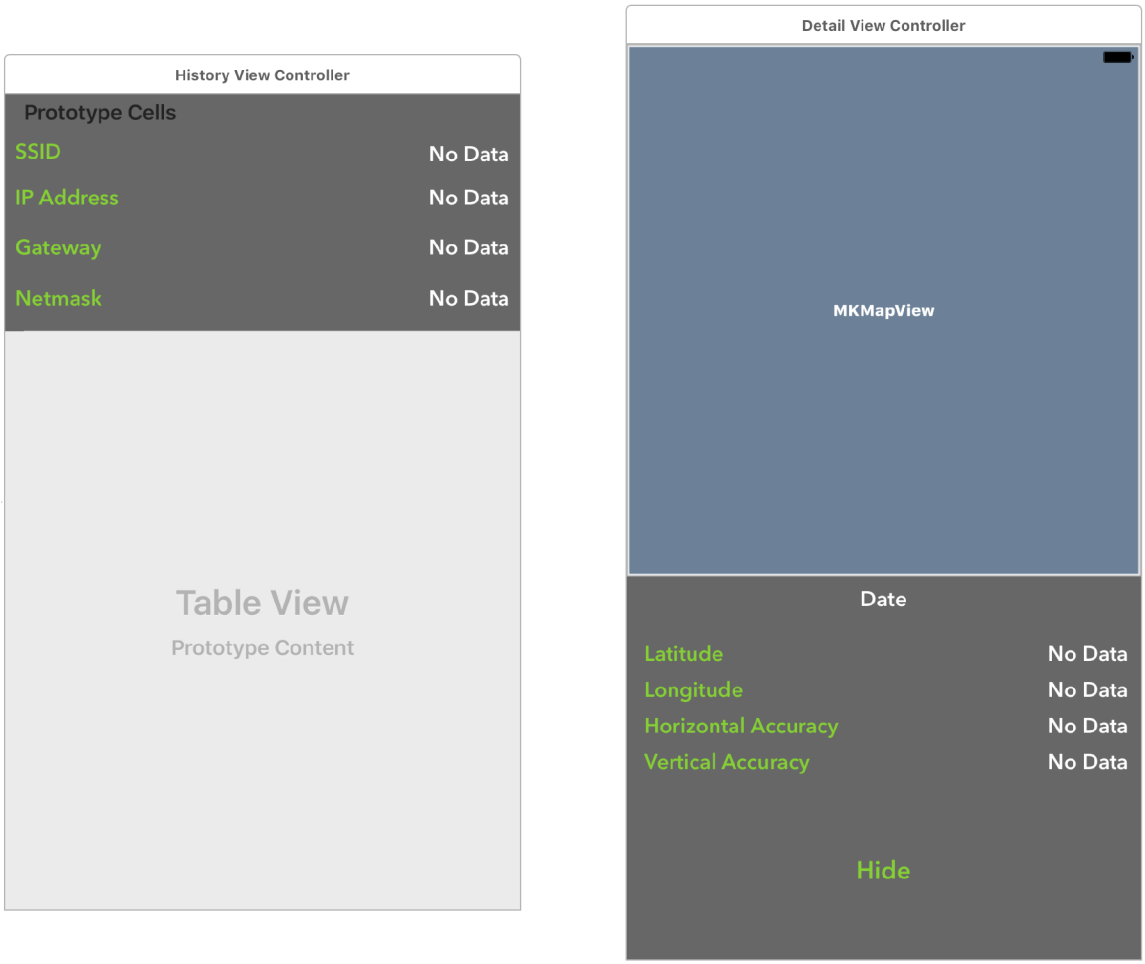


Obr. 4.9: Obrazovka ViewController.

Konfigurace obrazovky Historie

V této obrazovce je k nalezení tabulka (*UITableView*), která bude zobrazovat uložené údaje z databáze. Jednotlivý záznam bude zobrazovat informace o připojení

k Iternetu ve formě buňky, přičemž po jejím rozkliknutí dojde k zobrazení další obrazovky, jenž zobrazí informace o poloze uživatele. Nechybí zde ani zobrazení polohy na mapě. Možností také bude celou buňku smazat, čímž dojde ke smazání záznamu v tabulce. Nastavení této obrazovky je více znázorněno v sekci 4.9.3, kde je také definován způsob ukládání záznamů.

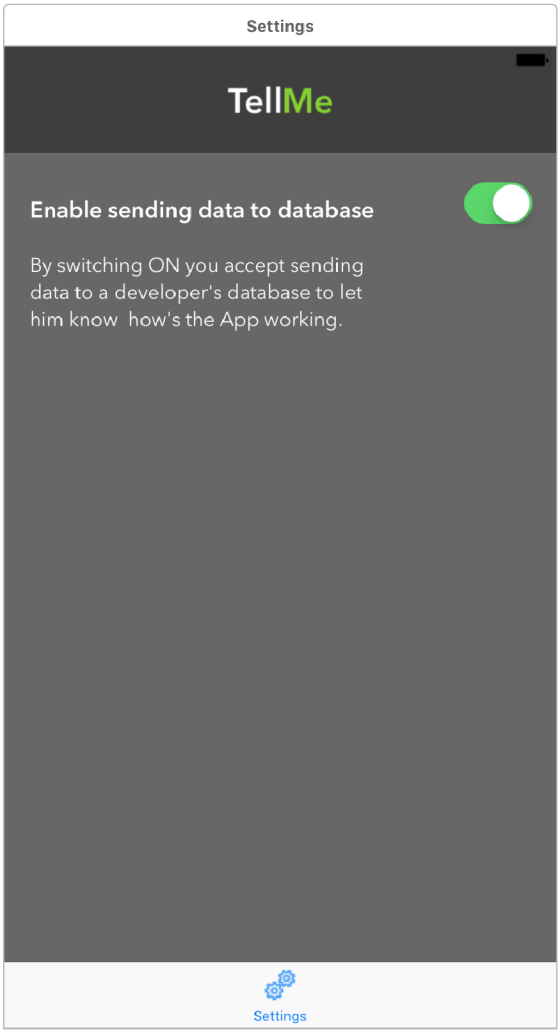


Obr. 4.10: Obrazovky znázorňující uložené záznamy a detail jejich zobrazení.

Konfigurace obrazovky Nastavení

Tato obrazovka definuje nastavení mobilní aplikace. Celkem zde ale bude jen jedna možnost povolení, jenž bude prezentovat ukládání dat na univerzitní server. Možnost povolení bude ve formě objektu typu *UISwitch*. Pokud bude v poloze vypnuto, nedojde k odesílání dat na server. Dále se zde nachází dvojice objektů typu *UILabel*.

Tyto objekty obsahují text, jenž zdůvodňuje potřebu povolení odesílání záznámů.[4]



Obr. 4.11: Obrazovka nastavení.

4.9.2 Zobrazení parametrů v aplikaci na chytrých hodinkách

Zobrazení informací na chytrých hodinkách probíhá v podobném stylu, jako na mobilním zařízení. Opět se jedná o využití obrazovek a objektů, které je potřeba vhodně nakonfigurovat.

Z důvodu malého displeje a zároveň zachování přehledného zobrazení informací došlo k samostatnému zobrazení jednotlivých sekcí. Všechny informace tedy nejsou

[4]Pro případné povolení/vypnutí polohových služeb (k získání údajů o poloze uživatele) je potřeba přejít do hlavního nastavení mobilního zařízení (*Nastavení -> Soukromí -> Polohové služby*).

zobrazeny na jedné obrazovce, jako tomu je na mobilním zařízení. Celkový počet vytvořených obrazovek na hodinkách je tři, přičemž k obrazovkám jsou opět přiděleny definiční třídy.

Přiřazené třídy kontrolérům:

- Hlavní obrazovka - ViewController.swift.
- Wi-Fi informace - WiFiController.swift.
- GPS informace - GPSController.swift.

Obrazovky je opět nutné mezi sebou propojit. Zde již však není zapotřebí využití navigačního panelu, ale stačí propojit hlavní obrazovku se zbývajícími. Tlačítko *Wi-Fi* tedy bude propojené s obrazovkou zobrazující informace připojení k Internetu a tlačítko GPS s obrazovkou, která poskytuje informace o poloze uživatele (propojení je realizováno pomocí funkce *push*).

Obr. 4.12: Propojení obrazovek chytrých hodinek.

Konfigurace hlavní obrazovky

Hlavní obrazovka slouží jako menu aplikace. Jsou zde implementovány dvě tlačítka, přičemž první při stisknutí přesměruje uživatele do sekce s informacemi o síti. Druhé

pak do sekce s GPS informacemi.

Obr. 4.13: Hlavní obrazovka Apple Watch.

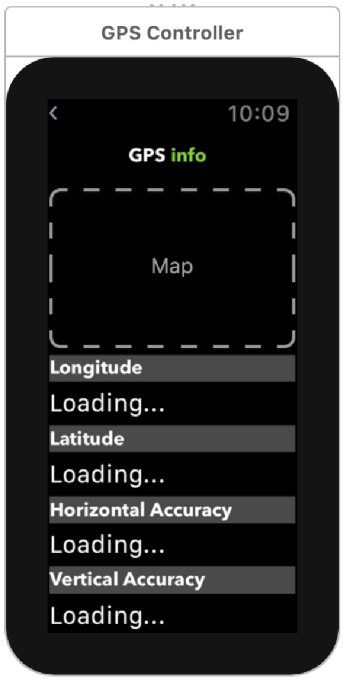
Konfigurace obrazovky Wi-Fi informace

Po přesměrování z hlavní obrazovky dojde k zobrazení osmi objektů typu *UILabel*. Stejně jako u mobilní aplikace slouží čtyři k názvu parametru (SSID, IP adresa, Výchozí brána, Maska sítě) a zbývající k vypsání jejich hodnoty. Dále se zde nachází šipka, jenž slouží k návratu do úvodního menu.

Obr. 4.14: Obrazovka Apple Watch pro připojení k internetu.

Konfigurace obrazovky GPS info

Zde se nachází stejný počet objektů typu *UILabel*(8), které prezentují názvy parametrů (zeměpisná šířka, zeměpisná délka, horizontální přesnost, vertikální přesnost) a jejich hodnoty. Dále je zde objekt typu *MapKit View*, který slouží pro zobrazení aktuální polohy uživatele. Šipka pro návrat do menu je zde implementována také.



Obr. 4.15: Obrazovka Apple Watch pro polohu uživatele.

4.9.3 Mobilní a webová databáze s využitím *Core Data*

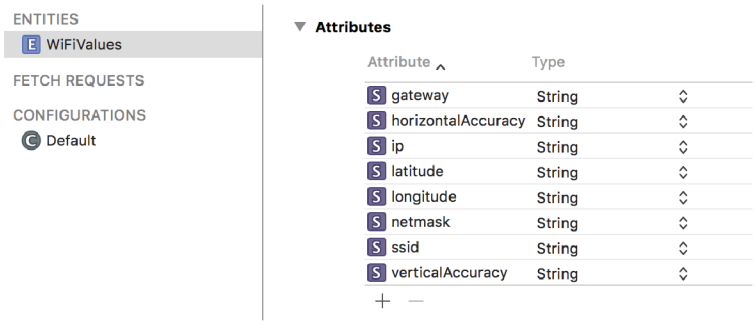
Aby bylo možné získané data nějakou formou uložit do mobilního zařízení a následně zobrazit, je potřeba využití frameworku *Core Data* [5], jenž působí jako lokální databáze. Po konfiguraci se nastaví objekty, jenž mají být v paměti telefonu uloženy a následně k dispozici při dalším spuštění aplikace. Pro jeho správnou funkčnost je potřeba přidat jeho metody do třídy *AppDelegate* a vytvořit model databáze. V něm je možné vytvořit potřebné entity. Entita prezentuje určitý celek atributů, kde jednotlivý atribut znázorňuje určitý záznam. Každému atributu je nutné přiřadit, jakého typu hodnot bude nabývat.

V případě této aplikace je dostačující vytvořit pouze jednu entitu, jenž bude obsahovat osm atributů typu *String* (atributy budou nabývat textovou hodnotu).

[5]Reference na framework *Core Data* - `https://developer.apple.com/reference/coredata`.

Tyto atributy budou ukládat informace ohledně připojení k Internetu a informace o poloze uživatele. Budou tedy zobrazovat uložené SSID, IP adresu, výchozí bránu, masku sítě, zeměpisnou délku a šířku, vertikální a horizontální přesnost. Atributy budou dále přiřazeny určitým objektům v aplikaci, aby zde docházelo k zobrazení uložených hodnot.

Po vytvoření a nakonfigurování modelu je potřebné vytvořit podtřídy, které umožní uložit data na dané pozice. Vytvoření podtříd je možné touto cestou: *Editor -> Create Subclassess ManagedObjects*.



Obr. 4.16: Vytvořený model databáze.

K nastavení a zobrazení hodnot slouží kontrolér s názvem *HistoryViewController*. Pro tento kontrolér bude vytvořena tabulka, jenž bude zobrazovat uložené údaje z databáze. Jedná se o dynamickou tabulku, jenž má jeden typ buňky. Tento typ buňky bude prezentovat údaje o připojení k Internetu a po stisknutí této buňky dojde k zobrazení detailu, jenž bude obsahovat informace o poloze uživatele (viz sekce Konfigurace obrazovky historie).

Uložení dat proběhne při prvotním spuštění aplikace a také při aktualizaci hlavní obrazovky, jenž je definována v kontroléru *ViewController*.

HistoryViewController bude využívat framework, který se nazývá *CoreData*. Jak již z názvu vyplývá, jedná se o framework umožňující ukládání dat. Bez této funkce by se při opětovném spuštění aplikace data vždy vymazala.

Pro získání požadovaných hodnot je potřeba vytvořit konstanty, které poté budou přiřazeny jednotlivým atributům v entitě. Tyto konstanty jsou vytvořeny v kontroléru *ViewController* a jsou definovány ve funkci *alertWifi()*. Zde je vytvořena podmínka. Pokud je uživatel připojen k Internetu, dojde k uložení dat do mobilní databáze (a také k zobrazení dat na hlavní obrazovce). V opačném případě se žádná data neukládají, neboť žádná nejsou k dispozici (na hlavní obrazovce jsou v tomto případě objekty znázorněny textem "*No Connection*").

Výpis 4.28: Konstanty pro zobrazení požadovaných dat.

```
let zobrazenoSSID = SSID().getSSIDInfo()
let zobrazenoIP = IPaMaska.getAddresses().last?.ip
let zobrazenoBrany = ObjCUtils.getGatewayIP()
let zobrazenoMasky = IPaMaska.getAddresses().last?
.netmask
let zemSirka = Float((locationManager.location?
.coordinate.latitude)!)
let zemDelka = Float((locationManager.location?
.coordinate.longitude)!)
let horPresnost = Float((locationManager.location?
.horizontalAccuracy)!)
let verPresnost = Float((locationManager.location?
.verticalAccuracy)!)
//Nyní uložení do entity \textit{WiFiValues} v modelu.
let task = WiFiValues(context: context)
task.ssid = zobrazenoSSID
task.gateway = zobrazenoBrany
task.netmask = zobrazenoMasky
task.latitude = zemSirka
task.longitude = zemDelka
task.horizontalAccuracy = horPresnost
task.verticalAccuracy = verPresnost
//Příkaz k uložení dat na příslušné atributy.
(UIApplication.shared.delegate as! AppDelegate)
.saveContext()
```

Nutné je, aby byly názvy atributů přesné. Pokud by tak nebylo, nedošlo by k uložení dat a aplikace by se neočekávaně ukončila.

4.9.4 Zobrazení uložených dat

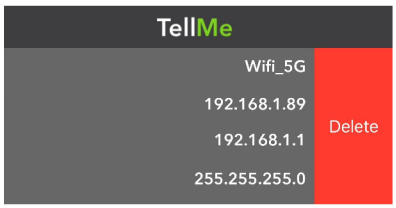
Ve výpisu 4.28 je znázorněno, jak data uložit do atributů v entitě *WiFiValues*. Nyní je však ještě potřebné data z databáze vyjmout a následně zobrazit. Získání i zobrazení bude definováno v metodě buňky tabulky, jenž je označena jako *cellForRowAt*. Celá tato konfigurace probíhá v kontroléru *HistoryViewController*.

Výpis 4.29: Zobrazení uložených dat

```
if let cell = tableView.dequeueReusableCell
(withIdentifier: "historyFirstTypeCell") as?
```

```
HistoryFirstTypeCellTableViewCell {
let task = tasks[indexPath.row]
let context = (UIApplication.shared.delegate
as! Appdelegate).persistentContrainer.viewContext
(UIApplication.shared.delegate as!
AppDelegate).saveContext()
do {
tasks = try context.fetch(WiFiValues.fetchRequest())
} catch {
print("Fetching Failed")
}
cell.ssidValueLabel.text = task.ssid
cell.ipValueLabel.text = task.ip
cell.gatewayValueLabel.text = task.gateway
cell.netmaskValueLabel.text = task.netmask
return cell
}
let cell = UITableViewCell()
return cell
```

Po úspěšném nastavení se nyní budou změřené hodnoty ukládat do lokální databáze v mobilním zařízení a následně zobrazovat v záložce *History*. Jak již bylo zmíněno, data se budou ukládat při spuštění aplikace či při aktualizaci údajů. Data bude také možné smazat, což je možné posunutím záznamu směrem vlevo a následně zvolit možnost *delete*.



Obr. 4.17: Smazání dat v databázi.

Detailní náhled historie

Nyní je možné také nastavit náhled informací ohledně polohy uživatele. Po vytvoření dalšího kontroléru (*Detail View Controller*) a odpovídající třídy (*DetailViewController*) je možné nastavit cestu k této obrazovce. Nastavení proběhne ve třídě *HistoryViewController*, v metodě *didSelectRowAt*. V tomto případě metoda definuje, co

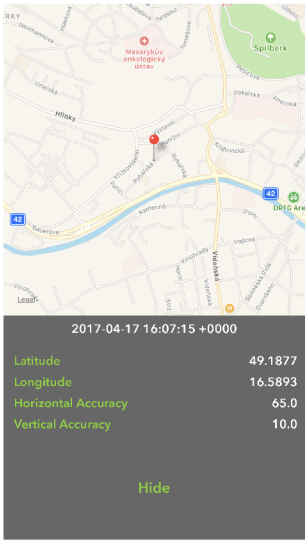
se stane s buňkou po jejím stisknutí. Požadováno je, aby došlo k zobrazení detailní obrazovky.

Výpis 4.30: Nastavení zobrazení detailní obrazovky.

```
let storyboard = UIStoryboard(name: "Main", bundle: nil)
let detailView = storyboard.instantiateViewController
(withIdentifier: "DetailController")
 as! DetailViewController
detailView.task = tasks[indexPath.row]
self.present(detailView, animated:true, completion: nil)
```

Detailní obrazovka obsahuje jedenáct objektů, přičemž devět je typu *UILabel*, jeden typu *MapView* a poslední ve formě tlačítka, které slouží pro návrat do tabulky. Osm zmíněných objektů prvního typu znázorňují název požadované informace a k nim přiřazené hodnoty. Devatý znázorňuje časový údaj, jenž slouží pro lepší přehlednost. Tento údaj je také využit pro umístění nejnovější buňky v tabulce na nejvyšší pozici.

Zobrazení požadovaných dat funguje na stejném principu, jako u tabulky *History*, opět se tedy jedná o výtah uložených záznamů v atributech entity *WiFiValues*.



Obr. 4.18: Struktura tabulky databáze.

4.9.5 Nastavení odesílání dat na server

Stejné konstanty lze využít pro ukládání dat do databáze na serveru univerzity VUT. Databáze byla vytvořena pomocí MySQL. Pro úspěšný záznam odpovídá

server HTTP 202 a HTTP 400 pro neúspěšný[6]. Ukládání do databáze na serveru je

Column	Type	Null	Default	Comments	MIME
id	int(10)	No			
routerIp	varchar(64)	No			
longitude	float	No			
latitude	float	No			
verticalAccurancy	float	No			
horizontalAccurancy	float	No			
ssid	varchar(64)	No			
ipAddress	varchar(64)	No			
subnetMask	varchar(64)	No			
timeStamp	float	No			
publicIp	varchar(64)	No			

Obr. 4.19: Struktura tabulky databáze.

znázorněno níže ve výpisu 4.29.

Výpis 4.31: Ukládání dat na server.

```
var request = URLRequest(url: URL(string:
"http://wislabres.utko.feec.vutbr.cz/qoe/networks/
saveData.php")!)
request.httpMethod = "POST"
let postString = (rpiterIP=\(String(describing:
task.gateway))&longitude=\(zemDelka)&latitude=
\(zemSirka)&verticalAccurancy=\(varPresnost)&
horizontalAccurancy=\(horPresnost)&ssid=
\(String(describing: task.ssid))&ipAddress=
\(String(describing: task.ip))&subnetMask=
\(String(describing: task.netmask))")
request.httpBody = postString.data
(using: .utf8)
let databaze = URLSession.shared.dataTask
(with: request) {data, response, error in
guard let data = data, error == nil else {
print("error=\(String(describing: error)")
return
}
if let httpStatus = response as? HTTPURLResponse,
```

[6]Do databáze je možné nahlédnout pomocí odkazu `http://wislabres.utko.feec.vutbr.cz/qoe/networks/showData.php`.

```
httpStatus.statusCode != 202 {
print("statusCode should be 202, but is
\(httpStatus.statusCode)")
print("response = \(String(describing: response)")
}
let responseString = String(data: data, encoding:
.utf8)
print("responseString = \(String(describing:
responseString))")
}
databaze.resume()
```

4.9.6 Nastavení tlačítka pro odesílání na server

Jak již bylo zmíněno, uživatel bude mít možnost ovlivnit, zda se data budou posílat na server (4.9.1/Konfigurace obrazovky nastavení). *Switch* je potřebné přidat do třídy dvakrát. Jednou jako proměnnou (*nastaveniSwitch*) a podruhé jako funkční tlačítko (*nastaveniChanged*). Důležité je však nastavit, aby tlačítko zůstalo ve stejné poloze i po novém zapnutí aplikace. Toho je možné docílit pomocí přiřazení klíče objektu *UISwitch*.

Výpis 4.32: Přiřazení klíče funkčnímu tlačítku.

```
{
  let userDefaults = userDefaults.standard
  userDefaults.set(sender.isOn, for key: "dbswitch")
  userDefaults.synchronize()
}
```

Dále je potřebné přidat podobný kód do funkce *ViewDidLoad()*, aby aplikace zjistila, v jaké poloze se *switch* nachází.

Výpis 4.33: Zjištění polohy funčkníh tlačítka.

```
{
  let userDefaults = userDefaults.standard
  let switchIsOn : Bool = (userDefaults.bool
  (forKey: "dbSwitch"))
  nastaveniSwitch.isOn = switchIsOn
}
```

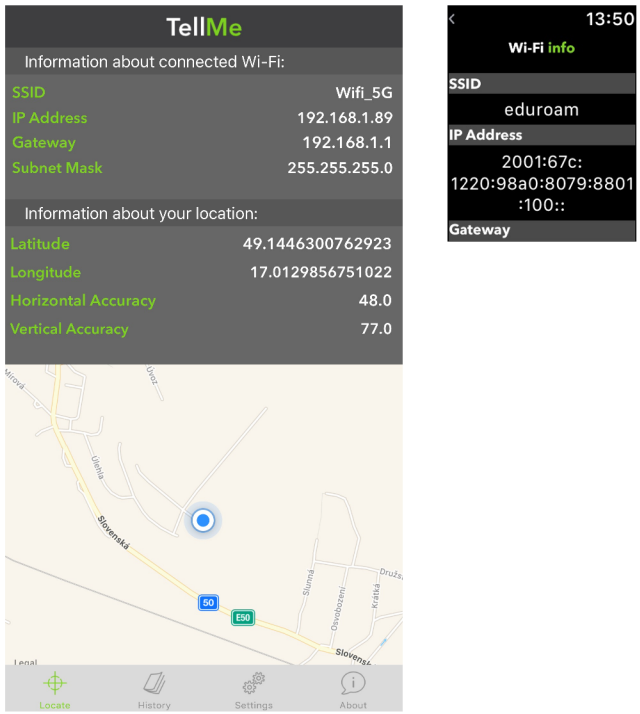
Nyní stačí vložit zdrojový kód pro posílání dat na server (výpis 4.31) do vytvořené podmínky. Pokud budou splněny požadavky, data budou odesílány na server.

Výpis 4.34: Umístění zdrojového kódu funkčního tlačítka.

```
let userDefaults = userDefaults.standard
let canSendToDatabase : Bool =
(userDefaults.bool(forKey: "dbSwitch"))
if canSendToDatabase {
... //Zdrojový kód databáze.
}
```

4.10 Design aplikací

Po zprovoznění veškerých funkcí, jenž byly požadovány, je nyní možné zabývat se designem aplikací. Ideální je zachování stejného designu pro obě aplikace, přičrmž by měl být jednoduchý a zároveň elegantní. V aplikacích jsou celkověp použity tři barvy, kde jedna je použita jen na pozadí (šedá barva #494949). Další použité barvy (bílá #FFFFFF a zelená #92CF48) jsou použity pro nadpisy a veškeré další objekty. Texty jsou typu fontu *Avenir Next, Demi Bold*. Použití těchto barev je znázorněno na obrazovkách na obr. č. 4.20.



Obr. 4.20: Znázorněné barvy na obrazovkách.

4.10.1 Loga aplikací

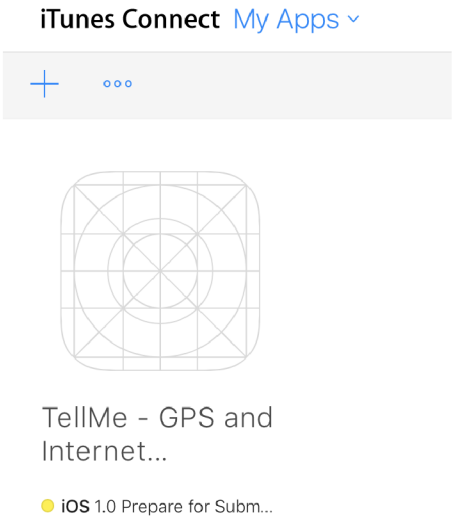
Loga aplikací byla vytvořena pomocí aplikace *Sketch*. Loga jsou na obou typech platformách zcela stejná, kde jsou opět použita stejná kombinace barev. Font názvu aplikací je *Avenir Next, Bold*. Pro mobilní zařízení iPhone je potřebné vytvořit logo ve dvou formátech, jeden typ pro iPhone 6/7 plus a druhý pro zbývající typy. Pro verzi iPhone plus je nutné nastavit velikost ikony 180px x 180x (*60pt x 60px @3x*) a pro ostatní typy velikost 120px x 120px (*60pt x 60pt @2x*). Pro přidání log do aplikací je nutné zobrazit složku *Assets.xcassets* (u obou platforem) a zde loga vložit do souboru *AppIcon* dle požadovaných velikostí. Stejné logo (v jiných velikostech) je použito i pro notifikace, nastavení, atd. Loga aplikací pro iPhone a chytré hodinky Apple Watch jsou znázorněna na obr. č. 4.21.

Obr. 4.21: Ikony aplikací.

4.11 Testování aplikací a odeslání do AppStore

Před odesláním aplikací do virtuálního obchodu *AppStore*, a tím umožnit uživatelům jejich stažení, je doporučené aplikace nejprve otestovat. Tím lze předejít určitým komplikacím a problémům, které by mohly významně poškodit chod ostrých aplikací. Testování aplikací proběhne pomocí aplikace *TestFlight* a frameworku *Fabric*[7]. Aplikace *TestFlight* umožní pozvaným uživatelům aplikace stáhnout a poté dát vývojáři zpětný feedback. Každému pozvanému testerovi dorazí e-mailová pozvánka a možnost stáhnutí aplikací.

Pro zprovoznění aplikací do testovaného režimu je nutné vytvořit iOS aplikaci (testování aplikace pro Apple Watch je zde propojené s iOS aplikací) na vývojářském účtu[8].



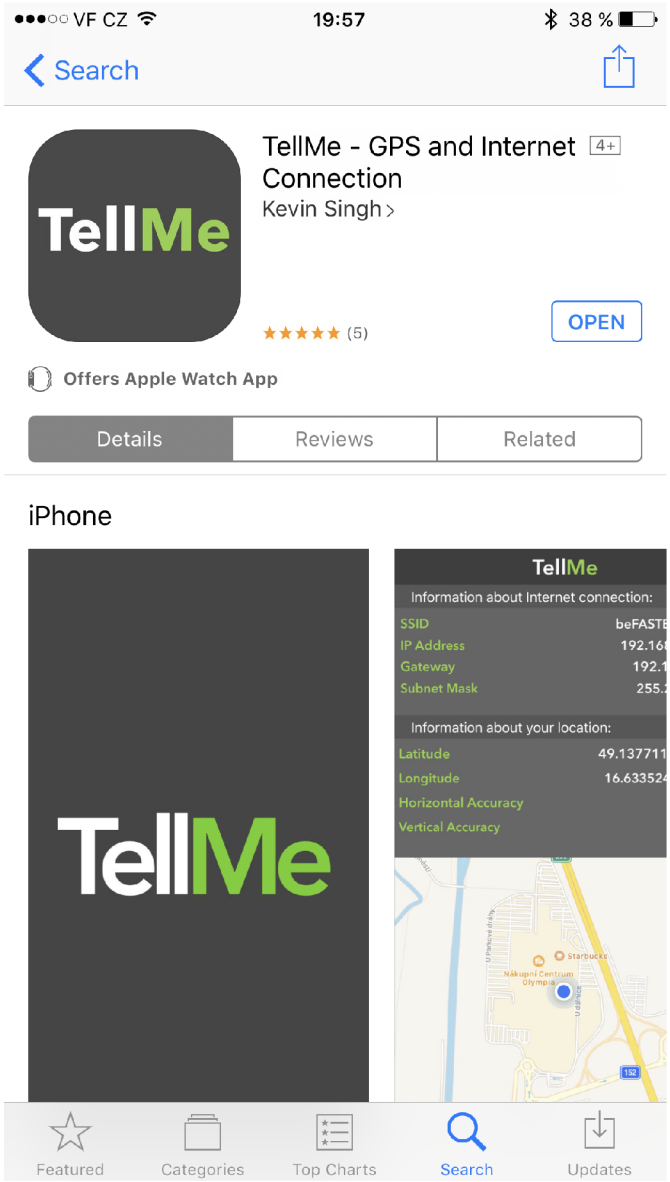
Obr. 4.22: Přidaná aplikace k testování.

Poté je možné aplikace archivovat z vývojového prostředí a zanedlouho se obsah aplikace zobrazí ve vytvořené aplikaci na vývojářském účtu. Nyní je možné spustit testování. Než však bude vývojáři umožněno testování spustit, zašle se aplikace automaticky společnosti *Apple* k recenzi funkčnosti. Pokud vše proběhne v pořádku, vývojáří dorazí e-mail o schválení aplikace k testování či vložení do *AppStore*. V opačném případě příjde zpráva, jenž informuje o chybách v aplikacích (na mysli jsou chyby, které například brání spuštění aplikace či použití nepovolených *frameworků* atd.). Po schválení aplikace je již možné uvést aplikaci do testovacího režimu a pozvat další vývojáře k testování.

[7]Reference na framework Fabric - `https://get.fabric.io`.
[8]Vývojářský účet lze založit na stránkách `https://developer.apple.com`.

4.11.1 Nahrání aplikací do obchodu AppStore

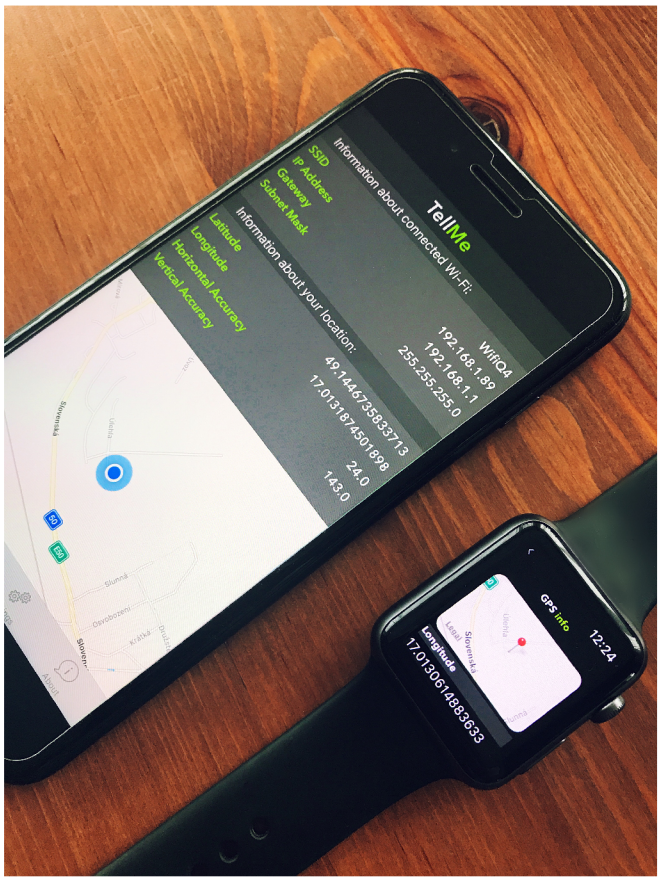
Po opravení případných chyb je nyní možné aplikace nahrát do obchodu *AppStore*. Před samotným vložením je však potřeba doplnit požadované informace. Mezi tyto informace spadá například cena aplikací, jejich definice, žánr apod. Poté jsou aplikace (opravené) opět zaslány k recenzi pro povolení vložení aplikací do obchodu. Tato recenze trvá v průměru jeden až tři dny. Pokud vše proběhne v pořádku a bez problémů, dojde ke vložení vytvořených aplikací do *AppStore*. Aplikace jsou možné k nalezení v obchodě *AppStore* pod názvem *TellMe - GPS and Internet Connection*. Na obr. 4.23 níže je znázorněno umístění v obchodě.



Obr. 4.23: Aplikace umístěné v AppStore.

4.12 Reálné využití aplikace

Na obrázku přiloženém níže lze vidět reálné využití aplikací. Aplikace jsou spuštěné na zařízeních *iPhone 7 plus* a *Apple Watch 42 mm*.



Obr. 4.24: Reálné využití aplikací.

5 ZÁVĚR

Na začátku této bakalářské práce byla popsána definice komunikací M2M (Machine-to-Machine), H2H (Human-to-Human) a D2D (Device-to-Device), která obsahuje i srovnání prvních dvou technologií. V případě komunikace M2M zde bylo provedeno srovnání se způsobem komunikace IoT (Internet of Things), neboť bývá často zaměňována. Dále bylo provedeno porovnání bezdrátových technologií Wi-Fi a ZigBee u M2M, které zobrazuje kladné a záporné stránky obou technogloiií. Oproti Wi-Fi poskytuje ZigBee lepší životnost baterie, ale rychlost přenosu je mnohem nižší.

Další část je věnována nositelné elektronice, která popisuje jednotlivé druhy nejrozšířenějších nositelností. Nechybí zde definice nejprodávanějších výrobků z kategorií chytré náramky, hodinky a oblečení.

Dále je zde zmíněna teorie ohledně chytrých hodinek se systémem WatchOS Apple Watch. Nechybí zde ani popis jednotlivých verzí tohoto systému a hodinek.

Další části bakalářské práce jsou věnovány vývoji aplikací pro mobilní zařízení iPhone a chytré hodinky Apple Watch. V této části jsou popsány soubory a třídy, které vznikly při tvorbě projektu. Nechybí zde ani stromová struktura aplikace, která znázorňuje a popisuje využití jednotlivých tříd. Dále je popsán způsob komunikace mezi hodinkami a mobilním zařízením, kde je zobrazen i postup ke konfiguraci.

Mobilní aplikace je schopna zobrazovat údaje o připojení k Internetu a dále polohové údaje uživatele. Tyto data je poté možné ukládat do databáze v mobilním zařízení, ale i na server Fakulty elektrotechniky a telekomunikačních technologií. V rámci aplikace pro chytré hodinky je možné získaná data na hodinkách zobrazovat, kde je také možnost zobrazení polohy uživatele na mapě.

Další část obsahuje design aplikací. V této části jsou zahrnuty použité palety barev, ale také veškeré potřebné ikony pro použití v obou aplikacích.

Poslední část je věnována testování aplikací a jejich vložení do obchodu *AppStore*. V rámci této sekce je již také zahrnuto reálné využití aplikací.

LITERATURA

[1] Hospodářské noviny: *Nositelnou elektroniku už trh přijal* [online]. Vydáno 21. 4. 2016. Dostupné z URL: <http://ictrevue.ihned.cz/c3-65260530-0ICT00_d-65260530-nositelnou-elektroniku-uz-trh-prijal>.

[2] ČT24: *M2M: Nová dimenze komunikační technologie* [online]. Vydáno 28. 6. 2011. Dostupné z URL: <http://www.ceskatelevize.cz/ct24/svet/1255400-m2m-nova-dimenze-komunikacni-technologie>.

[3] LinkedIn: *M2M vs IoT, what is the difference?* [online]. Vydáno 16. 8. 2015. Dostupné z URL: <https://www.linkedin.com/pulse/m2m-vs-iot-what-difference-carolina-ibarra-iese-mba?trkInfo=VSRPsearchId%3A517549282147930951674 1%2CVSRPtargetId%3A6038703060321136640%2CVSRPcmpt%3Aprimary&trk=vsrp_influencer_content_res_name>.

[4] LinkLabs: *The ZigBee Vs WiFi Battle For M2M Communication* [online]. Vydáno 2. 11. 2015. Dostupné z URL: <http://www.link-labs.com/zigbee-vs-wifi-802-11ah/>.

[5] ResearchGate: *Analyzing impacts of coexistence between M2M and H2H communication on 3GPP LTE system* [online]. Vydáno 2014. Dostupné z URL: <https://www.researchgate.net/publication/285054635_Analyzing_impacts_of_coexistence_between_M2M_and_H2H_communication_on_3GPP_LTE_system>.

[6] IEEE Xplore: *A ContextAware Backhaul Management solution for combined H2H and M2M traffic* [online]. Vydáno © 2013. Dostupné z URL: <http://ieeexplore.ieee.org.ezproxy.lib.vutbr.cz/xpls/icp.jsp?arnumber=6705719>.

[7] Ryu Seungwan: *Electronic proceedings of the 2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW 2013) podle ISBN 9781479916047* [online]. Poslední aktualizace 3. 10. 2013. Dostupné z URL: <http://ieeexplore.ieee.org.ezproxy.lib.vutbr.cz/document/6618379/>.

[8] Juniper Networks,Inc., 2014 *MACHINE-TO-MACHINE(M2M)-THE RISE OF THE MACHINES* 2011

[9] Cisco: *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper* [online]. Vydáno 1. 2. 2016. Dostupné z URL: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html>.

[10] Ericsson: *Wearable future* [online]. Vydáno 2016. Dostupné z URL: <https://www.ericsson.com/thinkingahead/consumerlab/consumer-insights/wearable-technology-and-the-internet-of-things>.

[11] Fitbit: *fitbit flex 2* [online]. Vydáno © 2016. Dostupné z URL: <https://www.fitbit.com/eu/flex2>.

[12] xiaomimobile.cz: *Xiaomi Mi Band 2* [online]. Vydáno © 2016. Dostupné z URL: <http://xiaomimobile.cz/342-xiaomi-mi-band-2.html>.

[13] Samsung: *Gear S3 Classic* [online]. Vydáno © 2016. Dostupné z URL: <http://www.samsung.com/cz/consumer/mobile-devices/wearables/gear/SM-R770NZSAXEZ/>.

[14] Garmin: *Garmin fenix 3 Silver Performer* [online]. Vydáno 27. 11. 2016. Dostupné z URL: <http://www.garmin.cz/produkty/0/fenix/garmin-fenix3-silver-performer.html>.

[15] AiraWear: *Science of AiraWear* [online]. Vydáno © 2016. Dostupné z URL: <https://airawear.com/index.php/massage-relieves-back-pain/>.

[16] Technet: *Chytré kontaktní čočky od Googlu se ukážou na začátku příštího roku* [online]. Vydáno 16. 7. 2014. Dostupné z URL: <http://technet.idnes.cz/chytre-kontaktni-cocky-google-dec-/hardware.aspx?c=A140715_141049_hardware_vse>.

[17] Apple: *Apple Watch Series 1* [online]. Vydáno © 2016. Dostupné z URL: <http://www.apple.com/apple-watch-series-1/>.

[18] MacRumors: *Apple Watch Losing Ground to „Basic Wearables“ Like Fitbit Ahead of New Models* [online]. Vydáno 16. 9. 2016. Dostupné z URL: <http://www.macrumors.com/2016/09/06/apple-watch-losing-ground-fitbit/>.

[19] Apple: *Apple Watch OS 1.0.1* [online]. Vydáno 191. 5. 2015. Dostupné z URL: <https://support.apple.com/kb/dl1812?locale=en_US>.

[20] TechRadar: *Apple watchOS 2: everything you need to know* [online]. Vydáno 14. 7. 2016. Dostupné z URL: <http://www.techradar.com/news/wearables/apple-watch-os-2-release-date-news-and-features-1296413/>

[21] Letem světem Applem: *První pohled na novinky v novém watchOS 3* [online]. Vydáno 14. 6. 2016. Dostupné z URL: <https://www.letemsvetemapplem.eu/2016/06/14/pohled-na-novinky-v-novem-watchos-3/>.

[22] Apple: *The Watch App Architecture* [online]. Vydáno 25. 11. 2016. Dostupné z URL: <https://developer.apple.com/library/content/documentation/General/Conceptual/WatchKitProgrammingGuide/DesigningaWatchKitApp.html>.

[23] Apple: *Watch series 2* [online]. Vydáno 2016. Dostupné z URL: <http://www.apple.com/cz/apple-watch-series-2/>.

[24] Apple: *Compare Apple Watch models* [online]. Vydáno © 2016. Dostupné z URL: <http://www.apple.com/lae/watch/compare/>.

[25] Apple: *Xcode 8* [online]. Vydáno 25. 11. 2016. Dostupné z URL: <https://developer.apple.com/xcode/>.

M2M: Nová dimenze komunikační technologie [online]. Vydáno 28. 6. 2011. Dostupné z URL: <http://www.ceskatelevize.cz/ct24/svet/1255400-m2m-nova-dimenze-komunikacni-technologie>.

[26] SketchApp: *Profesional digital design for Mac* [online]. Vydáno 2016. Dostupné z URL: <http://sketchapp.com>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ARPD	Average Relative Percentage Deviation
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
D2D	Device-to-Device
GPS	Global Positioning System
GPU	Graphics processing unit
H2H	Human-to-Human
IoT	Internet of Things
LoRa	Long Range
LTE	Long Term Evolution
M2M	Machine-to-Machine
NB-IoT	NarrowBand IoT
OS	Operační systém
QoS	Quality of Service
SIM	Subscriber Information Module
SWOLF	Swim Golf
Wi-Fi	Wireless Fidelity
WiGig	Wireless Gigabit Alliance
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network

A OBSAH PŘILOŽENÉHO CD

K bakalářské práci je také přiloženo elektronické médium ve formě CD, jenž se nachází na zadní části desek. Na médiu je také obsažena elektronická verze bakalářské práce, návrh designu a zdrojový kód aplikací, jenž byl vytvořený ve vývojovém prostředí Xcode, jazyce Swift.