VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO BEZPEČNOST DOMÁCNOSTI

VISUAL HOME SECURITY SYSTEM FOR IOS–BASED MOBILE DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. FILIP BAJANÍK**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/20910/2017/xbajan01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Bajaník Filip, Bc.**
Obor: Informační systémy
Téma: **Mobilní aplikace pro bezpečnost domácnosti**
Visual Home Security System for iOS-Based Mobile Devices
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou návrhu a vývoje mobilních aplikací; zaměřte se na platformu iOS.
2. Seznamte se s problematikou počítačového vidění, zaměřte se na algoritmy relevantní pro řešenou aplikaci.
3. Vyhledejte a analyzujte existující aplikace řešící podobný problém.
4. Navrhněte a prototypujte způsob interakce s aplikací a jednotlivé prvky uživatelského rozhraní.
5. Navrhněte, prototypujte a na vhodných datech vyhodnoťte algoritmy počítačového vidění relevantní pro řešený problém.
6. Navrhněte a implementujte řešenou aplikaci.
7. Testujte vytvořenou aplikaci na uživatelích a iterativně ji vylepšujte.
8. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN-13: 978-0321965516
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 až 6.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cieľom tejto diplomovej práce je návrh a implementácia mobilnej aplikácie pre bezpečnosť domácnosti na platforme iOS. Aplikácia predstavuje komplexné riešenie, umožňujúce prenos audia a videa medzi spárovanými mobilnými zariadeniami, s využitím technológie WebRTC. Výsledný modul predstavuje univerzálne riešenie pre peer-to-peer video a audio komunikáciu. Práca taktiež rieši problematiku počítačového videnia a konkrétne algoritmy efektívnej detekcie pohybu. Výsledkom je modul implementujúci algoritmus ViBe s použitím knižnice Metal. V prípade detekovaného pohybu aplikácia upozorní používateľa push notifikáciou. Synchronizácia aplikačných dát je realizovaná technológiou CloudKit a perzistencia dát knižnicou Realm.

Abstract

The goal of this diploma thesis is to design and implement a mobile application for home security system on the iOS platform. The application introduces a complex solution allowing the transmission of the audio and video streams between the paired mobile devices using WebRTC. The final module represents universal solution for peer-to-peer audio and video communication. The thesis also deals with the field of computer vision, namely efficient motion detection algorithms. The module for motion detection implements ViBe algorithm using Metal. In case that the motion is detected the application notifies a user with a push notification. Synchronization of application data is implemented using Cloudkit and the data persistance using Realm library.

Kľúčové slová

WebRTC, Real Time Communication, iOS, video, RTCPeerConnection, peer-to-peer, P2P, ViBe, Motion detection, MQTT, Frame differencing, Running average, Metal, Cloudkit, iCloud, bezpečnosť domácnosti

Keywords

WebRTC, Real Time Communication, iOS, video, RTCPeerConnection, peer-to-peer, P2P, ViBe, Motion detection, MQTT, Frame differencing, Running average, Metal, Cloudkit, iCloud, home security

Citácia

BAJANÍK, Filip. *Mobilní aplikace pro bezpečnost domácnosti.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Mobilní aplikace pro bezpečnost domácnosti

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením profesora Ing. Adama Herouta, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Filip Bajaník
23. mája 2018

Poďakovanie

Rád by som poďakoval profesorovi Ing. Adamovi Heroutovi, Ph.D. za vedenie tejto diplomovej práce, odbornú pomoc a cenné rady pri dosahovaní cieľa. Ďalej by som sa chcel poďakovať kolegom a kolegyniam za motiváciu pri boji s touto prácou. V neposlednom rade ďakujem rodine a priateľom za podporu.

Obsah

1 Úvod **3**

2 Vývoj na platforme iOS **4**
2.1 Jazyk Swift . . . 4
2.2 Životný cyklus aplikácie . . . 5
2.3 Grafický framework – UIKit . . . 6
2.4 CloudKit . . . 8
2.5 Využité knižnice . . . 11

3 Technológie pre prenos audia a videa **13**
3.1 Prenos mediálnych dát – WebRTC . . . 13
3.1.1 Architektúra . . . 14
3.1.2 ICE framework . . . 15
3.1.3 STUN servery . . . 17
3.1.4 TURN servery . . . 17
3.1.5 Naviazanie spojenia . . . 17
3.1.6 Bezpečnosť komunikácie . . . 21
3.2 Signalizácia pomocou MQTT . . . 22
3.2.1 Komunikácia a prenášané dáta . . . 22
3.2.2 Bezpečnosť . . . 24

4 Detekcia pohybu a analýza algoritmov **25**
4.1 Detekčné algoritmy . . . 25
4.1.1 Background subtraction . . . 25
4.1.2 Frame differencing . . . 26
4.1.3 Approximate median filter . . . 27
4.1.4 Running average . . . 27
4.2 ViBe . . . 28
4.2.1 Model pixelov a klasifikačný proces . . . 28
4.2.2 Aktualizácia modelu . . . 29
4.2.3 Inicializácia modelu . . . 30
4.3 Algoritmy potrebné pri implementácii vylepšeného algoritmu ViBe . . . 30
4.3.1 Základné binárne morfologické operácie . . . 31
4.4 Technológie . . . 32
4.4.1 Metal . . . 32
4.4.2 OpenGL ES . . . 34

5 Analýza funkcionality a návrh používateľského rozhrania **36**

5.1 Analýza funkcionality vybraných aplikácií . . . 36
5.2 Kompletný prehľad funkcionality . . . 39
5.3 Špecifikácia požiadaviek . . . 41
5.4 Návrh používateľského rozhrania . . . 42

6 Realizácia **46**
6.1 Realizácia modulu pre detekciu pohybu (VibeDetection) . . . 46
6.1.1 Implementácia prototypu algoritmom Running average . . . 46
6.1.2 Implementácia algoritmu ViBe . . . 47
6.2 Realizácia modulu pre prenos audia a videa (VibeHomeCore) . . . 53
6.3 Realizácia výslednej aplikácie (VibeHome) . . . 58
6.3.1 Prvý prototyp synchronizácie zariadení . . . 58
6.3.2 Synchronizácia dát pomocou CloudKitu . . . 59
6.3.3 Notifikácie . . . 61
6.3.4 Perzistentné úložisko . . . 61
6.3.5 Používateľské rozhranie (UI) . . . 61
6.3.6 Integrácia modulu na prenos audia a videa (VibeHomeCore) . . . 62
6.3.7 Integrácia modulu na detekciu pohybu (VibeDetection) . . . 62
6.4 Metriky kódu . . . 62

7 Testovanie **63**
7.1 Spôsob testovania . . . 63
7.2 Testovanie prvého prototypu aplikácie . . . 64
7.3 Finálne používateľské testovanie . . . 65
7.4 Testovanie výdrže . . . 69
7.5 Návrhy na zlepšenie . . . 69

8 Záver **71**

Literatúra **72**

Prílohy **77**

A Používateľské rozhranie ďalších najpopulárnejších aplikácií **78**

B Podoba výslednej aplikácie **81**

C Obsah CD **82**

Kapitola 1

Úvod

Téma inteligentnej domácnosti sa stala v posledných rokoch s príchodom *IoT*[1] veľmi populárna. S inteligentnou domácnosťou súvisí aj samotná bezpečnosť domácnosti. Tá sa dá zaručiť napríklad *IP kamerami*, avšak dnes má mnoho ľudí v dôsledku zastarávania spotrebnej elektroniky doma nevyužité mobilné zariadenie. Zariadenie je vďaka dnešnému rýchlemu vývoju mobilných technológií už morálne zastaralé, avšak so správnym softwarovým vybavením dokáže vhodne poslúžiť pre bezpečný dohľad nad domácnosťou.

Diplomová práca si dáva za cieľ navrhnúť a implementovať komplexnú aplikáciu pre bezpečnosť domácnosti, umožňujúcu vzdialený dohľad nad domácnosťou prostredníctvom spárovaného mobilného zariadenia. Pre implementáciu *peer-to-peer* video a audio komunikácie je použitá technológia *WebRTC*, ktorá bola vyvinutá a štandardizovaná iba v posledných rokoch. Samotná technológia *WebRTC* zaručuje prenos audio a video tokov, avšak pre nadviazanie spojenia vyžaduje implementáciu *signalizačného* mechanizmu. Za vhodný signalizačný mechanizmus bola zvolená open–source technológia *MQTT*, ktorá je známa svojou nenáročnosťou a vhodnosťou pre zariadenia *IoT*. Súčasťou práce je aj implementácia synchronizácie dát a ich perzistencia na zariadení..

Dôležitou súčasťou pre bezpečnosť domácnosti je detekcia pohybu. Detekcia pohybu patrí medzi problematiku oboru počítačového videnia. Práca sa venuje konkrétnym algoritmom, ktoré sú nenáročné na pamäť a výpočtový výkon. Taktiež je analyzovaný veľmi populárny a patentovaný detekčný algoritmus *ViBe*. Výsledná aplikácia má prívetivé používateľské rozhranie a ponúka sadu užitočných funkcií, ako napríklad upozornenie v prípade detekovaného pohybu alebo galériu zachytených snímkov.

Aktuálne rozloženie kapitol je nasledovné. Kapitola 2 sa venuje dôležitým oblastiam vývoju mobilných aplikácií pre platformu `iOs` a technológiám, ktoré sú na tejto platforme využité. Kapitola 3 je venovaná technológiám prenosu audia a videa. Algoritmy detekcie pohybu sú rozobraté v kapitole 4. Kapitola 5 analyzuje funkcionalitu podobných aplikácií a špecifikuje požiadavky. Dôležitou kapitolou je kapitola 6, ktorá sa zaoberá realizáciou čiastkových modulov a kompletného riešenia aplikácie. V kapitole 7 sa nachádza používateľské testovanie, testovanie výdrže ale aj návrhy na zlepšenie. Posledná kapitola 8 je venovaná záveru.

[1] **Internet vecí (Internet of Things)** je označenie pre prepojenie vstavaných zariadení s internetom. Prepojené zariadenia by malo byť najmä bezdrôtové a malo by priniesť nové možnosti vzájomnej interakcie.

Kapitola 2

Vývoj na platforme iOS

Súčasťou zadania práce je vytvorenie aplikácie pre mobilnú platformu *iOS*, preto je dôležité zoznámiť sa so základnými stavebnými prvkami danej platformy. Kapitola bude popisovať základnú architektúru operačného systému a behového prostredia. Dôležitou časťou bude aj sekcia o životnom cykle aplikácie a tvorbe používateľského rozhrania. Malá časť bude taktiež venovaná doplnkovej aplikácii pre platformu *WatchOS* bežiacou na zariadeniach *Apple Watch*.

2.1 Jazyk Swift

Tento jazyk bol vytvorený spoločnosťou Apple ako moderný následník jazyka Objective-C. Apple po zverejnení jazyk zverejnil ako open-source[1]. Jeho podpora zo strany vývojárov je obrovská a jazyk je dostupný na všetkých platformách Applu (iOS, WatchOS, AppleTV, MacOS), spolu s platformou Linux a komunita aktívne pracuje aj na ďalších platformách. Jazyk je aktuálne vo verzii 4.1 a okrem zlepšenia stability ABI[2] priniesol vylepšenia pre prácu s reťazcami vo formáte Unicode, rozšírené možnosti pre prácu so slovníkmi a sadami alebo mechanizmus serializácie a deserializácie objektov, ktorý je neoddeliteľnou súčasťou pri práci s dátovou vrstvou aplikácie.

Charakteristika jazyka

- **Bezpečnosť** – Swift sa snaží o elimináciu mnohých problémov, ktoré v ostatných kompilovaných jazykoch spôsobujú problémy. Premenné sú napríklad vždy inicializované pred použitím, polia a čísla sú kontrolované na pretečenie a manažment pamäte je automatický. Syntax je vytvorená tak, aby plnila už od pohľadu plnila svoj zámer – príkladom sú kľúčové slová ako `var`, ktorá vyjadruje premennú (variable) alebo `let`, čo vyjadruje konštantu.

  Ďalšou bezpečnostnou vlastnosťou je, že vo Swifte nemôže byť objekt nikdy `nil`[3]. Prekladač programátora upozorní a v prípade použitia vyhodí `compile-time` chybu. To umožňuje písať čistejší a bezpečnejší kód, pri ktorom sa predíde mnohým pádom počas behu aplikácie.

[1] https://www.swift.org
[2] https://en.wikipedia.org/wiki/Application_binary_interface
[3] V jazyku ako napríklad **C** to predstavuje `NULL`.

Sú však prípady, kedy je použitie `nil`-u validné a vhodné. Pre tieto situácie Swift predstavil inovatívnu funkciu známu ako *optionals* alebo „voliteľné“ premenné. Optional môže obsahovať `nil`, avšak syntax núti programátora k tomu, aby ich používal bezpečne za pomoci operátoru `?`, ktorý napovie kompilátoru, že programátor chápe jeho použitie a použije ho bezpečne.

- **Rýchlosť** – Swift bol navrhnutý tak, aby bol rýchly a jeho architekti k tomu použili prekladač *LLVM*[4], ktorý zdrojové kódy Swiftu transformuje do optimalizovaného natívneho kódu, využívajúceho moderný hardware naplno.

  Keďže je Swift následník pre jazyky **C** a **Objective-C**, zahŕňa primitíva ako typy, riadenie toku alebo operátory. Taktiež ponúka vlasnosti akými sú tvorenie tried, protokolov alebo aj generiká.

- **Interoperabilita s jazykom Objective-C** – Pomocou Swiftu je možné tvoriť nové aplikácie, ale taktiež je možné používať ho s aplikáciami, písanými v Objective-C a pristupovať k Objective-C rozhraniam.

Ďalšie vlastnosti, ktoré robia kód expresívnejším:

- Uzávery (*closures*) zjednotené s ukazovateľmi na funkcie
- N-tice alebo viacero návratových hodnôt
- Generiká
- Stručné spôsoby iterácie cez kolekcie (podmienená iterácia)
- Štruktúry podporujúce metódy, rozšírenia a protokoly
- Funkcionálne prvky (map, reduce, filter)
- Natívne spracovanie chýb použitím `try`, `catch` alebo `throw`

2.2 Životný cyklus aplikácie

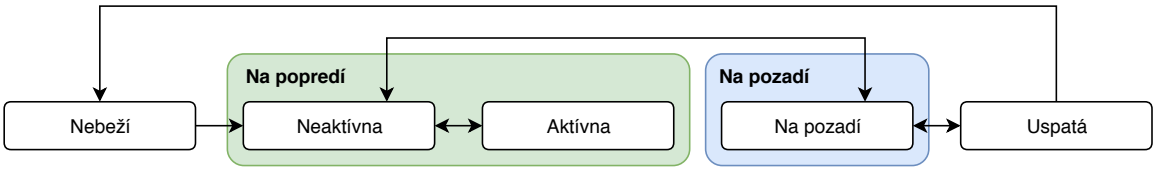
V akýkoľvek moment sa aplikácia nachádza v jednom zo stavov znázornených v diagrame 2.1. Systém presúva aplikáciu z jedného stavu, do stavu druhého, ako rekciu na podnety, ktoré sa dejú v celom systéme. Ak napríklad popoužívateľ stlačí `Home`[5] tlačidlo alebo začne prichádzať hovor, aktuálne bežiaca aplikácia zmení svoj stav.

Popis stavov:

- **Nebeží** – Aplikácia nebola spustená alebo bola terminovaná systémom.
- **Neaktívna** – Aplikácia beží na popredí, ale momentálne neprijíma žiadne udalosti. Avšak kód aplikácie sa vykonávať môže. V tomto stave sa aplikácia nachádza len v momente, kedy prechádza iného stavu.
- **Aktívna** – Aplikácia beží na popredí a prijíma udalosti od používateľa.

[4] https://llvm.org/
[5] **Tlačidlo Home** – hlavné tlačidlo umiestnené v dolnej časti telefónu.

- **Na pozadí** – Aplikácia je na pozadí a vykonáva kód. Väčšina aplikácií sa dostáva do tohto stavu zriedka, kedy sa ich stav zmení na uspatý. Avšak aplikácia vyžadujúca dokončenie napríklad sieťovej operácie zostáva v tomto stave dlhšie.
- **Uspatá** – Aplikácia je na pozadí a zostáva v operačnej pamäti, ale nevykonáva žiadny kód. Systém aplikáciu presúva do tohto módu automaticky. Ak systému dochádza pamäť, automaticky z nej uspanú aplikáciu odstráni.



Obr. 2.1: Diagram znázorňuje životný cyklus aplikácie a prechod medzi stavmi.

2.3 Grafický framework – UIKit

Hlavným komponentom zobrazujúcim používateľské rozhranie je `UIViewController`. Aplikácia sa obvykle skladá z viacero `UIViewController`ov. Ich usporiadanie je hierarchické.

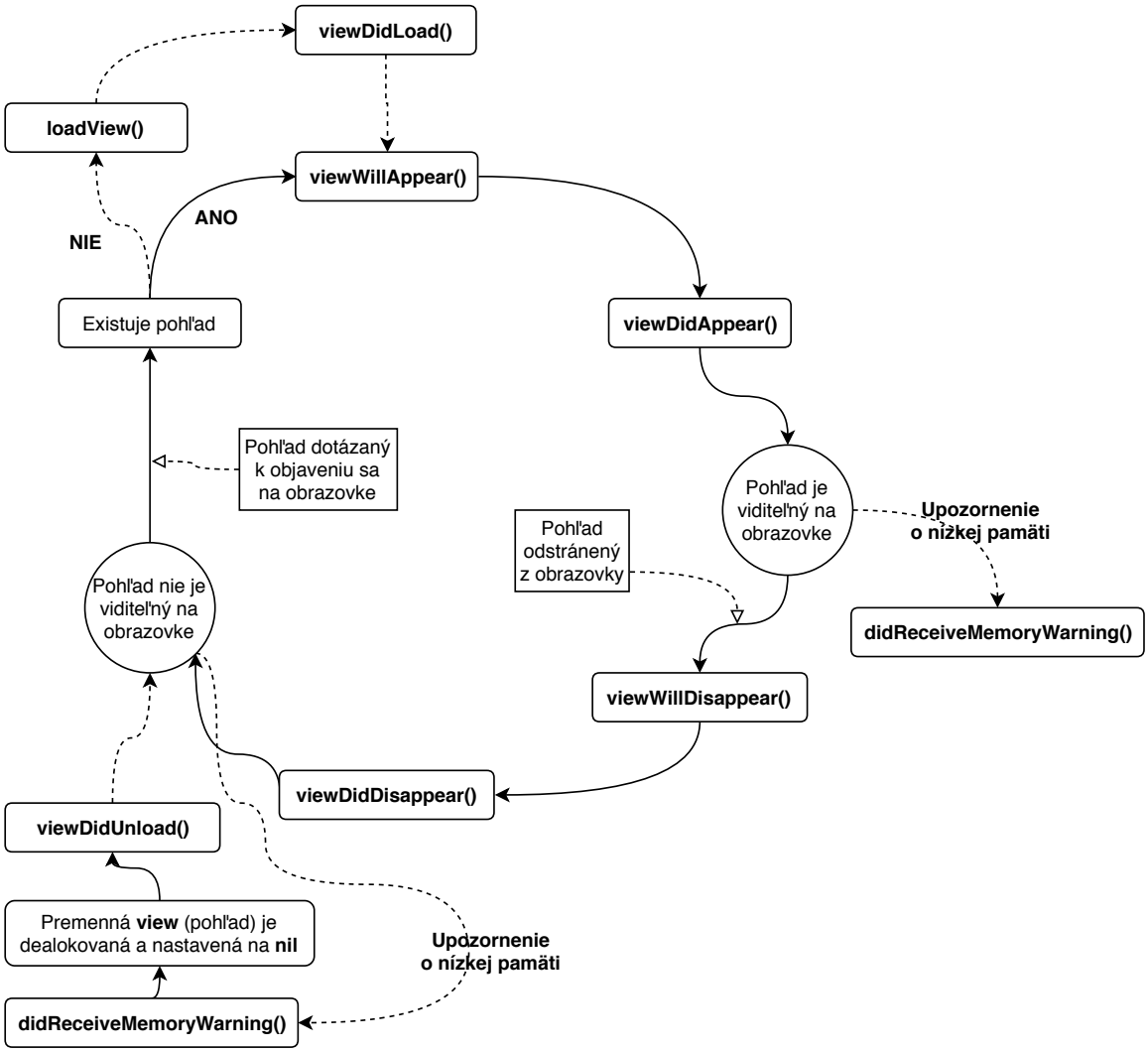
Diagram 2.2 znázorňuje životný cyklus `UIViewController`-u a jeho dôležitých metód, ktoré je potrebné preťažiť pre dosiahnutie dobrého UX[6]. Následuje zhrnutie najdôležitejších metód a ich prípadné použitie:

- **viewDidLoad()** – Volá sa v momente, kedy je najvrchnejší pohľad vytvorený a načítaný. Používa sa na dodatočné nastavenie pohľadov a registráciu observerov.
- **viewWillAppear()** – Volá sa pred pridaním najvrchnejšieho pohľadu do aplikačnej hierarchie. Používa sa na operácie, ktoré potrebujú byť vykonané pred tým, než sa pohľad zobrazí na obrazovku.
- **viewDidAppear()** – Volá sa po pridaní najvrchnejšieho pohľadu do aplikačnej hierarchie. Používa sa na vykonanie operácií, ktoré potrebujú byť vykonané hneď po tom, čo je pohľad zobrazený na obrazovke, ako napríklad načítanie dát alebo zobrazenie animácie.
- **viewWillDisappear()** – Volá sa pred tým, ako sa najvrchnejší pohľad odstráni z aplikačnej hierarchie. Používa sa napríkad na uloženie perzistentných dát.
- **viewDidDisappear()** – Volá hneď po odstránení najvrchnejšieho pohľadu z aplikačnej hierarchie alebo keď bol pohľad prekrytý alebo skrytý.

Tvorba používateľského rozhrania

Používateľské rozhranie je možné tvoriť rôznymi spôsobmi. Oba spôsoby využívajú systému `AutoLayout`, ktorý podľa nastavených obmedzení zostaví prvky používateľského rozhrania.

[6] **User experience** – používateľský zážitok je aspekt ľudskej interakcie s daným systémom, zahŕňajúci rozhranie, grafiku, priemyslový design, fyzickú interakciu a manuál.



Obr. 2.2: Zjednodušený diagram znázorňuje životný cyklus základnej aplikačnej komponenty **UIViewController**, ktorá tvorí základný stavebný kameň iOS aplikácie.

Jedným so spôsobov je zostavenie UI v kóde. Toto API sa však ťažko používa. Z toho dôvodu vznikli knižnice, ako napríklad `SnapKit` (popísaný v sekcii 2.5), ktoré uľahčujú prácu s `AutoLayotom`.

Ďalším spôsobom je využitie `Interface Builderu`, ktorý slúži pre tvorbu používateľským rozhraní formou WYSIWYG[7] editoru. `Builder` automaticky zostaví obmedzenia podľa toho, čo používateľ nástroju `XCode` vytvorí. Kompletná aplikácia je vytvorená z viacerých pohľadov, cez ktoré sa používateľ naviguje. Vzťahy medzi nimi sú definované tzv. `Storyboard`ami, ktoré zobrazujú kompletný pohľad na aplikačnú flow. `Interface Builder` využíva na zostavenie UI objekty ako napríklad spomínaný `UIViewController`, `UIButton` a podobne.

[7]**WYSIWYG** – angl. What You See Is What You Get – označuje spôsob editácie dokumentov, pri ktorom je zobrazovaná verzia totožná s výslednou verziou dokumentu.

2.4 CloudKit

CloudKit je framework od spoločnosti Apple, ktorý poskytuje rozhrania pre prenášanie dát medzi aplikáciami a `iCloud`[8] kontajnermi. CloudKit sa používa na ukladanie dát do používateľovho iCloud účtu, aby mal používateľ dáta dostupné odkiaľkoľvek a v každom jeho zariadení. Dáta umiestnené v iCloud-e je možné taktiež zdieľať alebo využívať verejný kontajner na prístup k verejným dátam, ku ktorým môžu pristupovať všetci používatelia (blog, diskusné fórum...).

CloudKit neslúži ako náhrada dátovej aplikačnej vrstvy, ale poskytuje doplnkovú službu pre správu prenosu dát medzi používateľom a iCloud servermi. CloudKit poskytuje minimálnu podporu pre offline cachovanie a spolieha na internetové pripojenie a validným iCloud účtom[9], ktorý je však potrebný len v prípade zápisu do privátneho používateľského úložiska.

Vhodnou poznámkou je, že triedy CloudKitu neboli navrhnuté, aby mohla byť využitá dedičnosť. Pri získavaní a ukladaní dát sa musia používať tak ako boli navrhnuté – vo forme slovníku.

Od iných konkurenčných riešení má veľkú výhodu v tom, že nevyžaduje registráciu a používateľ má svoje dáta okamžite k dispozícii prostredníctvom internetu. Ďalšou veľkou výhodou je to, že dáta súkromných databáz sa neúčtujú do kvóty vývojára, ale používateľa, čo šetrí náklady na prevádzku a údržbu.

Záznam

Záznamy (`Records`) sú kľúčovým prvkom všetkých transakcií v CloudKite. Záznam je slovník, obsahujúci hodnoty typu kľúč-hodnota a reprezentuje dáta, ktoré chce programátor uložiť. Do záznamov je možné kedykoľvek položky pridávať alebo odoberať. Taktiež je možné vytvárať referencie na iné záznamy a vytvoriť tak hierarchickú štruktúru dát. Trieda **CKRecord** definuje rozhranie pre správu obsahu záznamov. Položky záznamov je možné indexovať na vyhľadávanie alebo zoraďovanie. Záznamy môžu obsahovať položky základných primitívnych dátových typov, ale aj surových dát alebo objekty obsahujúce lokalizáciu zariadenia a to s použitím záznamu typu `CKAsset`.

Kontajner

Každý aplikácia má svoj kontajner, čo reprezentuje objekt, ktorý spravuje všetky implicitné a explicitné pokusy o prístup k obsahu kontajnera. Každá aplikácia má svoj predvolený kontajner, ktorý spravuje svoje vlastné dáta. Ak sa však vyvíja skupina aplikácií, zdieľajúcich dáta, je možné kontajnery zdieľať. Každý kontajner rozlišuje medzi verejne dostupnými a súkromnými údajmi.

Každý používateľ, ktorý pristupuje ku kontajneru, má príslušný popoužívateľský záznam v danom kontajnery. Podľa predvoleného nastavenia neobsahujú používateľské záznamy žiadne osobné informácie o popoužívateľovi a ani o jeho účte iCloud. Záznamy poskytujú spôsob, ako odlíšiť jedného používateľa od iného pomocou jedinečného identifikátora. Každý používateľ má v kontajnery svoj vlastný jedinečný identifikátor.

[8] **iCloud** je serverové riešenie, ktoré poskytuje používateľovi úložisko a možnosť okamžitej synchronizácie dát medzi viacerými zariadeniami.

[9] Väčšina používateľov Apple zariadení aktívne používa iCloud účet, ktorý je potrebný na sťahovanie aplikácií a mnohými ďalšími úkonmi spojenými s Apple ekosystémom.

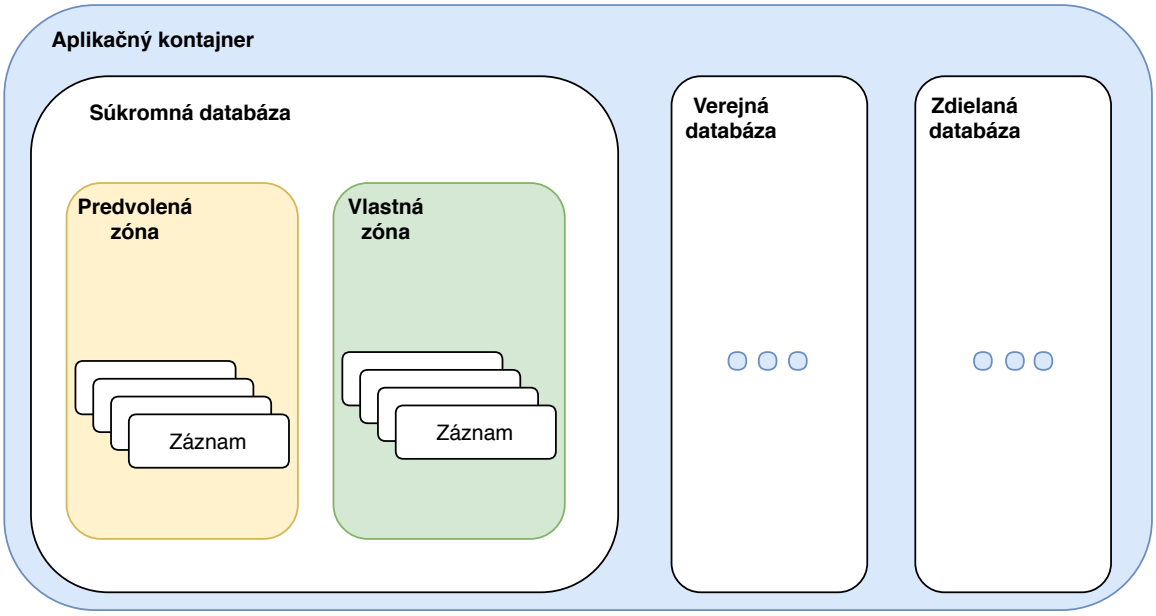
Databáza

Každý kontajner obsahuje odkazy na verejnú, súkromnú a zdieľanú databázu pre ukladanie dát. Obsah verejnej databáze je dostupný pre všetkých používateľov databáze, zatiaľ čo obsah súkromnej databáze je štandardne dostupný len pre samotného používateľa. Dáta verejnej databáze sa vzťahujú do kvóty vývojára aplikácie. Súkromné dáta sú vždy uložené v súkromnej databáze a na objem sa vzťahuje obmedzenie dané používateľskou dátovou kvótou, čo má svoje nesporné výhody. Vývojár tak nemusí platiť za databázové riešenie a jeho úložisko a používateľ má dáta na svojom súkromnom účte.

Zóna

Zóna je dôležitou súčasťou spôsobu usporiadania dát. Verejné, súkromné a zdieľané databázy majú jednu predvolenú zónu. V súkromnej databáze je možné použiť objekty `CKRecordZone` na vytvorenie ďalších vlastných zón podľa potreby. Použitie je vhodné na usporiadanie skupiny k sebe patriacich záznamov. Vlastné zóny majú však aj iné nesporné výhody ako napríklad schopnosť zapísať viacero záznamov ako jedinú atomickú operáciu a mnohé ďalšie.

Ukážka hierarchie spomínaných objektov je zobrazená na obrázku 2.3.



Obr. 2.3: Zobrazenie základných komponent CloudKit-u a ich zapúzdrenie.

Referencie

Objekt `CKReference` slúži na vytvorenie viacnásobnej väzby (*many-to-one*) medzi záznamami v databáze. Každý objekt referencie obsahuje informáciu o jednom zázname. Tento odkaz sa priradí do slovníku záznamu a prepojí sa s ďalším záznamom. Oba záznamy musia byť z rovnakej zóny, rovnakej databázy. Referencie tvoria silnú väzbu medzi záznamami a môžu obsahovať príznak akcie, čo sa s nimi má stať v prípade vymazania rodičovského záznamu. Tak je možné dosiahnuť vymazanie podradených záznamov pri vymazaní rodičovského záznamu.

Zdieľané záznamy

CloudKit umožňuje zdieľať dáta, uložené v rámci súkromnej databázy s ostatnými popoužívateľmi aplikácie a to podľa vlastného uváženia vlastníka záznamu. používateľ aplikácie môže sprístupniť napríklad jednu alebo viacero záznamov ostatným používateľom s možnosťou čítať alebo ak povolí, tak aj modifikovať záznamy. Pri zdieľaní záznamu sa ďalšiemu používateľovi odošle adresa vo forme *URL*. Táto pozvánka sa viaže na telefónne číslo alebo email. Keď používateľ akceptuje zdieľaný záznam, príslušná aplikácia sa spustí a odovzdá metadáta súvisiace so zdieľaným záznamom, aby bolo možné tento záznam načítať. Pozvánka je akceptovaná len v prípade, že sa užívateľ preukáže ako vlastník čísla. Tento mechanizmus zabezpečuje trieda `UICloudSharingController` popísaná nižšie. zdieľanie prostredníctvom CloudKit-u zahŕňa vytvorenie objektu `CKShare` s konštruktorom obsahujúcim samotný záznam, ktorý sa má zdieľať. Ak chceme aby bola zdieľaná celá hierarchia alebo len vybrané podradené záznamy, treba nastaviť u záznamov premennú `parent` na záznam, ktorý predstavuje rodiča, respektíve nadradený záznam.

Trieda `UICloudSharingController` poskytuje vopred vytvorený pohľad pre manažovanie a zdieľanie záznamov, ktorý vykoná väčšinu práce spojenej so zhromažďovaním potrebných informácií na odoslanie zdieľaného odkazu inému používateľovi. Okrem odoslania odkazu musí byť aplikácia prispôsobená tak, aby akceptovala zdieľanie a získala záznam pre zdieľanú databázu.

zdieľanie musí byť v rámci vlastnej zóny. Vhodnou poznámkou je, že účastníci, ktorým sa zdieľa záznam nemajú prístup do databázy vlastníka. Majú iba právo na pohľad k zdieľaným záznamom.

Operácie

CloudKit spolieha na používanie objektov typu **Operation**, ktoré sprostredkovávajú asynchrónny prenos dát medzi zariadením a serverom. Existuje viacero typov operácii, ale každý môže využívať využívať logické predikáty, ktorými sa upresní dotaz. Ak chceme získať konkrétny objekt, využijeme na to premennú záznamu `recordID`, ktorá slúži ako primárny kľúč, ktorý je unikátny v rámci danej databáze. Operácie môžu mať nastavené obmedzenie na počet dotazovaných záznamov. Záznamy je možné prijímať aj po dávkach. Ak sa preruší internetové spojenie a bola prijatá určitá dávka, je možné použiť objekt `CKQueryCursor`, ktorý bude pokračovať od určitého záznamu.

Prihlasovanie na odber záznamov

Pre aplikáciu je neefektívne sa periodicky dotazovať serveru o nové dáta. Namiesto toho je vhodnejšie sa prihlásiť na odber pre určité typy záznamov. Okrem typu záznamu sa dá špecifikovať v akom momente má byť klient notifikovaný o zmene. Klient si môže vyžiadať prihlásenie na odber pri zmene dát na servery, pri vytvorení alebo zrušení. Taktiež je možné definovať podmienky za akých bude daný záznam odoslaný. Okrem prihlásenia sa na záznam, je možné sa prihlásiť aj na obecnú zmenu databáze alebo používateľskej zóny. Aby však server vedel, o akých zmenách už používateľ vie, ukladá sa po prijatí nových dát `aktualizačný token`, ktorý sa napríklad pri spustení aplikácie a pri vyžiadaní nových zmien zašle.

2.5 Využité knižnice

Aplikácia využíva natívny iOS framework, avšak boli využité aj knižnice, ktoré uľahčujú implementáciu rutinných operácií alebo prácu s určitými komponentami. Knižnice sú stabilné a väčšina z nich je kompletne pokrytá testami a sú vhodné do produkčného prostredia.

- **Realm**[10] – Realm je mobilné riešenie databáze, ktoré je určené a optimalizované primárne na mobilné zariadenia. Dáta sú vystavené prostredníctvom API priamo ako objekt a dotazy sú vytvárané v samotnom kóde, čo odstraňuje potrebu ORM riešenia a zvyšuje výkon a problémy spojené s údržbou. Realm je rýchlejší ako SQLite a podporuje väzby a generiká.

- **GoogleWebRTC**[11] – Mobilná WebRTC knižnica je súčasťou úsilia spoločnosti Google o adaptáciu štandardu WebRTC pre mobilné zariadenia založené na platformách iOS a Android. Knižnica je publikovaná každý týždeň ako snapshot[12] z ich zdrojových súborov [24]. Protokolu WebRTC je venovaná sekcia 3.1 kapitoly 3.

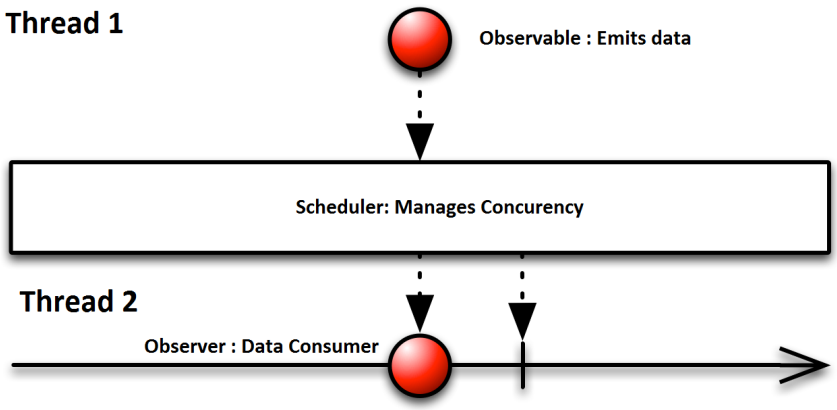
- **RxSwift**[13] **a RxCocoa**[14] – Predstavuje súbor knižníc, ktoré napomáhajú tzv. *reaktívnemu* programovaniu pre platformu iOS a programovací jazyk *Swift*, ktorého zámerom je sprístupnenie asynchrónnych operácií a dátových tokov.

  Reaktívne programovanie je programovacie paradigma orientované na dátové toky a propagáciu udalostí. Príklad komunikácie medzi komponentami je znázornený na obrázku 2.4. *Rx* sa skladá z troch základných komponent [21]:

  - **Observable** – *Observable* je trieda, obaľujúca dáta, ktorá môže byť jednoducho prenášaná z jedného vlákna do vlákna druhého. Táto trieda periodicky emituje dáta v závislosti od jej konfigurácie. Existuje mnoho funkcionálnych operátorov, ktoré zaručia emitovanie dát za určitých okolností alebo vykonajú určitú operáciu. *Observable* slúžia teda ako dodávatelia dát – spracovávajú a dodávajú dáta ostatným komponentom.
  - **Observer** – *Observer* je komponenta, ktorá konzumuje dátový tok emitovaný prostredníctvom *observable*. *Observery* sa prihlasujú na odber emitovaných dát metódou `subscribeOn()`. V momente keď *observable* vyšle dáta, všetky prihlásené *observery* príjmu dáta cez *callback* metódu `onNext()`.
  - **Schedulers** – *Rx* slúži pre asynchrónne programovanie, takže je potrebné uviesť komponentu, ktorá rieši prácu s vláknami. *Scheduler* (alebo plánovač) je komponenta, ktorá určuje na akom vlákne majú bežať *observable* a *observery*.

- **SnapKit**[15] – je nástroj, ktorý uľahčuje prácu s *AutoLayout*om pre systémy iOS a OS X. So *SnapKit*om je sádzanie UI elementov jednoduché a oproti samotnému *AutoLayoutu* je zdrojový kód kratší a čitateľnejší. Umožňuje reťazenie príkazov a je typovo bezpečný, čo značne redukuje výskyt programátorských chýb, ktoré by mali

[10] **Realm** – https://realm.io/
[11] **GoogleWebRTC** – https://cocoapods.org/pods/GoogleWebRTC
[12] **Snapshot** je stav zdrojových súborov a adresárov v repozitároch zachytený v čase
[13] **RxSwift** – https://cocoapods.org/pods/RxSwift
[14] **RxCocoa** – https://cocoapods.org/pods/RxCocoa
[15] **SnapKit** – https://cocoapods.org/pods/SnapKit



Obr. 2.4: *Observable* emituje dáta, ktoré pri emitovaní prejdú *schedulerom* z jedného vlákna do druhého, kde *observer* vo vlákne 2 dáta príjme.

za následok vytvorenie neplatných obmedzení (*„constraintov“*[16].) pri vytváraní používateľského rozhrania. Zdrojový kód je pod flexibilnou licenciou *MIT* a je teda možné ho použiť vo všetkých projektoch bezplatne.

- **ReachabilitySwift**[17] – *Reachability* demonštruje využitie systémového *frameworku* na monitorovanie aktuálneho stavu siete. Predovšetkým demonštruje ako zistiť, či komunikácia prebieha na *WWAN*[18] rozhraniach, ako napríklad *EDGE*, *3G* alebo *LTE*. *Reachability* neposkytuje informáciu, či sa k určitému uzlu môžme pripojiť, len že je dané pripojenie k dispozícii [20].

- **CocoaMQTT**[19] – knižnica zabezpečujúca implementáciu protokolu MQTT pre operačné systémy *OS X* a mobilný systém *iOS*, ktorá je písaná v jazyku Swift. Poskytuje implementáciu MQTT vo verzii *3.1.1*. Samotný protokol MQTT je popísaný v sekcii 3.2.

- **ObjectMapper**[20] – Jazyk *Swift* od verzie 4.0 ponúka sadu nástrojov na dekódovanie a enkódovanie objektov, avšak tieto nástroje vyžadujú vždy konkrétny typ triedy, aby bolo možné objekt dekódovať alebo enkódovať. Vďaka knižnici *ObjectMapper* je možné implementovať generický spôsob enkódovania a dekódovania správ, čo značne zjednoduší implementáciu akejkoľvek sieťovej komunikácie. Knižnica umožňuje mapovanie JSON štruktúri na objekty. Umožňuje taktiež serializovať aj vnorené štruktúry, vďaka čomu je jednoduché konvertovať modelové objekty na JSON a naspäť.

[16] **Constraint** typicky reprezentuje vzťah medzi dvoma používateľskými elementami (vzdialenosť, ukotvenie...)
[17] **ReachabilitySwift** – https://cocoapods.org/pods/ReachabilitySwift
[18] **Wireless Wide Area Network (WWAN)** je typ bezdrôtovej širokopásmovej siete, v ktorej sú oblasti prepojené bezdrôtovo tak, aby pokryli čo najväčšiu geografickú oblasť
[19] **CocoaMQTT** – https://cocoapods.org/pods/CocoaMQTT
[20] **ObjectMapper** – https://cocoapods.org/pods/ObjectMapper

Kapitola 3

Technológie pre prenos audia a videa

Základným stavebným kameňom aplikácie je prenos audia a videa, ktorý vyžaduje využitie komplexných technológií, ktoré spolu tvoria jeden celok. Kapitola preto opisuje technológie potrebné pre implementáciu takéhoto riešenia.

Obsahom je nasledujúca sekcia o technológii *WebRTC*, umožňujúca samotný prenos audio a video dát spolu s veľmi dôležitou bezpečnosťou tejto technológie a prenosu dát. Následne je rozobratá technológia, zabezpečujúca proces *signalizácie* pre WebRTC známa ako *MQTT*, ale aj pre prenos správ, ktoré riadia komponenty aplikácie (riadenie svetla, ovládanie zvuku, prenos nastavení...). Posledná časť je venovaná serverovému riešeniu zabezpečujúcemu párovanie zariadení alebo prípadnú registráciu.

3.1 Prenos mediálnych dát – WebRTC

Web Real-Time Communication (WebRTC) je kolekcia komunikačných protokolov a aplikačných rozhraní (*interfaces*), ktoré umožňujú komunikáciu prostredníctvom *peer-to-peer*[1] sietí v reálnom čase. Spolu poskytujú bohaté mediálne možnosti zahŕňajúce prenos audia, videa, prenosu súborov, zdieľanie pracovnej plochy a ďalšie.

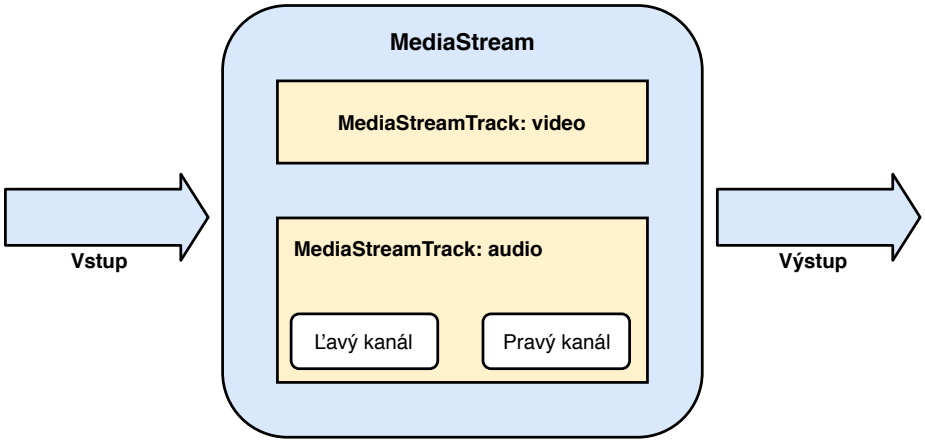
Spojenie medzi dvoma účastníkmi je tvorené a reprezentované rozhraním **RTCPeerConnection** [8, 33]. V momente, kedy je spojenie ustálené a otvorené, sú do spojenia pridávané mediálne streamy[2] (*MediaStreams*) a dátové kanály (*RTCDataChannels*).

Mediálne streamy sa môžu skladať z mnohých stôp (*tracks*), obsahujúcich dátové informácie o médiu. Tieto stopy sú reprezentované objektom, implementujúcim rozhranie *MediaStreamTrack*. Jedna stopa môže obsahovať iba jeden typ mediálnych dát (audio, video, titulky...). Väčšina streamov obsahuje aspoň jednu audio stopu a zvyčajne aj video stopu. Streamy môžu byť využíté na zasielanie živých dát, ale aj dát, ktoré su uložené na disku. Zapúzdrenie dát v mediálnom streame je znázornené na obrázku 3.1.

WebRTC je možné využiť aj na prenos ľubovolných binárnych dát medzi dvoma účastníkmi. Slúži na to rozhranie *RTCDataChannel*. Implementáciou tohto rozhrania je možné docieliť prenosu rôznych súborov, aplikačných metadát alebo ľubovoľného toku dát.

[1] **Peer-to-peer** je označenie typu počítačových sietí, v ktorých spolu komunikujú priamo jednotliví klienti.

[2] **Stream** alebo tok reprezentuje sekvenciu dát



Obr. 3.1: Príklad zapúzdreania stôp v mediálnom streame. *MediaStream* zapúzdruje viacero *MediaStreamTrack*ov (stôp), ktoré môžu obsahovať rôzne typy médií (video, audio).

Na inicializáciu spojenia, výmenu kontrolných informácií medzi zariadeniami sa využíva určitá forma signalizácie (*signaling*). Špecifikácia WebRTC tento proces vôbec nedefinuje. Je to z dôvodu, že medzi zariadeniami zvyčajne neexistuje priame spojenie a špecifikácia nedokáže odhadnúť všetky možné prípady užitia pre takúto aplikáciu. Autori WebRTC teda dávajú developerom možnosť vybrať vhodnú sieťovú technológiu a vhodný protokol pre prenos správ. Na druhú stranu, ak už určitá aplikácia využíva nejakú formu komunikácie medzi dvoma zariadeniami, je zbytočné uvádzať nový komunikačný kanál len pre potreby WebRTC.

Čo by mal proces signalizácie riešiť:

- Kontrolné správy, využívané na inicializáciu, otvorenie a ukončenie komunikačného kanálu, spolu s riešením neočakávaných chýb.
- Prenos informácií potrebných pre ustálenie spojenia: IP adresy a informácie o portoch, potrebné pre komunikáciu medzi dvoma účastníkmi.
- Sprostredkovanie informácií o prenášaných mediálnych dátach: akým kodekom a dátovým formátom účastníci rozumejú; tieto informácie je potrebné preniesť pred začiatkom každého sedenia (*session*).

Až sú všetky tieto náležitosti splnené, môže sa začať proces otvárania WebRTC spojenia. Za zmienku stojí aj to, že signalizačný server nepotrebuje rozumieť dátam alebo s nimi manipulovať, slúži teda len ako *relay*[3] server.

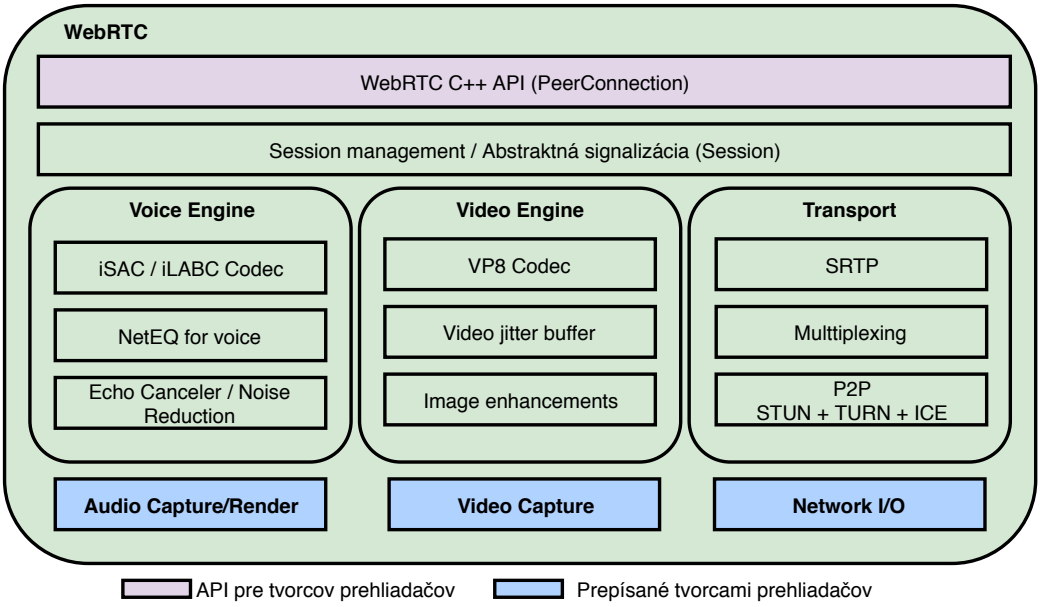
3.1.1 Architektúra

Architektúra WebRTC [3] sa skladá z viacerých komponentov, ktoré odtieňujú developerov od komplexných úkonov a spolu s kodekmi[4] a ostatnými protokolmi umožňujú komunikáciu v reálnom čase aj pri sieťach, ktoré nie sú úplne spolahlivé. Architektúra je znázornená na obrázku 3.2 a zastrešuje napríklad nasledujúce mechanizmy:

[3] **Relay server** je server, ktorý len preposiela prijaté dáta.

[4] **Kodek** je zariadenie alebo SW komponenta zabezpečujúca kódovanie alebo dekódovanie dátového toku alebo signálu.

- Vysporiadanie sa so stratami packetov.
- Filtrácia ozveny (*echo*).
- Adaptivita šírky pásma.
- Dynamické vyrovnanie *jitteru*[5].
- Automatické vyrovnanie citlivosti[6].
- Zníženie a potlačenie šumu.
- „Čistenie“ obrazu.



Obr. 3.2: Znázornenie architektúry WebRTC. Architektúra sa skladá z viacero fundamentálnych komponent, ktoré sú popísané v sekcii 3.1.1. Architekrúra obsahuje implementované tzv. *engines* a verejne dostupné rozhrania, ktoré sú dostupné pre developera.

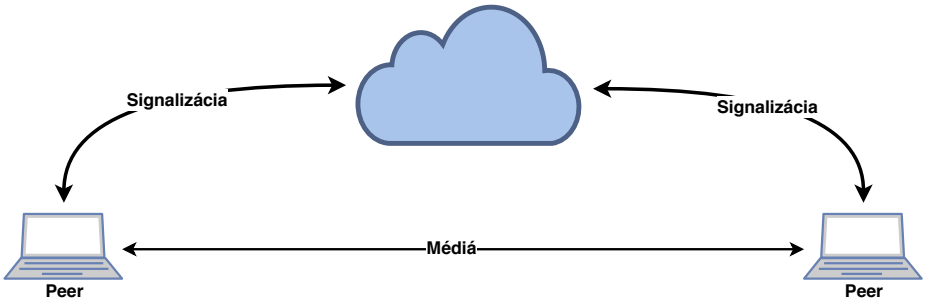
3.1.2 ICE framework

Signálizácia vyžaduje využitie externého serveru pre výmenu metadát, no po dokončení výmeny týchto metadát sa WebRTC pokúša vytvoriť priame *peer-to-peer* spojenie medzi účastníkmi. Tento proces sa uskutočňuje prostredníctvom *frameworku ICE*[7].

ICE (Interactive Connectivity Establishment) je framework používaný na vytvorenie spojenia medzi účastníkmi na internete. Aj keď sa WebRTC snaží využívať priame *peer-*

[5] **Jitter** je nežiaduca odchýlka, vyjadrujúca kolísanie veľkosti oneskorenia packetov pri priechode sieťou.
[6] **Automatic Gain Control (AGC)** - automatické vyrovnanie citlivosti je systém používaný v mnohých elektronických zariadeniach; bez AGC by boli príjmače veľmi citlivé na zmeny veľkosti príjmaného signálu.
[7] **Framework** (aplikačný rámec) je softwarová štruktúra, slúžiaca ako podpora pri programovaní, ktorá obsahuje podporné programy a knižnice.

to-peer spojenie, v skutočnosti robí rozšírená prítomnosť *NATu*[8] problém pri dohadovaní komunikácie.



Obr. 3.3: Predstava teoretickej komunikácie *WebRTC* pri ideálnej sieti, ktorá neobsahuje *NAT* zariadenia a komunikácia je teda možná napriamo bez použitia *STUN* alebo *TURN* serverov. Príkladom môže byť interná domáca sieť a zariadenia komunikujúce v nej.

Vďaka pretrvávajúcemu rozšíreniu IPv4 adries s obmedzenou 32–bitovou reprezentáciou, väčšina zariadení pripojená dnes na internet nevlastní unikátnu *IPv4* adresu, ktorá by bola viditelná na internete. Je to z dôvodu dochádzajúcich *IPv4* adries[9]. Odpoveďou mal byť protokol *IPv6*[10], ktorý je podľa štatistík Googlu na úrovni iba 21.97% globálne a 11.18% v Českej republike[11]. Preto sa prišlo na spôsob, ako namapovať viacero zariadení na jednu *globálnu IPv4* adresu, využitím štandardu *NAT*.

Zariadenia zvyčajne nevlastnia unikátnu adresu a taktiež sa adresa mení nielen pri prechode medzi sieťami, ale aj kvôli *NAT*u alebo napríklad *DHCP*[12] serveru. Developerom, ktorí sa snažia o implementáciu *peer-to-peer* komunikácie, to prináša veľký problém: bez unikátnej identifikácie neexistuje jednoduchý spôsob ako sa napriamo pripojiť ku konkrétnemu zariadeniu. Ak aj poznáme účastníka, ku ktorému sa chceme pripojiť, nie je jednoduché zistiť ako toho dosiahnuť a akú má cieľové zariadenie adresu.

Skúšaním všetkých možných ciest paralelne je *ICE* schopné nájsť najefektívnejšiu možnosť, ktorá úspešne spojí dvoch účastníkov. Služba *ICE* sa najskôr pokúsi vytvoriť spojenie za použitia hostiteľskej adresy získanej z operačného systému a sieťovej karty zariadenia. Ak to zlyhá (čo je veľmi pravdepodobné pre zariadenia za *NAT-om*), *ICE* sa následne pokúsi získať svoju externú IP adresu za použitia tzv. *STUN* serveru. Ak zlyhá aj táto metóda, prevádzka sa presmeruje cez tzv. *TURN relay* server.

Trasy, ktoré sú vhodné na prenos sa prenášajú v textovom formáte a uzly sú zoradené v zozname podľa priority. Možnosti sú teda následovné:

- priama *peer-to-peer* komunikácia za použitia *STUN* serveru s mapovaním portu pre priechod cez *NAT* (táto trasa nakoniec vedie k priamej *P2P* komunikácii)
- nepriama komunikácia za použitia *TURN* serveru ako sprostredkovatela

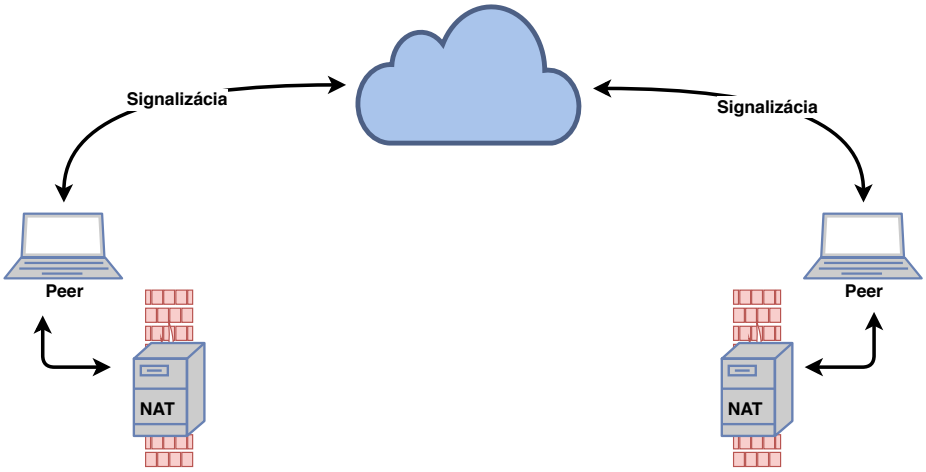
[8] **Network Address Translation (NAT)** pracuje na princípe dynamického prekladania privátnych adries na verejnú - pri odchádzajúcich požiadavkach. Podobne aj s prichádzajúcimi požiadavkami - prekladá verejnú IP adresu naspäť na privátnu, aby sa zabezpečilo správne smerovanie v rámci internej siete.

[9] Adresný priestor *IPv4* tvorí 2^{32} adries (4 miliardy adries)

[10] Adresný priestor *IPv6* [2] poskytuje až 2^{128} adries

[11] Údaj pochádza z dátumu *8.1.2018* - https://www.google.com/intl/en/ipv6/statistics.html

[12] **Dynamic Host Configuration Protocol (DHCP)** protokol, ktorý sa používa pre automatickú konfiguráciu zariadení pripojených do siete.



Obr. 3.4: Znázornenie reálnej situácie využitia *WebRTC* protokolu v prítomnosti *NAT*u. Pre nadviazanie spojenia je v takomto prípade nutná prítomnosť *STUN* serveru pre zistenie IP adries účastníkov.

3.1.3 STUN servery

Aby mohla prebiehať *P2P* komunikácia, obe strany nevyhnutne vyžadujú minimálne znalosť IP adries svojho partnera a priradeného portu *UDP*[13]. V dôsledku toho je potrebná určitá miera výmeny informácií predtým, ako je možné zahájiť komunikáciu WebRTC.

STUN (Session Traversal Utilities for NAT) server je používaný oboma účastníkmi na zistenie ich verejnej IP adresy a je pri vytváraní spojenia odkazovaný na rámec *ICE* [13]. *STUN* servery sú zvyčajne verejne prístupné a môžu byť voľne používané WebRTC aplikáciami.

3.1.4 TURN servery

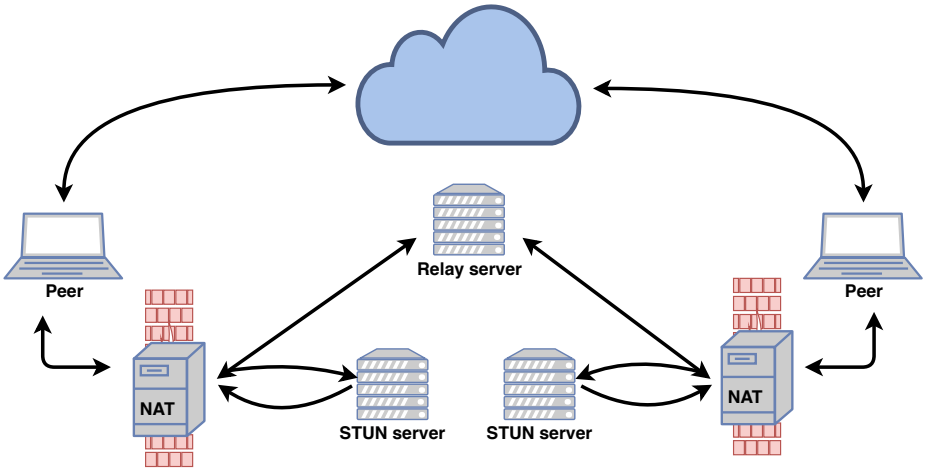
V prípade, že zlyhá vytvorenie *P2P* komunikácie je náhradnou možnosťou využitie tzv. *TURN (Traversal Using Relays around NAT)* serveru. Presmerovaním prevádzky medzi účastníkmi je možné tento problém vyriešiť, avšak môže prísť k zhoršeniu kvality a latencie médií. Podľa Googlu k tejto možnosti však štatisticky prichádza iba pri 14% komunikácie [55], viď obrázok 3.6.

TURN servery dokážu úspešne zabezpečiť naviazanie spojenia bez ohľadu na prostredie koncového používateľa. Ako sú dáta posielanie skrz server, využíva sa taktiež šírka pásma daného serveru. Ak sú viaceré hovory súčasne smerované cez server, záťaž pripojenia serveru je taktiež vyššia. Samotný server zvyčajne nie je voľne dostupný a musí ho poskytovateľ aplikácie prenajímať alebo vlastniť, preto býva zvyčajne táto možnosť v aplikáciách spoplatnená. *TURN* server môže fungovať aj ako *STUN* server.

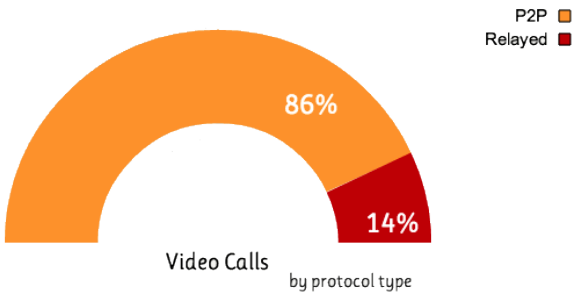
3.1.5 Naviazanie spojenia

Ako bolo už spomenuté, WebRTC nedokáže nadviazať spojenie bez použitia externej formy signalizácie (signalizačný kanál). Môže to byť akýkoľvek komunikačný kanál, ktorý dokáže prenášať informácie pred samotným zostavením WebRTC spojenia [4]. Príkladom môže byť

[13] **User datagram protocol (UDP)** je protokol transportnej vrsty orientovaný na správy, ktorý neposkytuje žiadne záruky doručenia.



Obr. 3.5: Diagram znázorňujúci hľadanie kandidátov *ICE*. V prítomnosti *NAT*u sa využívajú servery *STUN*. Ak tento spôsob zlyhá, využije sa presmerovanie video a audio komunikácie cez servery *TURN (Relay)*.



Obr. 3.6: Percentuálne zastúpenie mediálneho prenosu prostredníctvom **TURN** serveru. Informácia pochádza z [55]. Z grafu je zrejmé, že väčšina komunikácie prebieha prostredníctvom priameho *peer-to-peer* spojenia.

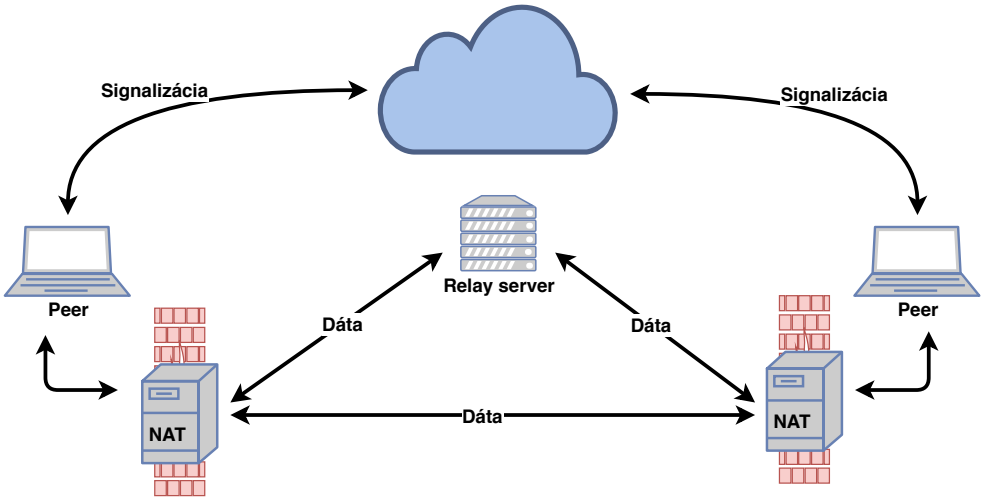
protokol *WebSocket*[14], *SIP*[15], *MQTT* (viď. sekciu 3.2) a mnohé ďalšie. Informácie o *SDP*, ktoré sú potrebné preniesť sú obsiahnuté v správach *Offer (ponuka k spojeniu)* a *Answer (odpoveď)*.

SDP (Session Description Protocol) je protokol určený k popisu vlastností relácie multimediálneho prenosu dát. Neprenášajú sa pomocou neho vlastné dáta, slúži však na vyjednanie parametrov prenosu, akými sú napríklad typ média, transportný protokol, typ kodeku a ostatné parametre popísané v RFC [9].

Ak účastník vytvára WebRTC spojenie s druhým účastníkom, vyvtorí sa špeciálna správa *Offer* [12]. Táto správa zahŕňa informácie o konfigurácii volajúceho. Príjemca následne odpovedá správou *Anwser* obsahujúcou taktiež svoju konfiguráciu. Každý účastník si uchováva dve konfigurácie:

[14]**WebSocket** je komunikačný protokol poskytujúci plne duplexný komunikačný kanál prostredníctvom jedného *TCP* spojenia.

[15]**Session Initiation Protocol (SIP)** je signalizačný protokol určený na vytváranie a ukončovanie multimediálnych spojení.



Obr. 3.7: Diagram znázorňujúci prenos WebRTC komunikácie a rôznych dátových tokov v prítomnosti *NAT*u. Pri tzv. *čiastočnom ICE reštarte* popisovanom v sekcii 3.1.5 môžu mediálne toky prúdiť rôznymi cestami. V tomto prípade jeden mediálny tok preteká cez *TURN* server a ďalší pokračuje v prenose napriamo.

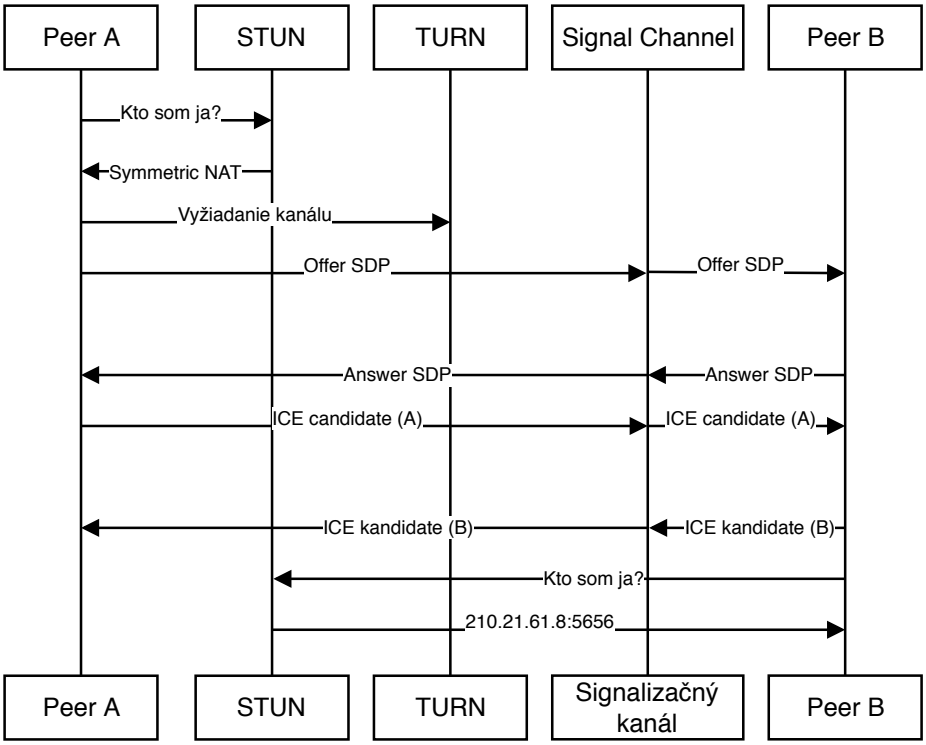
- *Local description* popisujúcu konfiguráciu o sebe samom.
- *Remote description* popisujúcu konfiguráciu účastníka na druhej strane.

Proces výmeny *Offer/Answer* správ sa vykonáva vždy na začiatku spojenia, ale taktiež aj v momente, kedy je treba zmeniť formát alebo konfiguráciu. Či už ide o vytvorenie nového spojenia alebo rekonfiguráciu, následné kroky sú spoločné pre obe operácie:

1. Volajúci účastník zavolá metódu `RTCPeerConnection.createOffer()` pre vytvorenie *Offer* správy.
2. Volajúci účastník zavolá `RTCPeerConnection.setLocalDescription()` pre nastavenie vlastnej konfigurácie (*local description*).
3. Volajúci využije signalizačný server pre výmenu správy *Offer* s druhým účastníkom.
4. Príjemca príjme *Offer* správu a zavolá `RTCPeerConnection.setRemoteDescription()` (*local description* volajúceho).
5. Príjemca vykoná potrebné nastavenie (napríklad pridanie audio/video *MediaStreamTrack*ov do streamu).
6. Príjemca vytvorí odpoveď zavolaním metody `RTCPeerConnection.createAnswer()`.
7. Príjemca zavolá metódu `RTCPeerConnection.setLocalDescription()` aby naplnil obsah správy svojou *local description*.
8. Príjemca využije signalizačný server na zaslenie odpovede volajúcemu.
9. Volajúci obdrží odpoveď.
10. Volajúci zavolá metódu `RTCPeerConnection.setRemoteDescription()` aby si uložil obsah správy ako *remote description*.

11. Obaja účastníci majú potrebné informácie na začatie hovoru.

Premenné *local description* a *remote description* môžu byť počas konfigurácie (aj opätovnej) zamietnuté. Môže to nastať kvôli nekompatibilite formátov zariadení. Je preto dôležité, aby si dokázal každý účastník navrhnúť nový formát, bez toho aby ho okamžite začal používať, pokiaľ ho druhá strana neschváli. Z toho dôvodu si WebRTC uchováva aktuálnu aj dočasnú konfigurácia a potom, čo sa obaja účastníci zhodnú na formáte, dočasná konfigurácia sa nastaví ako aktuálna.



Obr. 3.8: Diagram znázorňujúci naviazanie WebRTC spojenia (offer dance). V komunikácii účinkujú dvaja účastníci, *TURN* a *STUN* server, spolu so signalizačným kanálom. Komunikácia je detailnejšie popísaná v sekcii 3.1.5.

Kandidáti ICE

Okrem výmeny mediálnych informácií si musia účastníci vymeniť aj informácie o sieti. Účastníci hľadajú tzv. *ICE* kandidátov. Tento proces označuje hľadanie sieťových rozhraní a portov. Typicky navrhne každý účastník svojich najlepších kandidátov. Kvôli rýchlosti sú preferovaný UDP kandidáti, avšak *ICE* štandard akceptuje aj TCP kandidátov. Tí sú však používaný len v prípade, že UDP nie je k dispozícii. Kompetný diagram naviazania spojenia spolu s výmenou *ICE* kandidátov je znázornený na diagrame 3.8.

Reštart ICE

Ak sa počas dátového prenosu zmenia sieťové podmienky, či už ide o prechod z WiFi siete do mobilnej siete alebo dôjde k preťaženiu siete, *ICE* framework sa môže rozhodnúť prekonfigurovať nastavenia a vykonať tzv. *ICE reštart*. Je to proces, v ktorom sa opätovne

vyjednáva konfigurácia spojenia a je takmer totožný s prechádzajúcim popisom zahajovania spojenia. Jediným rozdielom je, že dáta naďalej pretekajú pôvodnou cestou, dokiaľ sa neustáli nové spojenie. V tom momente začnú pretekať dáta novým spojením a staré sa uzatvorí. Existujú však dva spôsoby reštartu:

- **Úplný reštart** - Pri tejto metóde je nutné zahájiť konfiguráciu všetkých mediálnych tokov odznova; pre zahájenie stačí znovu začať konfiguračný proces zavolaním metódy `RTCPeerConnection.createOffer()`.
- **Čiastočný reštart** - Konfigurácia sa vykoná len pre vybrané mediálne toky (napríklad iba dátový tok, bez konfigurácie mediálneho toku); vykoná sa zavolaním metódy `RTCPeerConnection.setConfiguration()` obsahujúcej upravenú konfiguráciu.

3.1.6 Bezpečnosť komunikácie

Existuje množstvo spôsobov, ako môže byť aplikácia komunikujúca v reálnom čase vystavená bezpečnostným rizikám. Príkladom je zachytenie nešifrovaného mediálneho alebo dátového toku počas samotného prenosu. Táto situácia môže nastať pri komunikácii medzi dvoma účastníkmi alebo medzi serverom a účastníkom, pričom útočník môže získavať prenášané dáta. Šifrovaním sa však zabezpečí, aby iba strany vlastniace šifrovací kľúč, mohli komunikáciu dešifrovať.

Šifrovanie je povinnou funkciou WebRTC a musia mu podliehať všetky komponenty vrátane signalizácie [10]. Z toho dôvodu sú všetky mediálne toky šifrované štandardizovanými šifrovacími protokolmi. Použitý šifrovací protokol závisí od typu kanála:

- Dátové toky sú šifrované protokolom *DTLS*.
- Mediálne toky sú zabezpečené pomocou *SRTP*.

DTLS – Datagram Transport Layer Security

WebRTC šifruje dátové kanály pomocou technológie *DTLS*. Všetky dáta odoslané prostredníctvom *RTCDataChannel* sú zabezpečené pomocou protokolu *DTLS*.

DTLS je štandardizovaný protokol, ktorý je integrovaný vo všetkých prehliadačoch a knižniciach implementujúcich WebRTC. Je vhodným protokolom, ktorý sa používa v prehliadačoch, emailových klientoch, pri *VoIP*[16] platformách a ďalších, pre šifrovanie prenášaných dát. Kvôli rozšírenosti protokolu nie je potrebná žiadna inicializačná fáza. *DTLS* je stavaný na protokole *TLS*, ktorý poskytuje úplné šifrovanie pomocou asymetrickej kryptografie, autentifikácie dát a autentifikácie správ. *TLS* je štandard pre webové šifrovanie používaný pri protokoloch ako napríklad *HTTPS*[17]. *TLS* je však navrhnutý pre spoľahlivý mechanizmus prenosu (*TCP*), avšak nie je uspôsobený pre potreby *VoIP* aplikácií, ktoré bežne využívajú nespoľahlivý transportný mechanizmus (*UDP*).

Keďže *DTLS* je deriváciou protokolu SSL[18], všetky dáta sú rovnako dobre zabezpečené ako pri *SSL* pripojení [18]. V skutočnosti môžu byť WebRTC dáta zabezpečené pomocou

[16] **VoIP** (Voice over internet protocol) je prenos hlasovej prevádzky prostredníctvom prenosu paketov pomocou protokolu *IP*.

[17] **HTTPS** je zapezpečená verzia *HTTP* protokolu.

[18] **Secure Sockets Layer (SSL)** predstavuje protokol, resp. sieťovú vrstvu umiestnenú medzi transportnou a aplikačnou vrstvou poskytujúca zabezpečenie komunikácie šifrovaním a autentizáciou komunikujúcich strán.

ľubovolného pripojenia založenom na protokole *SSL*, čo umožňuje WebRTC ponúkať *end-to-end* šifrovanie medzi účastníkmi a to pri akomkoľvek usporiadaní serverov.

SRTP – Secure Real-time Transport Protocol

Základný *RTP* protokol neobsahuje bezpečnostné mechanizmy a preto neposkytuje žiadnu ochranu prenášaných údajov. Kvôli tomu sa pri *RTP* vyžaduje externá forma šifrovania. V skutočnosti je však využívanie nešifrovaného *RTP* pri WebRTC vyslovene zakázané špecifikáciou.

WebRTC používa *SRTP* na šifrovanie mediálnych tokov namiesto *DTLS*. Dôvodom je to, že *SRTP* je menej náročná oproti *DTLS*. Preto špecifikácia WebRTC prikazuje implementáciu *RTP–SAVPF* [11]. Počiatočná výmena kľúčov sa však vykonáva *end–to–end* pomocou *DTLS–SRTP*, čo umožňuje detekovať prípadné MiTM[19] útoky.

3.2 Signalizácia pomocou MQTT

Message Queing Telemetry Transport známy ako MQTT, je transportný protokol predstavený v roku 1999 firmou IBM. Bol štandardizovaný pod ISO štandardom **ISO/IEC PRF 20922** a v roku 2014 bol oficiálne schválený ako OASIS[20] štandard [7]. Jedná sa o nenáročný protokol využívajúci model *publish-subscribe* a je postavený nad protokolom *TCP/IP* [6].

Protokol bol navrhnutý pre nenáročné zariadenia, ktoré disponujú obmedzenými hardwarovými zdrojmi (8–bit controller s 256KB RAM), ako napríklad zariadenia s nízkym výkonom, drahým internetovým pripojením alebo vysokou latenciou. Príkladom sú senzory alebo inteligentné zariadenia pre internet vecí (*IoT*). Je zároveň ideálnym protokolom pre mobilné aplikácie a to kvôli jeho velkosti, malým dátovým packetom a efektívnej distribúcii informácií jednému alebo viacerým odberateľom (*subscribers*).

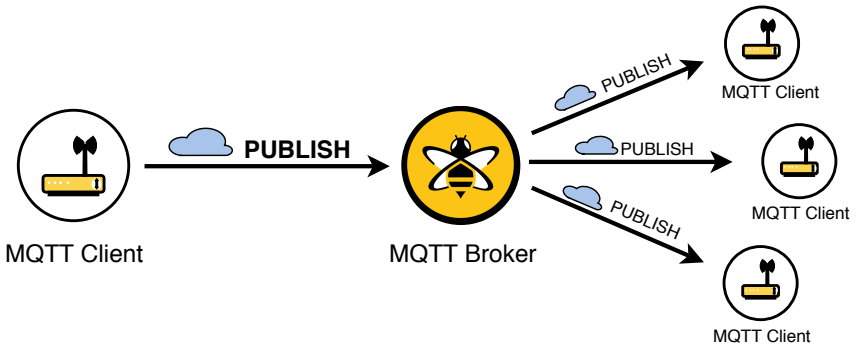
Zariadenie, ktoré odosiela správy sa nazýva *publisher*. Tieto správy sú zasielané do centrálneho bodu, nazývaného *broker*. Správy sú triedené podľa tém (*topics*), na ktoré sú prihlásený odberatelia (*subscribers*). Jedno zariadenie môže vystupovať pod oboma rolami – odosielať správy do daných tém a taktiež správy z rôznych tém prijímať.

3.2.1 Komunikácia a prenášané dáta

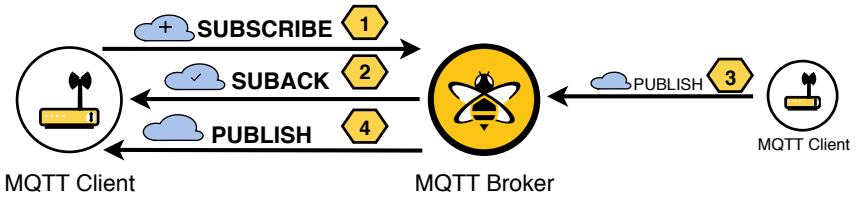
Pre vytvorenie spojenia medzi klientom a *brokerom* slúži správa *CONNECT*. *Broker* odpovie správou *CONNACK* a klient sa následne môže prihlásiť k jednotlyvím odberom. Na príklade 3.9 je znázornené zasielanie (*publishing*) správ prihláseným klientom. *Publisher* nemusí poznať klientov, ktorým zasiela správu – prihlásený klienti ju obdržia od *brokera*. Prihlasovanie (*subscribing*) je znázornené na obrázku 3.10. Klient, ktorý chce dostávať správy, zašle správu typu *SUBSCRIBE* (poprípade *UNSUBSCRIBE*) *brokeru* a ten mu odpovie správou *SUBACK*. Všetky správy, ktoré klienti odošlú na prihlásený *topic* budú doručené *subscriberovi*.
Príklad zasielaného packetu je znázornený v tabuľke 3.1. Popis štruktúry packetu:

[19] **Man in The Middle (MiTM)** je útok, ktorého podstatou je snaha útočníka odpočúvať komunikáciu medzi účastníkmi, tak, že sa stane aktívnym prvkom, ktorý preposiela komunikíciu medzi účastníkmi.

[20] **Organization for the Advancement of Structured Information Standards (OASIS)** - Organizácia na presadzovanie noriem pre štruktúrované informácie - je neziskové, globálne konzorcium, ktoré je hnacou silou vývoja, spájania a adaptovania noriem elektronického obchodu a webových služieb.



Obr. 3.9: Príklad zasielania správ. Jeden z *MQTT* klientov odošle určitú správu *brokerovi*. *Broker* ju rozpošle prihláseným klientom.



Obr. 3.10: Príklad prihlasovania k *topicom*. Klient zašle brokerovi správu *SUBSCRIBE*, broker mu následne odpovie správou *SUBACK*, ktorá potvrdzuje prihlásenie. Ak nejaký klient odošle správu na daný *topic*, prihlásený klient správu dostane.

- **PacketId** predstavuje identifikátor správy, ktorý je unikátny medzi klientom a *brokerom*. Nastavenie tohto identifikátora je záležitosť klienta alebo *brokera*.
- **Topic** znázorňuje hierarchickú štruktúru, obsahujúcu predné lomítko ako oddelovač cesty. Cesta môže obsahovať špeciálne znaky ako # alebo +. Znak mriežky nahradzuje jednu alebo viacero úrovní a musí byť uvedená vždy ako posledná - prihlásením na „dom/obyvacka/#“ prijímame všetky témy z obývacej izby. Znak *plus* nahradzuje jednu úroveň v hierarchii - prihlásením sa na „dom/+/teplota“ prijímame správy o všetkých teplotách v domácnosti.
- **Quality of Service (QoS)** je úroveň spolahlivosti doručenia správ. Ta sa delí na nasledujúce tri úrovne:
  - **QoS 0** (najviac 1x) - Prijatá správa nie je potvrdzovaná príjemcom a *broker* ju ani neukladá. Táto úroveň je vhodná pri stabilnom internetovom pripojení.
  - **QoS 1** (aspoň 1x) - Táto úroveň ručí za doručenie správy najmenej jeden krát. Príjemca musí odpovedať potvrdzovacím *PUBBACK* packetom, inak sa po určitom timeoute zasiela packet znovu.
  - **QoS 2** (práve 1x) - Najvyššia úroveň zaisťuje doručenie správy práve jeden krát. Nie je teda akceptovaná strata a ani duplicita správy. *Publisher* odošle správu *PUBLISH brokerovi* a ten ju prepošle *subscriberom* a odpovie správou *PUBREC*. *Publisher* odpovie správou *PUBREL* a *broker* správu smaže. Následne ukončí komunikáciu správou *PUBCOMP*. Používa pri kritických operáciách.

packetId	topicName	qos	retainFlag	payload	dupFlag
4314	"dom/obyvacka/teplota"	1	false	"teplota:32.5"	false

Tabuľka 3.1: Príklad zasielaného MQTT packet.u

- **Retain flag** rozhoduje o tom, či bude správa uložená u brokera ako posledná známa. Ak sa nový klient prihlási na daný *topic*, táto posledná známa správa mu bude automaticky doručená.
- **Payload** je obsah danej správy. MQTT je protokol, ktorý je *data–agnostic*, čo znamená, že obsahom môže byť čokolvek od textov, až cez obrázky a binárne dáta. Veľkosť dát je závislá na *brokerovi*.
- **Duplicate flag** indikuje, že sa jedná o správu, ktorá bola opakovane zaslaná, pretože niektorí klienti nezaslali potvrdzovací packet, ktorý je relevantný len pri QoS väčšom ako 0.

3.2.2 Bezpečnosť

Pre zaručenie bezpečnosti tohto protokolu sa odporúča použitie šífrovania správ pomocou *TLS/SSL*[21]. Zároveň sa pri pripojovaní klienta k *brokerovi* používa autorizácia pomocou bežného používateľského mena a hesla [19], jedná sa však o záležitosť implementácie konkrétneho *brokeru*.

[21] **Transport Layer Security** (TLS) a jeho predchodca **Secure Sockets Layer** (SSL) sú protokoly, ktoré slúžia na šifrovanie dát.

Kapitola 4

Detekcia pohybu a analýza algoritmov

Počet dostupných kamier dostupných po celom svete za posledné desťročie dramaticky vzrástol. Tento rast však viedol k obrovskému nárastu dát, ktoré už nie je kam ukladať a taktiež aj manuálne spracovanie už nie je realizovateľné. Z toho dôvodu patrí posledné roky snaha o získanie informácií o ľudských aktivitách z videa medzi dôležité prúdy oboru počítačového videnia. Rastúci záujem o analýzu pohybu je silne motivovaný aj zlepšeniami v oblasti hardwaru, či už sa jedná o lacné a výkonné mobilné zariadenia alebo kamery. Táto kapitola zhrňuje základné informácie o detekčných algoritmoch a ich analýze. Taktiež rozoberá algoritmy potrebné pre implementáciu detekčných algoritmov. Záver obsahuje zhrnutie použitých technológii pre implementáciu riešenia.

4.1 Detekčné algoritmy

Mechanizmus detekcie pohybu začína určením referenčného snímku. Následný snímok sa potom porovná so snímkom referenčným. Proces zachytenia snímku sa vykonáva v pravidelných intervaloch v súlade s požiadavkami daného systému. Podľa štúdie, vypracovanej Mishrom et al. [38], existujú tri bežne používané metódy pre zachytenie pohybu:

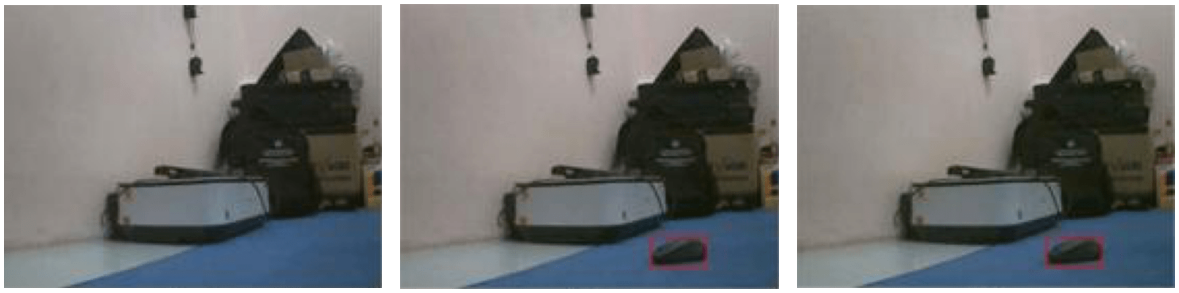
- **Background subtraction** – Metóda sa vykonáva porovnaním snímok so snímkom referenčným. Táto technika používa statický alebo dynamický referenčný snímok.
- **Optical flow** *(optický tok)* – Štúdia [39] používala na detekciu pohybu metódu *optical flow*. Táto metóda však vyžaduje okrem vysokého výkonu zariadenia aj podporu špeciálneho hardwaru, čo v prípade využitia algoritmu v mobilných zariadeniach nie je aktuálne možné, preto sa jej v tejto práci nebudeme ďalej venovať.
- **Frame differencing** – Metóda *frame differencing* je známa aj pod názvom *temporal differencing*. Táto metóda vykonáva porovnanie po sebe zachytených snímok.

4.1.1 Background subtraction

Background subtraction (BGS) je známa aj ako metóda detekcie popredia. Táto metóda je veľmi rozšírená pri detekcii obrazu primárne zo statických kamier. Princíp detekcie pohybujúcich objektov spočíva vo vykonaní rozdielu medzi aktuálnym snímkom a snímkom referenčným, ktorý je nazývaný aj ako *background image* alebo *background model*.

Popredie sa ďalej extrahuje pre nasledovné spracovanie. Po predspracovaní snímkov sa vykonáva lokalizácia objektov, ktorá využíva túto metódu. Oblasti záujmu predstavujú zvyčajne objekty ako ľudia, dopravné prostriedky, text a mnohé ďalšie, nachádzajúce sa v popredí. *Background subtraction* poskytuje uplnatnenie v mnohých aplikáciách počítačového videnia, ako napríklad v dohľadových systémoch alebo pri odhadovaní ľudských póz [54].

Backgroung subtraction je vo všeobecnosti založená na hypotéze statického pozadia, ktorá často nie je použiteľná v reálných podmienkach (znázorňuje obrázok 4.1). Pri scénách, odohrávajúcich sa v interiéri, často dochádza k odrazom alebo zmenám (šetrič obrazovky na počítači), ktoré spôsobujú zmenu *background modelu*. Podobne to je aj pri vonkajších scénach, kedy dochádza k zmenám osvetlenia alebo zmenám vpyvom dažďu, vetra (vlnenie vody, kymácajúce sa stromy) a iným vplyvom, ktoré je ovplyvnené počasím [44]. Algoritmus detekcie je triviálny v prípade, že je k dispozícii čisté nemenné pozadie.



(a) (b) (c)

Obr. 4.1: Jednoduchá metóda využívajúca *Background subtraction* so statickým pozadím ako referenčný snímok. Pohybujúci objekt bol detekovaný správne, no kvôli tomu, že metóda nie je adaptívna bol detekovaný objekt aj naďalej označovaný ako pohybujúci. Zdroj [57].

Metóda extrakcie pozadia počas inicializačnej a adaptačnej sekvencie sa nazýva *background modeling*. Hlavnou výzvou pri detekcii obrazu je práve extrakcia čistého pozadia. Existujú rôzne metódy pre *background modeling*, ako napríklad *Running average* [48, 59, 42] alebo *Running Gaussian average* [58, 47]. Tieto parametrické techniky určujú a aktualizujú popredie na základe roloženia hodnoty intenzity pixelu zo snímku [43]. Niektoré z týchto metód, ako napríklad *median filter* [30, 34], majú veľkú pamäťovú náročnosť a iné ako *Eigen-background* [41] alebo *Mixture of Gaussian (MOG)* [52] zase vysokú výpočtovú zložitosť.

Okrem parametrických metód existujú aj bezparametrické metódy (*samples-based*), ktoré rozpoznávajú pozadie a popredie podľa intenzity štatistických vlastností [37]. Táto technika vytvára svoj *model* agregáciou historicky pozorovaných hodnôt pre pixel. Z ostatných bezparametrických metód stojí za zmienku *Kernel density estimation* [39] alebo *Mean shift estimation* [45]. Všetky spomínané metódy majú volnejšiu licenciu, no jedna z aktuálne najznámejších metód *ViBE* podlieha patentu. Algoritmu *ViBe* bude venovaná sekcia **??**.

4.1.2 Frame differencing

Metóda *Temporal difference* je jednou z najznámejších techník *BGS*. Spočíva vo vypočítaní rozdielu medzi dvoma po sebe idúcimi snímkami [51]. Po odčítaní snímkov sú hodnoty pixelov, ktoré neprekročia prah (*treshold*) vynulované. Týmto postupom môžme získať výsledné pohybujúce sa objekty. Metóda má nízku výpočtovú zložitosť, no zároveň je príliš citlivá

na hodnotu *tresholdu*. Výsledkom príliš nízkeho *tresholdu* je v detekovanom výsledku šum. Ak je naopak hodnota *tresholdu* príliš veľká, môže dôjsť k skresleniu a strate dôležitých informácií.

Metódú však ovplyvňuje aj rýchlosť pohybujúceho sa objektu. Ak sa objekt pohybuje príliš rýchlo, použitie metódy *Temporal difference* bude viesť k tomu, že jeden objekt bude detekovaný ako dva, kvôli vzniku tzv. „dier".

$$F_i(x,y) = \begin{cases} 1, & |F_i(x,y) - B_i(x,y)| > A \\ 0, & \text{inak} \end{cases} \tag{4.1}$$

Pre rozpoznanie pohybu sa absolútny rozdiel medzi snímkami porovnáva s príslušným *tresholdom* A, ako je ukázané v rovnici (4.1). F_i značí aktuálnu hodnoty intenzity pixelu, B_i je hodnota intenzity pixelu na pozadí, Fg_i je hodnota intenzity popredia. Táto technika používa rovnaký statický *background model* pre všetky snímky vo video sekvencii.

4.1.3 Approximate median filter

Approximate mediam filter (AM) je adaptívna a dynamická metóda [50] v ktorej sa počíta rozdiel medzi dvoma snímkami. *AM* je považovaná za jednu z najakceptovanejších metód, pretože poskytuje najpresnejšiu identifikáciu pixelov pozadia.

Existujú viaceré štúdie, v ktorých sa vyhodnocovala účinnosť algoritmu *AM*. Napríklad autori Liu et al. testovali efktívnosť *AM* ako súčasť ich optimalizačného algoritmu pre detekciu vozidiel pre vstavané systémy [34].

$$B_{i+1}(x,y) = \begin{cases} F_i(x,y) + 1, & F_i(x,y) > B_i(x,y) \\ F_i(x,y) - 1, & \text{inak} \end{cases} \tag{4.2}$$

Rovnica (4.2) predstavuje aktualizáciu intenzity hodnôt pixelu pre každý ďalší nasledujúci snímok vo video sekvencii. Premenná B_{i+1} značí výslednú hodnotu intenzity ďalšieho referenčného snímku a je závislá na hodnote intenzity aktuálneho snímku (F_i) a súčasného referenčného snímku (B_i).

4.1.4 Running average

Background model sa pri metóde *Running average* prispôsobuje meniacej sa scéne použitím váženej sumy (*weighted sum*) aktuálneho snímku a aktuálneho snímku pozadia a je vyjadrená rovnicou (4.3).

$$B_{i+1}(x,y) = \begin{cases} B_{i+1}(x,y), & F_i(x,y) > A \\ B_{i+1}(x,y) = (1\text{-}\alpha)B_i(x,y) + \alpha F_i(x,y), & \text{inak} \end{cases} \tag{4.3}$$

Premenná α vyjadruje rýchlosť aktualizácie modelu, premenná B_i je intenzita hodnoty pixelu referenčného snímku, F_i je intenzita hodnoty aktuálneho snímku v čase i. Rýchlosť aktualizácie modelu α postupňe primiešava vo váženom pomere zmeny z aktuálneho snímku [35] do snímku *background modelu*. Ak je však α príliš veľká, môže to mať za následok vytvorenie tzv. umelých „chvostov", vyskytujúcich sa za pohybujúcimi predmetmi [36]. Vďaka tomu, že sa *background model* počíta len ako vážená suma dvoch snímkov, má algoritmus veľmi nízku výpočtovú a priestorovú zložitosť. Dynamická aktualizácia robí tento model adaptívny aj pre veľmi zložité scény.

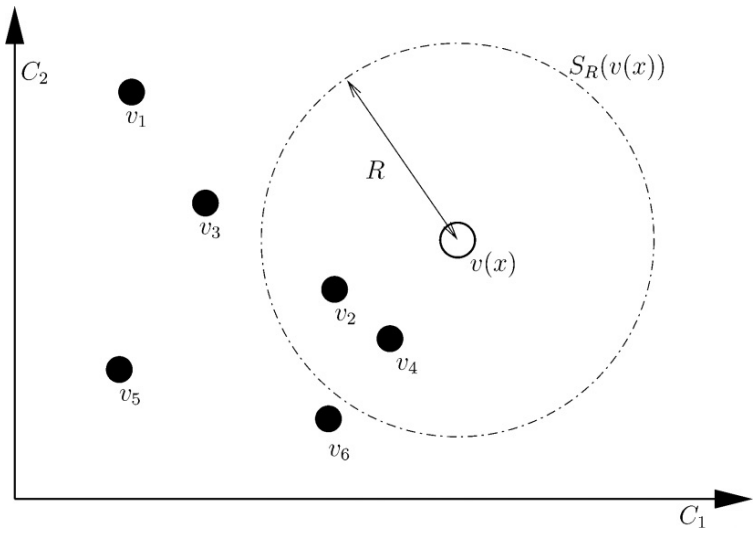
Z hľadiska signálov a systémov sa javí výraz v druhej podmienke rovnice 4.3 ako filter s nekonečnou impulznou odozvou (*IIR*). Z toho dôvodu je metóda *Running average IIR* systémom. Avšak vďaka nízkej výpočtovej zložitosti a pamäťovej kompaktnosti sa bežne používa v systémoch reálneho času dodnes [36, 56, 29].

4.2 ViBe

ViBe je *BGS* metóda, ktorá bola prvý krát prezentovaná na konferencii IEEE ICASP v roku 2009. Bola vyvinutá autormi Oliverom Barnichom a Marcom Van Droogenbroeckom z univerzity Montefiore a následne patentovaná [31]. Patent pokrýva rôzne aspekty ako je stochastické nahrádzanie vzoriek, priestorová difúzia a podobne. *ViBe* bol napísaný a zverejnený v programovacom jazyku *C* a implementovaný pre CPU, GPU a FPGA[1]. Táto sekcia obsahuje súhrn fundamentálnych informácii, ktoré sú potrebné pre základné pochopenie algoritmu. Text vychádza z nasledujúcich článkov: [28, 27].

4.2.1 Model pixelov a klasifikačný proces

Pre odhad funkcie hustoty pravdepodobnosti pixelu (*PDF*) sa využívajú rôzne pokročilé techniky. Prístup algoritmu *ViBe* je však iný, pretože výpočet hodnoty v euklidovskom priestore farieb je obmedzený len na miestne okolie, ako je vidieť na obrázku 4.2. *ViBe* v praxi neposudzuje *PDF*, ale používa na vyhodnotenie výslednej hodnoty pixelu *background modelu* sadu hodnôt, ktoré už boli v minulosti pozorované. Každý pixel pozadia x je modelovaný ako kolekcia N vzoriek *background modelu*: $M(x) = v_1, v_2, ..., v_N$



Obr. 4.2: Porovnanie hodnoty pixelov s množinou vzoriek v 2D euklidovskom priestore farieb $(C1, C2)$. Pre klasifikáciu $v(x)$ je potrebné vypočítať počet vzoriek $M(x)$ pretínajúcich kruh s rádiusom R, stredom v bode $v(x)$. Obrázok prevzatý z [28].

[1]Programovatelné hradlové polia (**Field Programmable Gate Array**) je typ logického integrovaného obvodu, ktorý je vyrobený tak, aby mohol byť naprogramovaní až u zákazníka

4.2.2 Aktualizácia modelu

Klasifikačný krok porovnáva aktuálnu hodnotu pixelu $v_t(x)$ priamo so vzorkami obsiahnutými v *background modeli* predchádzajúceho snímku $M_{t-1}(x)$ v čase $t-1$. Ktoré vzorky je však potrebné si zapamätať a na ako dlho? Klasickým prístupom aktualizácie histórie *modelu* je zahodenie starej vzorky po určitom počte snímok alebo po určitom čase. Otázkou však je, ako sa zachovať k pixelom, ktoré patria do popredia? Ak sa nejakým spôsobom nebudú postupne začlenovať do *modelu*, tak sa model nikdy neprispôsobí zmenám prostredia. Preto nastáva otázka medzi výberom konzervatívneho alebo slepého prístupu:

- **Konzervatívny prístup** - Konzervatívny prístup do *modelu* nikdy nezahŕňa vzorku patriacu do popredia; to však často vedie k *deadlocku*[2] a nikdy nemiznúcim „duchom"; vzorka pozadia je nesprávne klasifikovaná ako vzorka patriaca do popredia, čo bráni aktualizácii *modelového* pixelu.
- **Slepý prístup** - „Slepá" aktualizácia nie je citlivá na *deadlocky*; vzorky sú pridávané do modelu či už sú klasifikované ako vzorky patriace do pozadia alebo popredia; hlavnou nevýhodou tejto metódy je slabá detekcia pomaly pohybujúcich sa objektov, ktoré sú progresívnejšie zaraďované do *background modelu*.

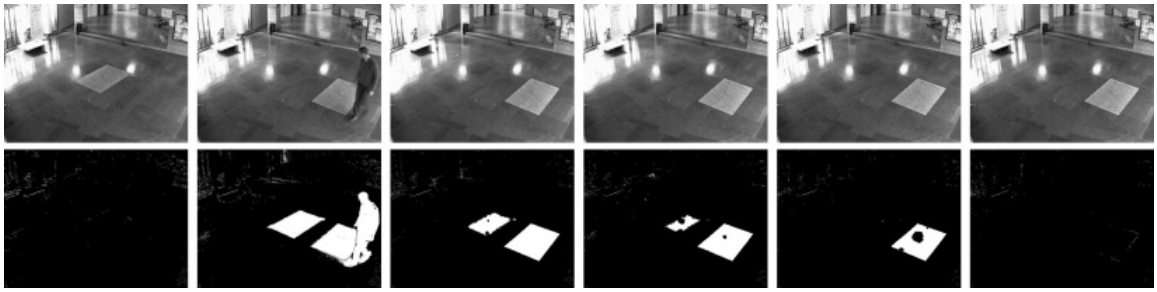
Metóda aktualizácie *ViBe* modelu zahŕňa vo výsledku tri dôležité spôsoby:

- Aktualizačná politika bez potreby využitia pamäte pre históriu, ktorá zaisťuje hladký rozpad životnosti vzoriek historických *modelových* pixelov; namiesto systematického odstraňovania najstaršej vzorky si *ViBe* vyberie vzorku náhodnú.
- Spôsob výberu náhodného časového vzorku (*subsampling*), umožňujúci *modelu* pokryť široké časové okno (histórie vzoriek) za použitia obmedzeného počtu vzoriek; keď je teda hodnota pixelu klasifikovaná ako pozadie, náhodný proces rozhodne, či sa táto hodnota použije pre aktualizáciu zodpovedajúceho *modelového* pixelu.
- Mechanizmus, ktorý propaguje *modelové* vzorky priestorovo pre zabezpečenie priestorovej konzistencie, čo dovoľuje adaptáciu pre pixely, ktoré sú aktuálne maskované popredím; podľa tohto spôsobu sa budú pixely *modelu*, skrývajúce sa za popredím občas aktualizovať vzorkami okolitých pixelov pozadia; *ViBe model* je tak schopný prispôsobiť sa meniacemu sa osvetleniu a štrukturálnym zmenám, pričom spolieha na konzervatívny prístup aktualizácie *modelu*.

Príklad fungovania algoritmu *ViBe* je znázornený na obrázku 4.3, pričom jeho výhody môžme zhrnúť takto:

- Rýchle potlačenie „duchov".
- Odolnosť voči zlému umiesneniu snímacieho zariadenia (vibrácie).
- Odolnosť voči šumu.
- Náročnosť osekanej verzie algoritmu je vhodná aj pre vstavané systémy.

[2] **Deadlock** alebo inak, uviaznutie, je výraz pre situáciu, kedy je úspešné dokončenie prvej akcie podmienené predchádzajúcim dokončením druhej akcie, pričom druhá akcia môže byť dokončená až po dokončení akcie prvej.



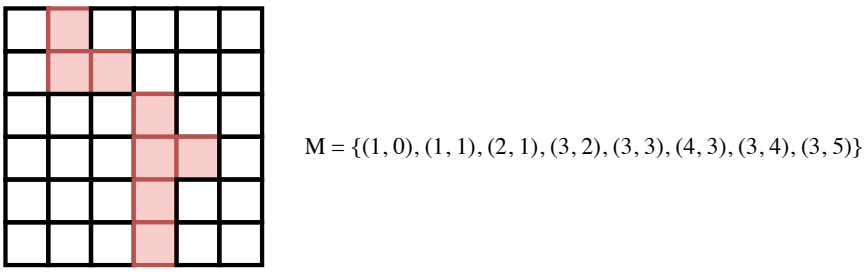
Obr. 4.3: Obrázok znázorňuje rýchle potlačenie „duchov“. V tejto scéne sa nachádza pohybujúci objekt (koberec), ktorý za sebou zanecháva „ducha“ v pozadí a je detekovaný ako časť popredia. Je viditeľné, že duch je do modelu absorbovaný oveľa rýchlejšie ako oblasť popredia zodpovedajúcej fyzickému objektu. Obrázok prevzatý z [28].

4.2.3 Inicializácia modelu

Aj keď sa algoritmus dokáže ľahko zotaviť z akéhokoľvek typu inicializácie, napríklad zvolením náhodnej sady hodnôt, je vhodné, čo najskôr zostaviť presný odhad *modelu* pozadia. V ideálnom prípade by mal byť algoritmus segmentácie detekovaných objektov schopný detekovať objektu už od druhého snímku, pričom prvý snímok by mal slúžiť ideálne na inicializáciu *modelu*. Keďže pred druhým snímkom nie sú dostupné žiadne historické informácie, *ViBe* ich naplní hodnotami nachádzajúcimi sa v okolí každého pixelu. Presnejšie ich inicializuje náhodnými hodnotami pixelov prvého snímku, nachádzajúcimi sa v okolí daného pixelu.

4.3 Algoritmy potrebné pri implementácii vylepšeného algoritmu ViBe

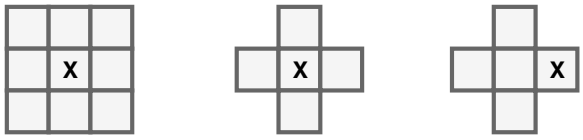
Pri vylepšovaní funkčnosti algoritmu ViBe popísanej v sekcii 6.1 som využil **morfologické operácie**. Pre lepšie pochopenie je treba vysvetliť význam morfológie. Matematická morfológia predpokladá, že obraz sa dá modelovať pomocou bodových množín, ktorého definičným oborom pre popis dvojrozmerných útvarov je euklidovský priestor. Základným prvkom pre binárnu morfológiu je usporiadaná dvojica celých čísiel. Binárny obraz sa dá vyjadriť ako 2D bodovú množinu. Príklad je znázornený na obrázku 4.4.



Obr. 4.4: Príklad bodovej množiny.

Ďalším pojmom je morfologická transformácia, ktorá predstavuje reláciu medzi bodovou množinou X a štrukturálnym elementom B. Štrukturálny element je bodová množina,

skladajúca sa z menšieho počtu bodov, obsahujúca počiatočný bod, zvaný reprezentatívny bod. Príklad je zobrazený na obrázku 4.5.



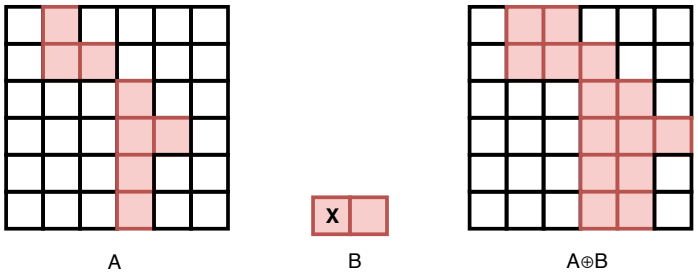
Obr. 4.5: Príklad štrukturálnych elementov.

4.3.1 Základné binárne morfologické operácie

Medzi základné binárne morfologické operácie patrí dilatácia, erózia a morfologické otvorenie a morfologické uzavretie.

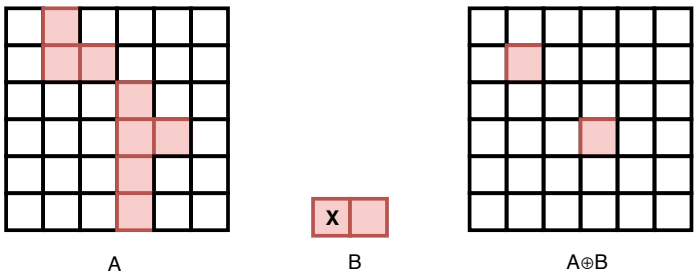
Dilatácia a erózia

Dilatácia $\oplus$ skladá body dvoch množín pomocou vektorového súčtu a je vyjadrená následovne: $A \oplus B = \{p \in \varepsilon^2 : p = a + B, a \in A, b \in B\}$. Používa sa k zaplneniu malých dier, úzkych zálivov a ako základ pre zložitejšie operácie. Dilatácia zväčšuje objekty. V prípade zachovania pôvodného rozmeru sa dilatácia kombinuje s eróziou. Príklad dilatácie je zobrazený na obrázku 4.6.



Obr. 4.6: Príklad dilatácie so štrukturálnym elementom B a bodovej množiny A.

Erózia Θ je duálna operácia k dilatácii a je založená na vektorovom rozdiele. Erózia je vyjadrená následovne: $A \Theta B = \{p \in \varepsilon^2 : p + b \in A \text{ pre každé } b \in B\}$. Erózia sa používa na zjednodušenie štruktúry objektov. Erózia stenčuje objekt.

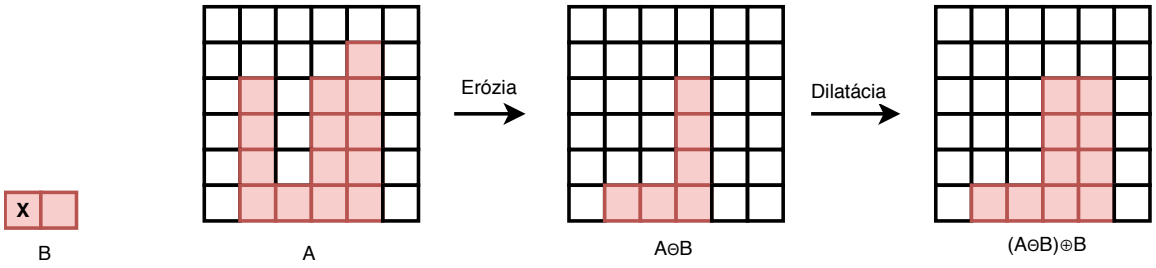


Obr. 4.7: Príklad erózie so štrukturálnym elementom B a bodovej množiny A.

Uzavretie a otvorenie

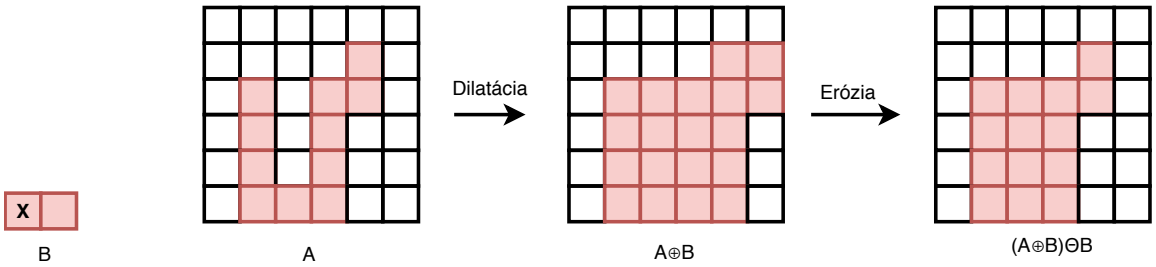
Kombináciou dilatácie a erózie vzniknú ďalšie dve významné morfologické operácie, ktorými sú otvorenie a uzavretie. Morfologické uzavretie a otvorenie sa používa pre zníženie množstva detailov v obraze, tvar objektu však zostáva neporušený. Obidve operácie sú idempotentné (opätovná aplikácia nemení predošlý výsledok).

Pri **otvorení** sa najprv vykoná erózia a následne dilatácia. Otvorenie oddeľuje objekty spojené tenkou čiarou a odstraňuje šum. Operácia otvorenia je znázornená na obrázku 4.8.



Obr. 4.8: Príklad otvorenia bodovej množiny A so štrukturálnym elementom B.

Uzavretie spojuje blízke objekty a zaplňuje diery. Pri uzatvorení sa najprv vykoná dilatácia a potom erózia. Operácia uzavretia je znázornená na obrázku 4.9.



Obr. 4.9: Príklad zatvorenia bodovej množiny A so štrukturálnym elementom B.

4.4 Technológie

Nasledujúca sekcia obsahuje súhrn použitých technológií pre implementáciu detekčných algoritmov. Pre prvotný prototyp aplikácie bola využitá knižnica `OpenGL ES`. Finálna verzia aplikácie využíva knižnicu `Metal`.

4.4.1 Metal

Jedná sa o knižnicu pre 3D vykresľovanie a paralelné výpočty na grafickom čipe, ktorej hlavným cieľom je minimalizovať režijne náklady procesoru spojené s vykonávaním úkonov na GPU. Na rozdiel od `OpenGL ES` je Metal navrhnutý výhradne pre platformy spoločnosti Apple.

Jedna z kľúčových vlastností knižnice je popísaná v zmysle „Robiť náročné veci menej často“ [16]. Pre ozrejmenie je treba priblížiť princíp vykresľovania pomocou OpenGL (ES). Ten je možné chápať ako sled udalostí, ako napríklad nastavenie `shader`ov, textúr, `vertex bufferu` a volanie vykresľovacích funkcií. V okamžiku volania vykresľovacej funkcie sa začne

vykonávať validácia stavov, poprípade kompilácia `shader`ov a taktiež samotná operácia na GPU. Táto postupnosť sa vykonáva pre iné vykresľované objekty znovu. Naproti tomu Metal volí prístup, pri ktorom sa náročné operácie ako napríklad validácia stavov alebo kompilácia `shader`ov vykonáva pri samotnej kompilácii aplikácie. Vo vnútri vykresľovacieho cyklu sa vykonávajú len operácie na GPU.

Ďalšou nespornou výhodou oproti OpenGL (ES) je práca s viacvláknovým procesorom. V OpenGL môže v rámci aktuálneho kontextu generovať príkazy len jedno vlákno, v Metal-e sa podobný globálny kontext nevyskytuje. Metal je optimalizovaný priamo na konkrétny hardware a poskytuje maximálny možný výkon. Oproti OpenGL je nárast vo výkone v niektorých situáciách až viac ako 200%.

Rozhranie knižnice Metal

Medzi kľúčové objekty pre vykresľovanie v Metal-e patrí objekt, pomocou ktorého sa popisuje aktuálny stav vykresľovacieho reťazca (**Pipeline state object**). Jedná sa o nemenný objekt, ktorý zapúzdruje informácie ako formát vstupných dát alebo použité `vertex` / `fragment` funkcie. Ďalší je objekt reprezentujúci jeden priechod vykresľovacím reťazcom (**Render pass descriptor**), slúžiaci ako kontajner pre prílohy (`attachments`). Pre každú prílohu je možné definovať tzv. *store* a *load* akcie, ktorými definujeme, či bude príloha slúžiť na čítanie, zápis alebo oboje. Na generovanie príkazov pre GPU slúži **Render command encoder**.

Generovanie príkazov

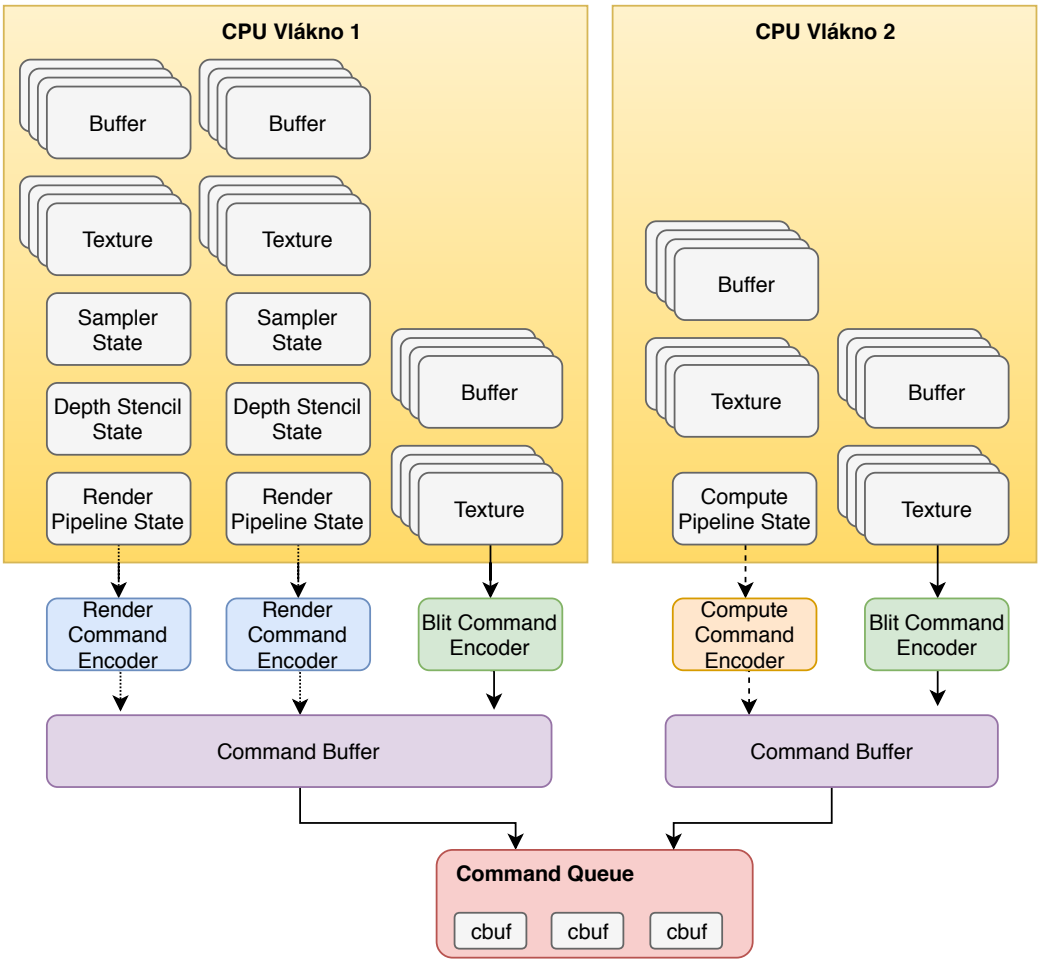
Pre generovanie príkazov sa používajú tzv. `enkódery`, ktoré prevádzajú príkazy knižnice, na príkazy pre grafický čip. Metal obsahuje viacero typov `enkóderov`:

- **Render command encoder** – slúži pre generovanie HW príkazov pre jeden priechod vykresľovacím reťazcom,
- **Parallel render command encoder** – objekt, ktorý rozdelí jeden vykresľovací priechod reťazcom (`render pass`), aby mohol byť súčasne enkódovaný z viacerých vlákien,
- **Blit command encoder** – slúži pre graficky akcelerované operácie s dátami a textúrami,
- **Compute command encoder** – vhodný pre obecné výpočty na GPU.

Príkazy pre GPU generované pomocou `enkóderov` sú uchovávané v štruktúre `Command buffer`, ktorú je možné generovať na viacerých vláknach súčasne, ako je znázornené na obrázku 4.10. V tejto fáze už neprebieha žiadna validácia a jedná sa teda o priamu komunikáciu s ovládačom GPU.

Shadery

Pre písanie `shader`ov v Metale sa používa jazyk, založený na podmnožine jazyku C++. Obsahuje taktiež rozšírenia, ktoré vylepšujú možnosti interakcie s GPU a taktiež možnosť preťažovať funkcie a šablony. Rekurzia je však zakázaná [5].



Obr. 4.10: Súhrnná schéma paralelného generovania príkazov v Metal-e.

Metal Performance Shaders

Metal Performance Shaders (MPS) je kolekcia optimalizovaných grafických `shaderov` a `compute shaderov`, ktoré sú navrhnuté pre jednoduchú integráciu do Metal aplikácie. Aby bol zaručený čo najoptimálnejší výkon, tieto data-paralelné primitíva sú špeciálne optimalizované pre každú jednu generáciu grafických procesorov od Apple-u. Obsahujú rôzne typy hotových `shaderov` pre:

- Konvolučné filtre (Sobel, Gaussian)
- Morfologické operácie (diletácia, erodovanie)
- Operácie pre prácu s histogramom (ekvalizácia, špecifikácia)

4.4.2 OpenGL ES

V tejto práci boli prvé iterácie detekčného algoritmu implementované v OpenGL ES, avšak kvôli problémom spomínaných v kapitole 6 bola na finálnu implementáciu zvolená knižnica OpenGL ES. Je však potrebné sa oboznámiť so základnými stavebnými kameňmi knižnice. Knižnica vychádza z názvu *Open Graphics Library for Embedded Systems* a ako názov

napovedá, jedná sa o grafické rozhranie pre 3D grafické vykresľovanie cielené na prenosné a vstavané zariadenia.

Architektúra bola navrhnutá tak, aby bola knižnica použiteľná na rôznych operačných systémoch a grafických čipoch. Jadro je implementované v jazyku C, avšak jej aplikačné rozhranie je dostupné takmer pre každý programovací jazyk. OpenGL ES využíva koncept `klient-server`, kde klient prekladá dáta do formátu, ktorému grafický hardware rozumie a posiela ich na server. Server spracováva príkazy a vykonáva ich v poradí v ktorom prichádzajú.

Základné stavebné kamene OpenGL ES:

- **Kontext** – Je dátová štruktúra, obsahujúca všetky stavové informácie potrebné k vykresleniu scény. Medzi tieto informácie patria napríklad stavové premenné, ktoré nie sú programovateľné, fragment `shadery`, vertex `shadery`, pole vrcholov alebo aktuálny `framebuffer`.
- **Framebuffer** – Existujú dva typy `framebuffer`ov. Prvý predvolený obvykle reprezentuje okno aplikácie, poprípade obrazovku. Druhým typom sú tzv. `framebuffer objekty`, ktoré predstavujú textúru a nie sú priamo viditeľné.
- **Buffer objekty** – Sú objekty pre správu dát pevnej velikosti v pamäti. Medzi také objekty patrí napríklad `vertex buffer object (VBO)` pre uloženie vrcholov. Tie umožňujú alokovať priestor priamo v pamäti GPU, čím sa zabráni opätovnému posielaniu dát pri každom vykresľovacom cykle.
- **Kontajnerové objekty** – Medzi ne patrí spomínaný `framebuffer objekt` alebo `Vertex array object (VAO)`.

Shadery

Pre programovanie `shader`ov, teda programovateľných častí grafického reťazca slúži jazyk `GLSL` (OpenGL Shading Language), ktorý je založený na jazyku C. Jazyk však obsahuje rozšírenie pre prácu s vektorovými a maticovými typmi a niektoré mechanizmy jazyka C++ ako napríklad preťažovanie funkcií.

Vhodnou poznámkou je fakt, že verzia OpenGL ES pre iOS (aktuálne 3.0) nepodporuje `geometry shader` a ani `compute shader`, ktorý je vhodný na obecné výpočty (napríklad pre detekčné algoritmy).

Kapitola 5

Analýza funkcionality a návrh používateľského rozhrania

Pre vytvorenie aplikácie s dobrým používateľským rozhraním a funkcionalitou je vhodné zmapovať trh a analyzovať existujúce riešenia, z ktorých je možné sa inšpirovať a navrhnúť tak riešenie, ktoré bude používateľom poskytovať to, čo naozaj od danej aplikácie očakávajú. Prvá časť kapitoly sa venuje analýze funkcionality najpopulárnejších existujúcich riešení. Druhá časť je venovaná prehľadu funkcií ostatných populárnych aplikácií v prehľadnej tabuľke s popisom. Ďalšou časťou bude špecifikácia požiadaviek, ktorá spolu s diagramom prípadu použitia utvorí funkčný základ novej aplikácie a jej používateľského rozhrania. Posledná časť bude venovaná návrhu používateľského rozhrania, ktoré berie ohľad na špecifikáciu požiadaviek a prípady použitia.

5.1 Analýza funkcionality vybraných aplikácií

Na analýzu boli vybraté ku dňu 2.5.2018 tri najpopulárnejšie aplikácie s funkčnosťou pre prenos audia a videa, pre hľadaný výraz „Home security“. Analýza sa venuje celkovému dojmu aplikácie, funkčnosti a spoľahlivosti prenosu audia a videa a taktiež detekcii obrazu. Aplikácie boli testované na aktuálne najnovšom operačnom systéme iOS 11.3.1 a na vybraných zariadeniach:

- iPhone 7
- iPad 2017
- iPhone SE

Aplikácia AtHomeVideo[1]

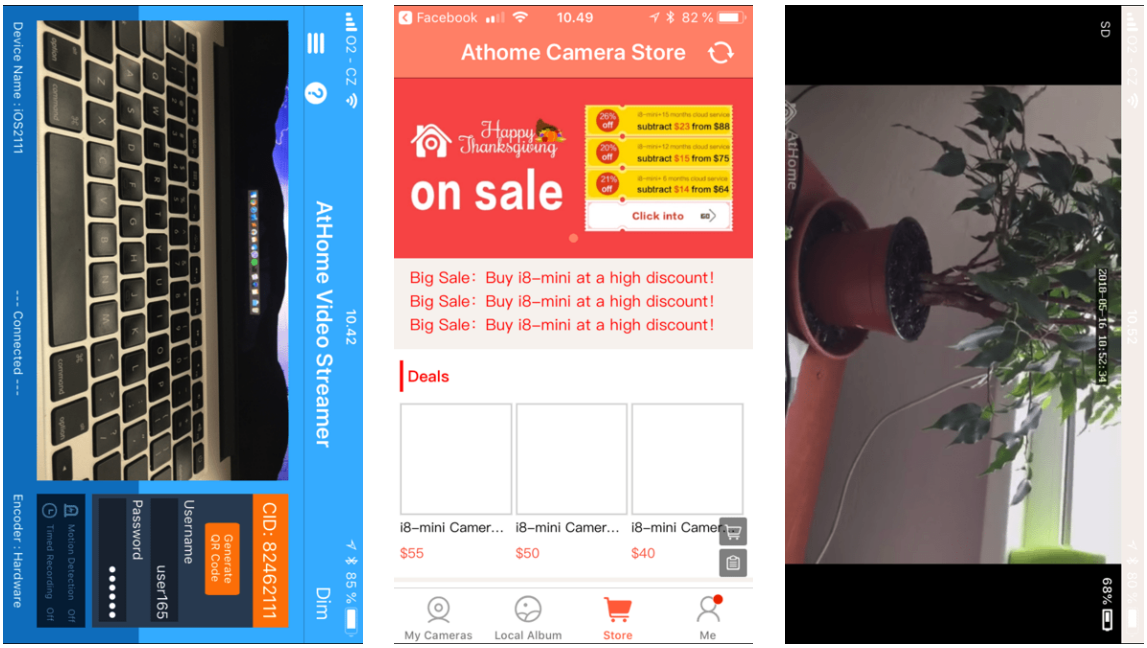
AtHomeVideo používateľom ponúka dve separátne aplikácie. Jednou je Streamer, slúžiaci ako `slave` jednotka vysieľajúca mediálny stream a ďalšou Viewer, slúžiaci ako `master` jednotka prijímajúca mediálny stream. Tieto aplikácie majú zastaralý design. Menu aplikácie je pretkané zbytočnými možnosťami ako hodnotenie aplikácie, zaslanie logov alebo sekciu o aplikácii, ktoré sú súčasťou *AppStoru* a v aplikácii nemajú čo hľadať. Aplikácie obsahuje obchod s kamerami a na každom rohu je pretkaná reklamami. Režim kamier je podporovaný

[1] https://itunes.apple.com/us/app/athome-video-streamer-cctv-cam/id590416158?mt=8

len v orientácii `landscape`[2]. Aplikácia obsahuje veľké množstvo grafických chýb akými sú napríklad nedosiahnuteľné tlačidlá a podobne. Ďalším príkladom nevhodného návrhu UI je, že zapnutie blesku mobilného telefónu sa zobrazuje aj na tabletoch, ktoré im nedisponujú, tlačidlo na otočenie kamery sa nachádza až v nastaveniach a podobne.

Čo sa týka prenosu audia a videa, autori uvádzajú, že dáta prechádzajú šifrovane cez ich server alebo `peer-to-peer`. Obraz bol prenášaný v nízkom rozlíšení a za vyššie rozlíšenie je treba platiť mesačný poplatok. Aplikácia podporuje aj prenos prostredníctvom mobilných sietí. Táto funkcionalita fungovala dobre, avšak napájanie sa na druhú jednotku trvalo niekedy rádovo desiatky sekúnd a spoľahlivosť kazil aj fakt, že počas krátkeho testovania aplikácia tri krát spadla. Ak bola master a slave jednotka blízko seba, začala sa hlasito prejavovať zvuková spätná väzba, ktorá nie je odfiltrovaná. Párovanie je cez QR kód.

Aplikácia disponuje detekciou pohybu. Detekciu sa mi podarilo spustiť až na niekoľký pokus, avšak pri pohybe žiadna notifikácia nechodila. Táto funkcia je teda nepoužiteľná.



(a) obrazovka (b) obrazovka (c) obrazovka

Obr. 5.1: Ukážka používateľského prostredia aplikácie AtHomeVideo.

Aplikácia Manything[3]

Ďalšou populárnou aplikáciou je aplikácia Manything, vyžadujúca registráciu cez sociálne siete. Aplikácia je bohatá na nastavenia pre `power userov`[4] a obsahuje aj údaje o objeme prenesených dát. Aplikácia používanie gamifikuje[5] a používateľ môže zverejňovať nahrané videá s komunitou aplikácie a kategorizovať ich do rôznych kategórií. Videá sú následne hodnotené komunitou.

[2] **Landscape** – zariadenie otočené na šírku obrazovky.

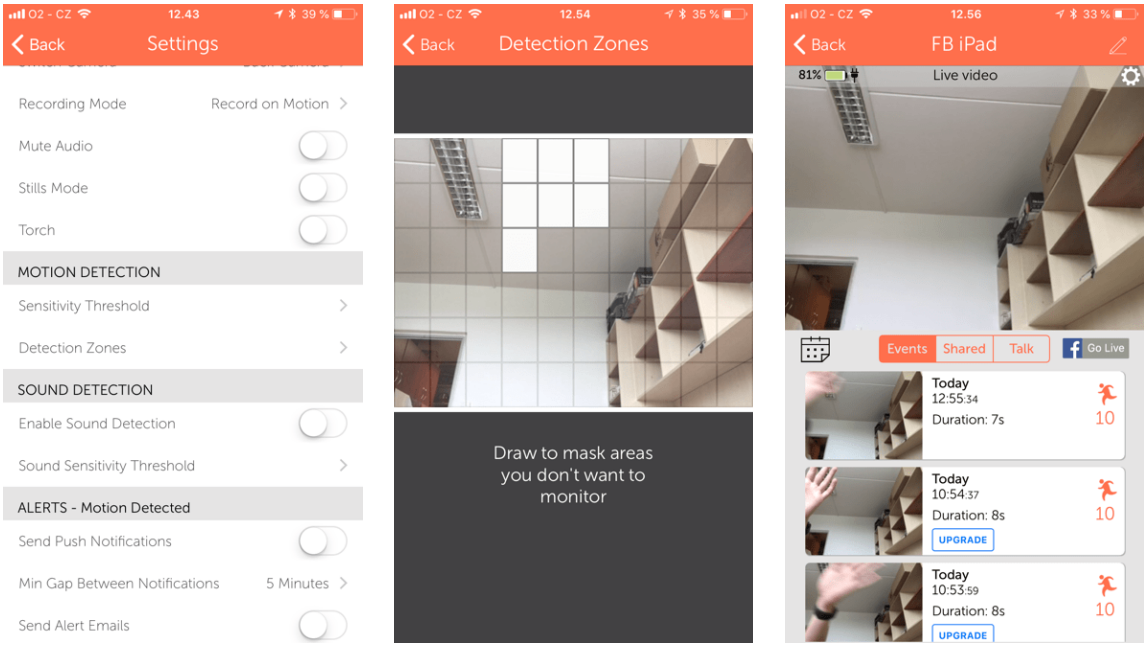
[3] https://itunes.apple.com/us/app/manything/id639672976?mt=8

[4] **Power user** – pokročilý používateľ, využívajúci pokročilé funkcie, ktoré bežný používateľ nevyužíva.

[5] **Gamifikácia** – technika marketingu pre zvyšovanie záujmu klientov prostredníctvom užívania herných prostriedkov.

Predvolené nastavenie aplikácie poskytuje prenos audia a videa v nízkej kvalite, ktoré je však spoľahlivé a s dobrou latenciou. Pri miernom zvýšení kvality spojenie vypadáva a snímková frekvencia klesá. Prenos cez mobilnú sieť (LTE) vypadával. Aplikácia sa nedokáže prepnúť z WiFi na LTE. Ovládacie prvky ako napríklad prepnutie na opačnú kameru zariadenia je hlboko v nastaveniach.

Aplikácia obsahuje jednoduché rozpoznávanie pohybu a zasielanie prípadných notifikácií alebo emailov. Tieto notifikácie však chodia veľmi nepravidelne, niekedy v jednotkách minút. Výsledný krátky záznam je k dispozícii v aplikačnej galérii, avšak prehrať je možné len posledný záznam. Nastavenie detekcie obsahuje zaujímavú možnosť zvolenia oblasti, ktorá má byť detekovaná.



(a) obrazovka (b) obrazovka (c) obrazovka

Obr. 5.2: Manything

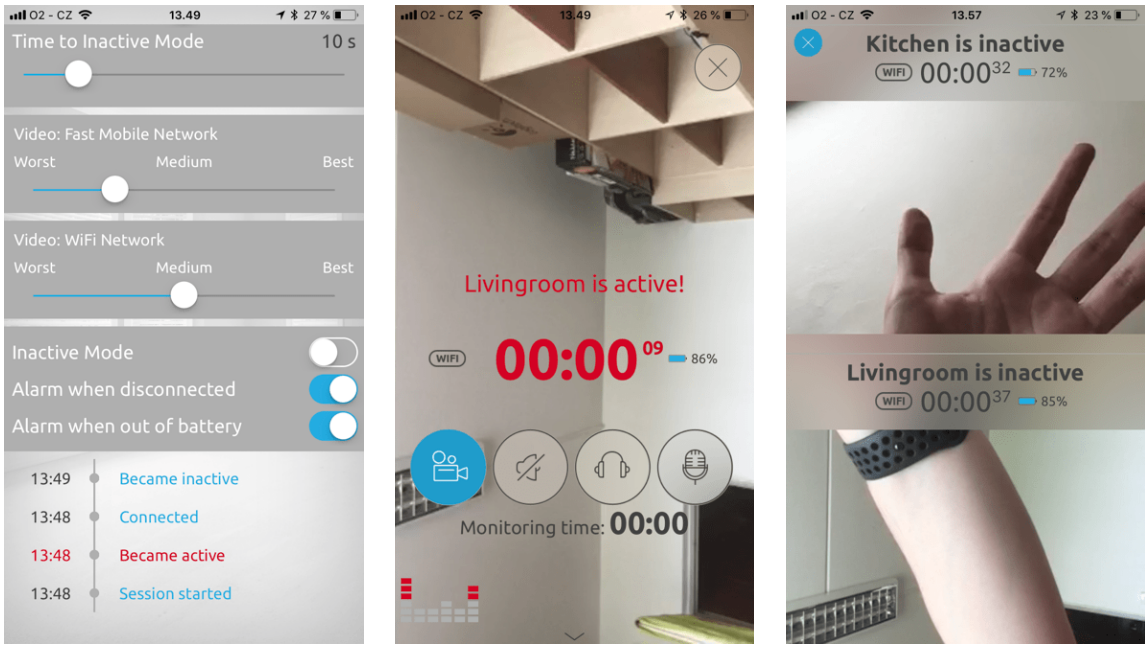
Aplikácia Home Security Monitor Camera[6]

používateľské rozhranie tejto aplikácie pôsobí vyladeným a uceleným dojmom. Rozhranie aplikácie je jednoduché a prehľadné. Najdôležitejšie nastavenia sa nachádzajú priamo na monitorovacej obrazovke a pokročilejšie nastavenia sú dostupné pod monitorovacou obrazovkou, takže pri nastavovaní sa nestratí pohľad na monitorovaný priestor. používateľovi sa ukazuje doba, ktorú bolo zariadenie aktívne v danom režime. Režim sa mení v závislosti na hluku alebo stave pripojenia. Master jednotka zobrazuje hodnotu batérie sledovanej jednotky.

Autori aplikácie uvádzajú, že prenos je šifrovaný a pri lokálnej sieti funguje peer-to-peer. Pre použitie s mobilnými sieťami je potrebné platiť mesačný poplatok. Pri dobrom pripojení je kvalita videa vysoká, avšak obraz prichádza mnoho krát s veľkým oneskorením. Aplikácia disponuje možnosťou obojstrannej audio komunikácie. Zaujímavou funkcionalitou je sledovanie prenosu viacerých zariadení naraz.

[6] https://itunes.apple.com/us/app/home-security-monitor-camera/id1097819311?mt=8

Home Security Monitor Camera má možnosť nastavenia notifikácií pri zvýšenej hladine zvuku, ktorá je nastaviteľná. Detekciou pohybu však aplikácia nedisponuje.



(a) obrazovka (b) obrazovka (c) obrazovka

Obr. 5.3: Home Security

5.2 Kompletný prehľad funkcionality

Spomínané aplikácie patrili medzi top 3 najsťahovanejšie avšak zaujímavé aplikácie sa nachádzali aj v zozname pod nimi. Našli sa tam aj aplikácie, ktoré boli spoľahlivejšie alebo ponúkali rozšírenú funkcionalitu. Väčšina z aplikácii dokázali prenášať audio aj video. Mnohé z nich obsahovali určitý mechanizmus oznamovania, či už pri detekcii pohybu alebo zvuku. Tieto oznámenia sa zasielali vo forme emailu alebo mobilnej notifikácie. Pri detekcií pohybu boli snímky ukladané na server autorov alebo v prípade aplikácie WardenCam na Google účet. Vyše polovica aplikácii, umožňujúcich detekciu pohybu obsahovali galériu zachytených snímkov.

Čo sa týka spoľahlivosti prenosu videa, väčšina aplikácií v tomto zozname na tom nebola veľmi dobre. Video bolo často veľmi oneskorené oproti zdroju, malo malú snímkovaciu frekvenciu (7fps) a rozlíšenie nebolo uspokojujúce. Niektoré z nich však punúkali za príplatok zvýšenie kvality. Výnimkou bola aplikácia *Alfred*, ktorej jadro beží pravdepodobne na technológii `WebRTC`. Úsudok vznikol z toho, že aplikácia má aj webovú verziu, ktorá vyžaduje na platforme *MacOS* webový prehliadač *Safari* vo verzii 11.0 [53]. V tejto verzii pribudla práve podpora `WebRTC` [32]. Aplikácia pri prechode medzi sieťami (WiFi -> LTE a naopak) nedokázala znovu nadviazať spojenie so slave[7] jednotkou a bolo treba aplikáciu reštartovať.

Polovica aplikácií dokázala vysielať mediálny stream na viaceré jednotky. To je výhodné v situácii, kedy chcú mať napríklad rodinný príslušníci naraz dohľad nad domácnosťou.

[7] **Slave jednotka** – zariadenie, ktoré vysiela mediálny stream.

V prípade zdieľania jednotiek viacerými členmi je potrebné zdieľať jeden rodinný účet. Ostatné aplikácie fungovali len v režime `1:1`.

používateľom, ktorý by chceli neustále využívať aplikáciu ako monitorovacie centrum a monitorovať tak viacero miestností naraz, ponúkali aplikácie *AtHomeVideo* a *Home Security Monitor Camera* možnosť sledovania viacerých jednotiek naraz.

Dôležitým faktorom bol aj vzdialený prístup k domácim jednotkám. Väčšina aplikácií fungovala obstojne pri pripojení do domácej sieti, ale akonáhle bola master[8] jednotka pripojená z vonkajšej alebo mobilnej siete, kvalita prenosu sa rapídne zhoršila. Aplikácie, obsahujúce túto funkcionalitu sa ale nedokázali prispôsobiť zmenám siete a bolo nutné ich reštartovať.

	AtHomeVideo	Manything	Home Security	WardenCam	Presence	IP Webcam	Alfred	Camera Surveillance
Prenos videa	✔	✔	✔	✔	✔	✔	✔	✔
Prenos audia	✔	✔	✔	✔	✔	✘	✔	✔
Detekcia pohybu	✔	✔	✘	✔	✔⊕	✘	✔	✘
Upozornenia notifikáciou	✘	Notifikácia	Notifikácia	Email	Notifikácia	✘	Notifikácia	✘
Oneskorenie prenosu videa (1-5)	3	3	2	2	2	2	1	2
Snímkovacia frekvencia videa (1-5)	3	3	2	5	4	2	1	2
Spoľahlivosť prenosu videa (1-5)	5	4	2	4	2	2	1	2
Kvalita prenosu videa (1-5)	3⊖	3	3⊖	3	5	3	2⊖	2
Sledovanie prenosu na viacerých jednotkách	✔	✘	✔	✘	✔	✔	✘	✘
Sledovanie viacerých jednotiek naraz	✔	✘	✔	✘	✘	✘	✘	✘
Galéria	✘	✔	✘	✘⊗	✔	✘	✔	✘
Vzdialený prístup odkiaľkoľvek	✔	✔	✔⊘	✘	✔	✘	✔	✔
Párovanie jednotiek	QR	Účet	Účet	Účet	Účet	IP adresa	Účet	PIN

Tabuľka 5.1: Prehľad funkcionality analyzovaných aplikácií. ⊕ – nepoužiteľná funkcia. ⊖ – za príplatok väčšia kvalita. ⊗ – na ukladanie snímkov použitý Google účet. ⊘ – platená funkcionalita. Rozsah hodnotenia – 1 najlepšie, 5 najhoršie.

Poslednou funkciou bol spôsob párovania zariadení. 60% z nich ponúkalo párovanie pomocou vytvorenia používateľského účtu. Výnimkou bola aplikácia AtHomeVideo, pri ktorej sa dali zariadenia spárovať QR kódom. Ďalšie dve aplikácie poskytovali párovanie pomocou

[8] **Master jednotka** – zariadenie, ktoré prijíma mediálny stream.

zadania IP adresy alebo vygenerovaného kódu PIN. Výsledné porovnanie je vidieť v tabuľke 5.1.

5.3 Špecifikácia požiadaviek

Táto sekcia zhrňuje zistené poznatky z predchádzajúcej analýzy existujúcich riešení. Berú sa do úvahy len najdôležitejšie rysy aplikácie. Zároveň kladie dôraz na osobné preferencie, ktoré sú do požiadaviek taktiež zaradené.

Jednoduché používateľské rozhranie (UI)

Väčšina analyzovaných aplikácií malo buď komplikované alebo nezrozumiteľné UI. Aplikácie mali veľa krát príliš veľa nastavení, ktoré používateľa odradzujú od používania aplikácií. Aj prítomnosť reklám na nevhodných miestach zhoršovala používateľský zážitok. Moje riešenie by malo mať čo najmenej používateľských prvkov a nastavení, ktoré by nútili používateľa premýšľať nad voľbou.

Jednoduché párovanie jednotiek

Párovanie jednotiek je prvá vec, ktorú musí používateľ vykonať pred začatím používania akejkoľvek z aplikácií. Je potrebné, aby `master` a `slave` jednotka o sebe vedeli a aby bol proces párovania a komunikácie zabezpečený. Moje riešenie by malo zabezpečiť nulovú konfiguráciu a jednotky by sa mali automaticky prepojiť. Na to bola použitá technológia *CloudKit*, popísaná v sekcii 2.4, v prvej iterácii aplikácie však bola použitá technológia *Firebase*[9].

Prenos audia a videa

Jednou z najdôležitejších funkcií aplikácie pre bezpečnosť domácnosti je prenos audia a videa. Niektorým analyzovaným aplikáciám dosť často vypadávalo spojenie alebo samotné aplikácie padali. Ďalšie mali problémy s kvalitou prenášaného videa alebo oneskorením s malou snímkovacou frekvenciou. Moje riešenie by malo prenášať audio a video vo vysokom rozlíšení v reálnom čase, s vysokou obnovovacou frekvenciou obrazu (30fps) na kvalitnom pripojení. Cielenú funkcionalitu poskytuje multiplatformový projekt `WebRTC`, ktorý bol optimalizovaný, aby najlepšie slúžil tomuto zámeru. Detaily o fungovaní sú popísané v kapitole 3 sekcii 3.1

Vzdialený prístup odkiaľkoľvek

Vzdialený prístup odkiaľkoľvek je komplexným problémom, ktorý nie každá aplikácia zvládla a aj tie, ktoré túto funkcionalitu implementovali nedokázali pri prechode medzi sieťami (WiFi <-> LTE) znovu obnoviť spojenie. Moje riešenie by malo túto funkcionalitu zabezpečiť a to vďaka správnemu využitiu WebRTC a správnej implementácii signalizačného protokolu. Bez tejto funkcionality by nemohol používateľ vzdialene pristupovať k jednotke, ktorá sa nachádza doma.

[9] **Firebase** – https://firebase.google.com/

Detekcia pohybu

Polovica aplikácii nemala implementovanú žiadnu formu detekčného algoritmu. Jedinou možnou voľbou v prípade, že chce užívateľ ochrániť domácnosť je sledovať neustále priamy video prenos. Niektoré z aplikácií, ktoré mali implementovanú detekciu pohybu však reagovali aj na postupnú zmenu sveteľných podmienok alebo na šum, prítomný v interiéry. Naopak niektoré z nich nezachytili pomalý pohyb. Moje riešenie by malo zachytiť či už pomalý alebo rýchly pohyb v interiérovom prostredí za bežných alebo zhoršených svetelných podmienok.

Notifikácie

Pri detekovaní pohybu zasielali niektoré z aplikácií email s priloženým obrázkom na registrovaný účet. Vhodnejším spôsobom by pri detekovaní pohybu mala byť notifikácia obsahujúca samotný obrázok s názvom daného zariadenia. To zabezpečí technológia `CloudKit`.

Galéria zachytených snímkov

Niektoré z aplikácií implementovali galériu, niektoré z nich však túto funkcionalitu mali ako platenú a zadarmo ponúkali len zobrazenie posledného snímku. Výsledná aplikácia by mala ponúkať prehľadnú galériu zobrazujúcu snímky z času, kedy bol detekovaný pohyb, chronologicky.

Zdieľanie sledovaných jednotiek

Pre sprístupnenie sledovaných jednotiek napríklad rodinným príslušníkom je potrebné implementovať bezpečné zdieľanie jednotiek. Túto funkcionalitu by mala taktiež zabezpečiť správna implementácia s použitím `CloudKitu`.

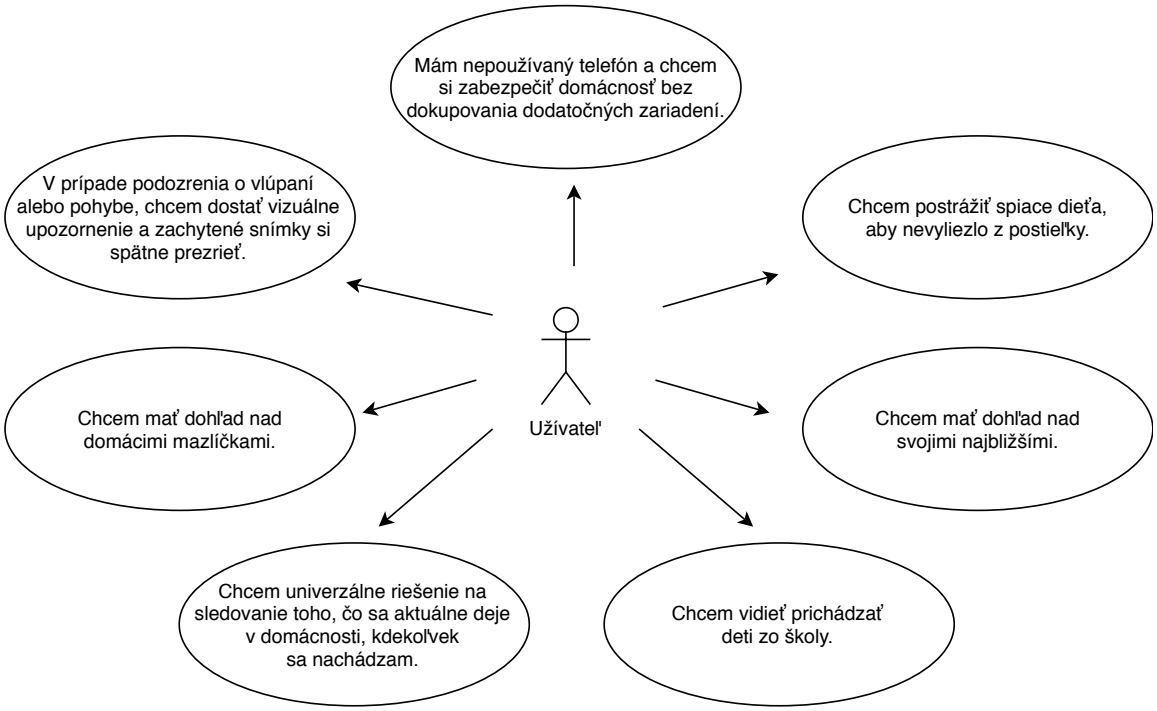
5.4 Návrh používateľského rozhrania

Pri návrhu užívateľského rozhrania je vhodné začať prípadom užitia. Ten odhalí, pre koho je aplikácia určená a aká funkcionalita sa od aplikácie očakáva. Z prípadu použitia sa vytvorí `wireframe`[10] obrazoviek a taktiež návrh navigácie medzi obrazovkami. Tieto obrazovky budú v jednotlivých sekciách detailnejšie popísané.

Prípady použitia

Diagram prípadu použitia zobrazený na obrázku 5.4 poskytuje prehľad hlavných rysov a funkcií aplikácie. Aplikácia bude určená pre bežného používateľa, ktorý si chce jednoducho zabezpečiť domácnosť bez toho, aby dokupoval dodatočné zariadenia. Aplikácia by ho pri podozrivom pohybe mala upozorniť a zaslať mu notifikáciu obsahujúcu snímok s podozrivým pohybom. Dôležitým aspektom je vzdialený prístup odkiaľkoľvek, inak by bolo používanie obmedzené len na lokálnu sieť. Užívateľom môže byť aj človek, ktorý chce mať dohľad nad svojimi najbližšími alebo chce vidieť čo robia jeho domáce zvieratá.

[10] **Wireframe** – definuje textový aj grafický obsah, rozmiestnenie funkčných prvkov, ale aj navigáciu a znenie nadpisov, kľúčových textov či tlačidiel.



Obr. 5.4: Diagram prípadu užitia reflektujúci špecifikáciu požiadaviek.

Navigácia v aplikácii

Používateľská skúsenosť (UX) je pri aplikácií zásadným faktorom. Používateľská skúsenosť pred-stavuje ľudské vnímanie a reakcie, ktoré sú výsledkom z používania alebo predpokladaného používania produktu, služby alebo systému [26].

Na obrázku 5.4 je znázornený finálny návrh obrazoviek aplikácie a navigácia medzi nimi. Návrh vychádza z diagramu použitia a zo špecifikácií požiadaviek navrhnutých v sekcii 5.3. Počiatočnou obrazovkou je obrazovka číslo 1. Z ostatných obrazoviek sa dá cez tlačidlo spätnej šípky, nachádzajúcej sa vo vrchnej časti obrazovky alebo gestom potiahnutia obrazovky zľava doprava, dostať naspäť na obrazovku počiatočnú.

Úvodná obrazovka

Pri prvom spustení aplikácie sa používateľ ocitne na hlavnej obrazovke, bez potreby registrácie alebo prihlasovania. Každý používateľ iOS zariadení využíva na sťahovanie aplikácií z AppStoru[11] iCloud účet. Tento účet bude použitý na synchronizáciu nastavení a jednotiek. Používateľ sa bude môcť z danej obrazovky dostať do obrazovky so slave jednotkou (Kamera) alebo do obrazovky obsahujúcej zoznam slave jednotiek (Monitoring centrum). Ak po prvýkrát vyberie používateľ možnosť kamery, dostane sa na obrazovku, ktorá sa ho spýta na pomenovanie danej kamery. Toto nastavenie bude automaticky synchronizované s osobným účtom iCloud. Detaily fungovania iCloudu a technológie CloudKit sú popísané v sekcii 2.4.

[11] **AppStore** – aplikácia umožňujúca nakupovať a sťahovať aplikácie pre operačný systém iOS.

Slave jednotka (Kamera)

Ak už používateľ zadal názov jednotky (kamery), dostáva sa do obrazovky s danou slave jednotkou. Táto obrazovka ponúka náhľad aktuálne vybratej kamery. Používateľ si bude môcť tlačidlom *„Otoč kameru“* zvoliť, či bude používať kameru zadnú (rear-facing) alebo kameru prednú (front-facing). Ak bude používateľ chcieť zdieľať kameru s ostatnými členmi domácnosti, jednoduchým tlačidlom pre zdieľanie vytvorí pomocou systémového dialógového okna pozvánku, ktorú odošle ako správu, email alebo len skopíruje vygenerovaný odkaz (viď. sekcia 2.4). Po akceptovaní jednotky druhým používateľom je možné toto právo zdieľania odobrať v rovnakom dialógovom okne ako pri zdieľaní.

Pre pohodlnejšie používanie funkcie detekcie pohybu bude zapnutie alebo vypnutie tejto funkcie prístupné zo všetkých master jednotiek a taktiež samotnej slave jednotky. Pri zapnutí detekcie pohybu sa zobrazí upozornenie, že jednotka sa stabilizuje. Po stabilizovaní sa zmení farba rámu popisujúceho aktívnu detekčnú oblasť. Rám je zobrazený na obrázku 5.4 na obrazovke č.3. Tento rám zmení farbu na červenú v prípade zvýšenej úrovne pohybu.

Jednotku bude môcť aj vymazať a to tlačidlom na vrchu obrazovky. Vymazanie spôsobí stratu zachytených snímkov o pohybe. Snímky sa však budú dať zdieľať a archivovať.

Zoznam slave jednotiek (Monitoring centrum)

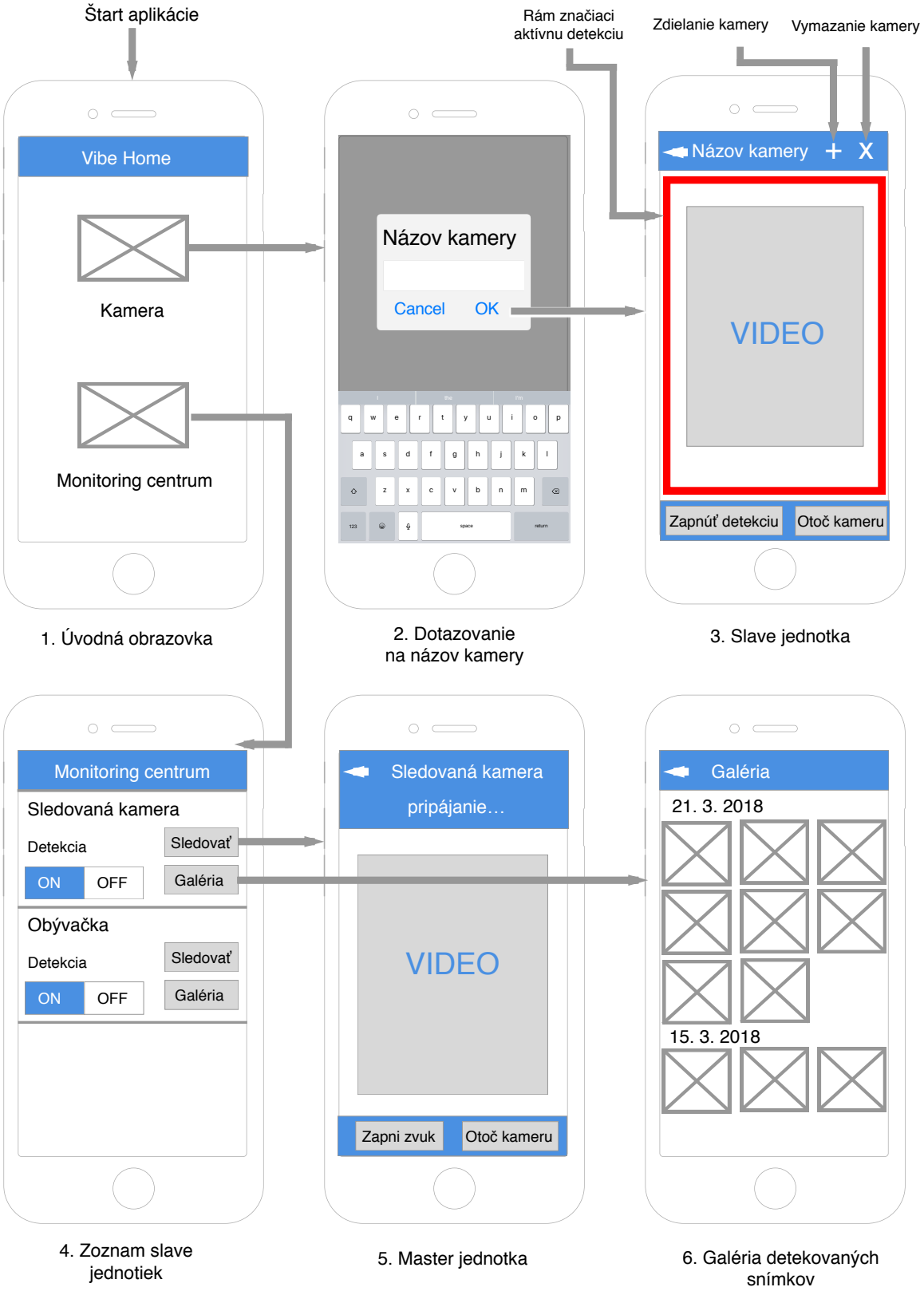
V zozname slave jednotiek budú zobrazené všetky slave jednotky a ich názvy zviazané s používateľským účtom v abecednom poradí, ale aj jednotky, ktoré boli s používateľom zdieľané prostredníctvom funkcionality zdieľania. Pri každej jednotke bude zobrazené, či je detekcia pohybu zapnutá alebo vypnutá a bude sa dať tento stav meniť. Používateľ si bude môcť zobraziť galériu zachytených snímkov pre danú jednotku. V neposlednom rade sa pri jednotke bude nachádzať tlačidlo *„Sledovať“*, ktoré inicializuje video spojenie s jednotkou.

Master jednotka

Obrazovka slúži pre prijímanie audio a video streamu. Vo vrchnej časti obrazovky sa nachádza názov jednotky, s ktorou aktuálne komunikujeme a pod ňou stav jednotky. Stav jednotky reflektuje stav pripojenia (napájanie, pripojený...), ale zobrazí aj stav nadmerného hluku, v prípade zvýšenia hladiny zvuku nad určitú úroveň. Používateľ môže následovne pomocou tlačidla povoliť prenos audia streamu alebo otočiť kameru tak, ako v prípade slave jednotky.

Galéria detekovaných snímkov pohybu

Pri zachytení pohybu nahrá aplikácia daný snímok do užívateľského účtu iCloud a užívateľovi, ktorý ma zapnutú detekciu pohybu príde oznámenie vo forme notifikácie, obsahujúcej snímok. Všetky tieto snímky sú spätne zobraziteľné v galérii danej jednotky a sú chronologicky zoradené za sebou, zoradené do sekcií podľa dní. Aplikácia nebude obsahovať žiadny mechanizmus manuálnej synchronizácie dát, ale všetko sa bude vďaka technológii CloudKit synchronizovať automaticky na pozadí. To znamená, že keď slave jednotka zachytí pohyb, používateľovi, ktorý má otvorenú galériu na master jednotke, pribudne na obrazovke nový snímok. Po zvolení snímku z galérie sa otvorí galéria do režimu plnej obrazovky (`fullscreen`). Snímok v režime fullscreen bude zobrazovať čas zachytenia a možnosť zdieľania snímku.



Obr. 5.5: Výsledný wireframe návrhu obrazoviek aplikácie a navigácia medzi nimi.

Kapitola 6

Realizácia

Implementácia aplikácie prebiehala vo vývojovom prostredí `Xcode`, ktorý je určený pre multiplatformový vývoj v rámci ekosystému firmy Apple. Implementácia vychádza z teoretických východísk skúmaných protokolov a algoritmov pre detekciu pohybu. Prvá časť sa zaoberá realizáciou modulu pre detekciu pohybu (`VibeDetection`) a ich iteráciami. Ďalšia časť sa venuje realizácii jadra na prenos audia a videa (`VibeHomeCore`). V nasledujúcej sekcii sú všetky tieto moduly spojené do výsledného aplikačného riešenia pod názvom `VibeHome`. V neposlednom rade sa na konci kapitoly popíšu metriky kódu pre všetky implementované moduly.

6.1 Realizácia modulu pre detekciu pohybu (VibeDetection)

V rámci práce boli implementované dva algoritmy. V prvotnej verzii implementácie prototypu bol implementovaný algoritmus `Running average`. Finálne riešenie však obsahuje vylepšenú verziu algoritmu ViBe spolu s aplikačnou logikou, ktorá slúži na reflektovanie zmien používateľského rozhrania a zasielanie zachytených snímkov.

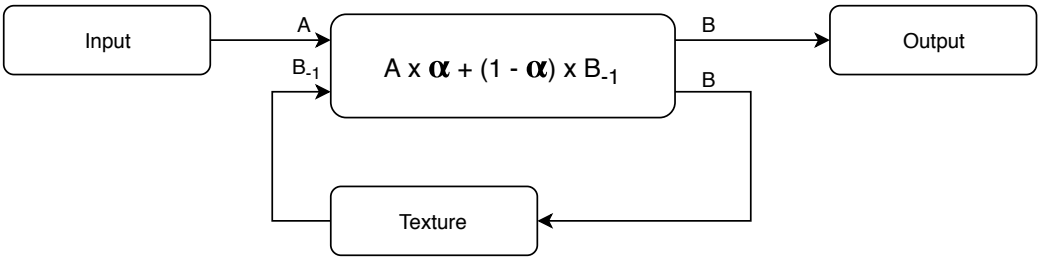
6.1.1 Implementácia prototypu algoritmom Running average

Pre prvý prototyp detekčného modulu bol zvolený algoritmus `Running average`. Dôvodom pre tento prístup bolo otestovanie efektivity algoritmu a overenie myšlienky prototypom aplikácie. Tento algoritmus bol zvolený aj z dôvodu jednoduchosti jeho implementácie. Fungovanie algoritmu je popísané v sekcii 4.1.

Algoritmus bol implementovaný v knižnici OpenGL ES 3.0[1]. Vstup shaderu tvorí textúra, ktorej zdroj je kamera zariadenia a spätná väzba `IIR filtra`. V prvej iterácii sa však na oba vstupy privádza rovnaká textúra. Blok filtra má nastavenú konštantu α na úroveň 40%, čo sa experimentami osvedčilo ako nastavenie filtra, ktoré podávalo najlepšie výsledky. Bloková schéma filtra je znázornená na obrázku 6.1. Výstup algoritmu reprezentuje euklidovskú vzdialenosť medzi výstupom z filtra so samotným vstupom, čo tvorí výslednú masku `Background modelu`.

Ďalším testovaním sa však ukázalo, že algoritmus nie je vhodný pre prípad užitia mojej aplikácie. Algoritmus veľa krát nezachytil pomalý pohyb pred kamerou. Pri zhoršenom osvetlení trpel algoritmus príliš veľkým šumom, ktorý sa musel redukovať manipuláciou

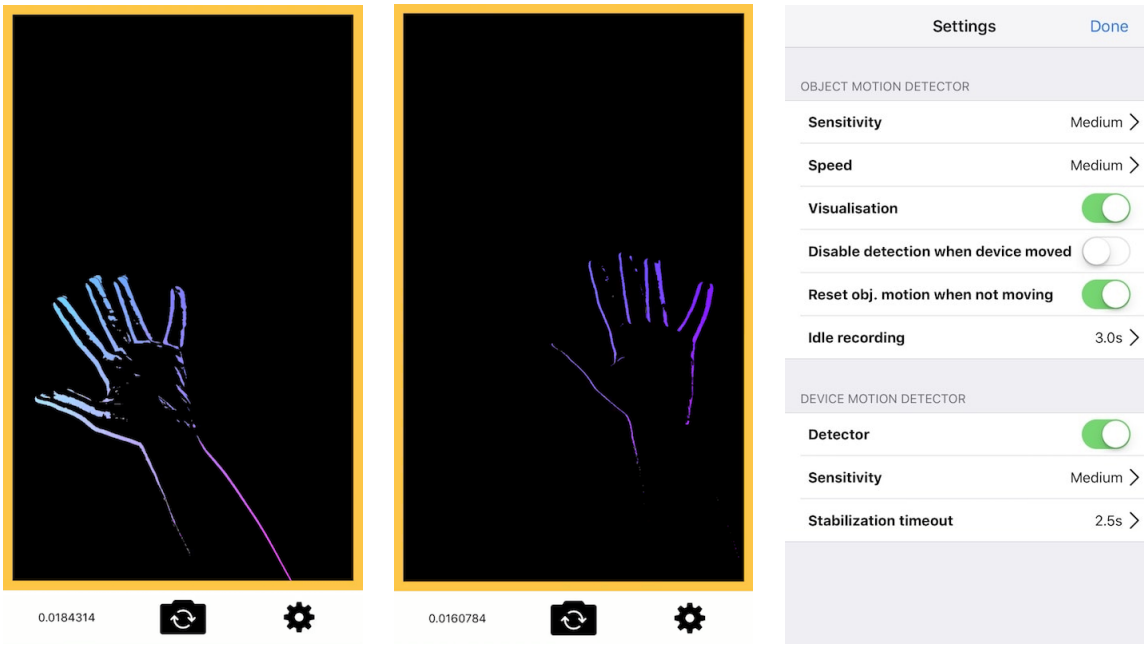
[1] Táto verzia je posledná vydaná verzia OpenGL ES pre platformu iOS z dôvodu presadzovania grafickej knižnice Metal.



Obr. 6.1: Bloková schéma low-pass filtra (IIR) znázorňujúca transformáciu vstupu na výstup.

konštanty α. To malo za následok zhoršenie kvality detekovania a vo výsledku zhoršenú použiteľnosť takéhoto riešenia.

Pre experimentálne účely bola vytvorená aplikácia, znázornená na obrázku 6.2, ktorá ponúkala rozšírené možnosti nastavenia algoritmu. Nad daným algoritmom vznikol vylepšený mechanizmus detekcie, ktorý posielal signály o pohybe v prípade, že intenzita pohybu neklesla pod určitú úroveň počas zvoleného časového intervalu.



(a) (b) (c)

Obr. 6.2: Ukážka implementovaného algoritmu Running average v experimentálnom prostredí testovacej aplikácie. Obrazovka a) a obrazovka b) znázorňujú pomalý priechod rukou nad prednou kamerou zariadenia a výslednú vizualizáciu masky. Obrazovka c) zobrazuje pomocné rozhranie pre efektívnejšie ladenie algoritmu a detekčnej logiky postavenej nad algoritmom Running average.

6.1.2 Implementácia algoritmu ViBe

Algoritmus ViBe bol vybraný pre jeho dobré vlastnosti a nenáročnosť na HW požiadavky. ViBe je popísaní v sekcii 4.2. V publikácii [28] predstavili Olivier Barnich et al. sekvenčnú

verziu algoritmu ViBe vo forme pseudokódu. Mojím cieľom bolo naprogramovať algoritmus ViBe pre spracovanie obrazu v rozlíšení 1280x720 (720p) so snímkovacou frekvenciou 30fps. Pre dosiahnutie tohoto cieľa je vhodné implementovať daný algoritmus na GPU mobilného zariadenia.

Riešenie pomocou OpenGL ES

Klasifikačný proces algoritmu si vyžaduje udržiavanie historických snímkov, z ktorých algoritmus číta s využitím určitej heuristiky a taktiež do nich aj zapisuje. Vo svete CPU toto nie je problém. Vytvorenie histórie snímkov je len otázka vytvorenia správnej štruktúry pre jej ukladanie. OpenGL ES má však vlastný reťazec spracovania (*pipeline*), s ktorým sme obmedzený len na zápis do výstupného `framebufferu`. Je však možné využiť techniku zvanú `Multiple Render Targets (MRT)`, kedy je možné k výstupnému `framebufferu` pripojiť prílohy a zapisovať do nich. Obmedzenie systému iOS ponúka možnosť vytvorenia štyroch `framebufferov` [22], kde formát, ktorý ponúka najväčšie úložisko o veľkosti je 128bitov (GL_RGBA32F), resp. 32bitov pre každú farebnú zložku. Toto obmedzenie sa zdalo byť dostačujúce, avšak nefungovalo, pretože som neskôr zistil, že maximálny objem dát je pri štyroch `framebufferoch` je taktiež 128bitov. Táto informácia však nebola spomenutá nikde v dokumentácii pre OpenGL ES, ale v článku pre osvedčené postupy (*Best practices*) pri práci s OpenGL ES, v poslednom odseku daného článku [23]. 128bitov na pixel však nie je dostatočná veľkosť pre uchovanie pixelov dvadsiatich historických snímkov s farebným RGB formátom. Jedinou možnosťou je uspokojiť sa s formátom v odtieňoch sivej (grayscale), ktorý nepodáva oveľa horšie výsledky. Implementácia tohoto prístupu by však vyžadovala kopírovať vždy všetky pixely historických snímkov do príloh `framebufferu`. Elegantnejším spôsobom by bolo využitie `Shader Storage Buffer Objektov (SSBO)`, ktoré ponúkajú možnosť atomických read/write operácií a prácu s veľkými zásobníkmi. Táto funkcionalita je však dostupná až v OpenGL ES vo verzii 3.1, ktorá na platforme iOS nie je dostupná [15].

Prvá funkčná verzia s využitím knižnice Metal

Dôvodom k prechodu na knižnicu Metal bola práve absencia `SSBO` pri aktuálne podporovanej verzii OpenGL ES. Metal ponúka od 10. verzie iOS-u funkcionalitu čítania a zápisu pre zásobníky (MTLBuffer) dostupné s shaderu, ktoré môžu mať maximálnu kapacitu 256MB [17]. Pre históriu dvadsiatich historických snímkov stačilo vytvoriť zásobník s rozmermi `(x: šírka textúry, y: výška textúry x 20)` a indexovať historické snímky po násobkoch výšky textúry.

Ďalšou výzvou bolo generovanie náhodných čísiel. Táto funkcionalita nie je v knižniciach Metal a OpenGL ES implementovaná. Bolo treba implementovať generátor náhodných čísiel. Prvým experimentom bola implementácia jednoduchého `LFSR`[2], ktorého seed[3] bol násobkom aktuálneho času predaného do shaderu ako vstupná premenná a aktuálnej polohy pixela. Zistil som, že generovanie pseudonáhodných čísiel je veľmi dôležité, pretože pri nevhodnom rozložení generátora je propagovanie okolitých pixelov algoritmu ViBe nerovnomerné alebo veľmi pomalé. Našiel som však špeciálnu knižnicu[4], založenú na článku [40], ktorý popisuje ako vytvoriť náhodný generátor na paralelných procesoroch, pre simulácie

[2] **Lineárny kongruentný generátor (LFSR)** – patrí medzi najjednoduchšie generátory pseudonáhodných čísiel.

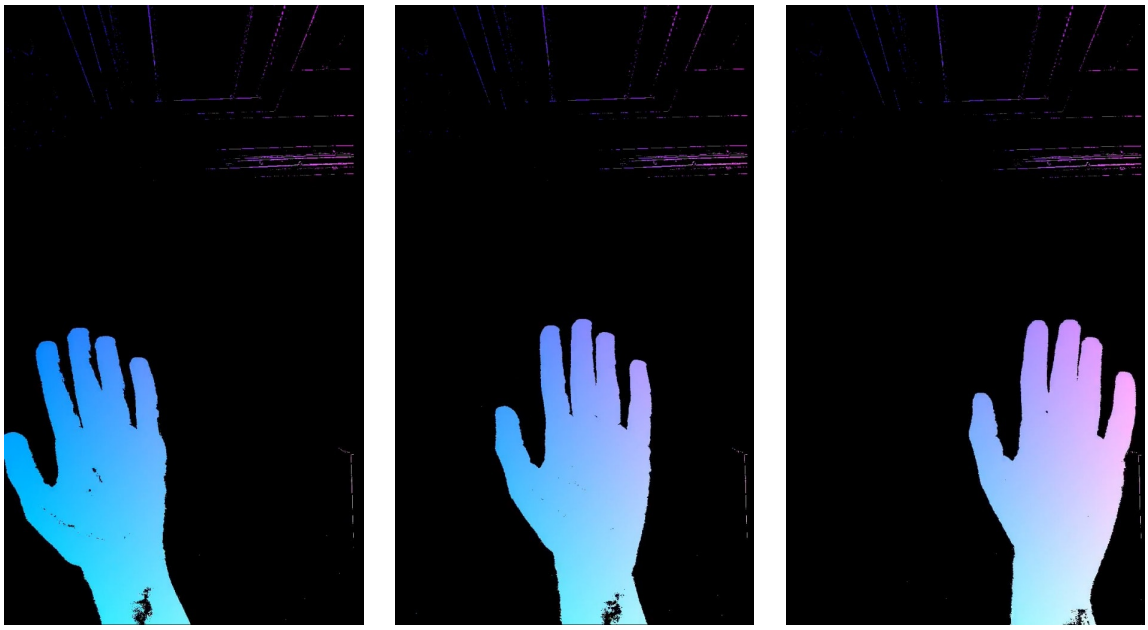
[3] **Seed** (náhodné semienko) – predstavuje počiatočné nastavenie generátoru pseudonáhodných čísiel.

[4] **Loki** – knižnica vhodná na generovanie náhodných čísiel na GPU: https://github.com/YoussefV/Loki/tree/master.

Monte Carlo. Algoritmus má teoretickú periódu 2^{121} a mal by byť postačujúci pre väčšinu výpočtov, vykonaných na GPU. Algoritmus však vyžaduje seed. Ten je generovaný na CPU pomocou BSD funkcie arc4random[5] a pričítaním aktuálnej pozície pixela pre každý ďalší snímok. Po vytvorení `Loki` objektu so seedom je možné volať metódu `next`, ktorá vygeneruje ďalšieho náhodného následníka.

Výslednému algoritmu je možné meniť parametre. Autor algoritmu však v článku predložil parametre, s ktorými zaznamenal najlepšie výsledky. Pri experimentoch som napríklad zmenšoval hodnotu ϕ, čo malo za následok rýchlejšie vstrebávanie tzv. „duchov“. Zostal som však pri pôvodných hodnotách a problém „duchov“ som vyriešil aplikačnou logikou postavenou nad algoritmom ViBe popísanou nižšie. Výsledok je vidieť na obrázku 6.3.

Ako je spomínané v sekcii 4.2, tak v ideálnom prípade by mal byť algoritmus schopný detekovať pohyb už od druhého snímku. Pre zjednodušenie som implementoval inicializáciu modelu tak, že pri zapnutí detekcie sa najprv naplní história dvadsiatich snímkov a proces pokračuje podľa pôvodného algoritmu. Pri snímkovacej frekvencii 30fps je história snímkov naplnená takmer okamžite.



(a) (b) (c)

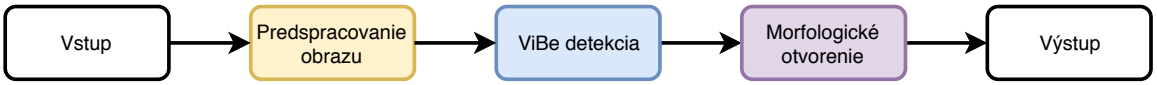
Obr. 6.3: Výsledok prvej iterácie implementovaného algoritmu ViBe. Obrazovky znázorňujú pomalý priechod rukou nad prednou kamerou zariadenia a výslednú vizualizáciu masky. Oproti prechádzajúcej implementácii už algoritmus nerozoznáva pri pomalom pohybe iba hrany, ale je vidieť plný objem pohybujúceho sa objektu.

Druhá iterácia implementácie algoritmu ViBe

Predchádzajúca iterácia implementácie algoritmu ViBe podávala slušné výsledky, avšak vo výslednom obraze sa vplyvom šumu, slabého osvetlenia alebo otrasov kamery občas objavovali vo výslednej maske artefakty. Našiel som však článok, v ktorom sa autor za-

[5] **Arc4random** – https://man.openbsd.org/arc4random.3

oberal vylepšeniami algoritmu ViBe [46]. Bloková schéma tohto vylepšenia je zobrazená na obrázku 6.4.



Obr. 6.4: Navrhované vylepšenie algoritmu ViBe z článku [46]. Predspracovanie obrazu vylepší výslednú kvalitu detekcie pri zlej kvalitu vstupu. Morfologické otvorenie napríklad pomôže pri eliminácii šumu a malých objektov.

Autor článku sa snažil vylepšiť algoritmus v prostredí tunelov, kde nepriaznivé svetelné podmienky a blikajúce svetlá občas spôsobovali, že malé oblasti alebo dokonca aj oblasti o veľkosti jedného pixela spôsobovali nesprávne detekovanie pohybu. Autor pracoval so snímkami v odtieňoch sivej a v prostredí v ktorom pracoval bolo vhodné snímky predspracovať využitím algoritmu ekvalizácie histogramu. Tým rozšíril histogram a umožnil tak lepšie rozlíšenie medzi objektami. V mojej práci však pracujem s formátom RGB a ekvalizácia nie je potrebná. Ďalšie navrhované vylepšenie, ktoré bolo použité v mojej práci bolo využitie **morfologického otvorenia**, vysvetleného v sekcii 4.3. Operácia morfologického otvorenia sa skladá z operácií erózie a dilatácie. `Erózia` môže eliminovať hraničné body, zmenšiť vnútorné hranice, ale hlavne eliminovať malé objekty alebo šum. `Dilatácia` môže zlúčiť blízke body a zaceliť tak otvory v objektoch.

Oproti knižnici OpenGL ES vo verzii 3.0 ponúka knižnica Metal využitie `compute shaderov`, popísaných v sekcii 4.4. S malými úpravami sa mi podarilo prepísať `fragment shader` na `compute shader`, čo napomohlo aj výslednému výkonu a pridalo možnosť používať predprogramované `performance shadery`. Metal taktiež obsahuje tzv. `metal performance shadery`, poskytujúce optimalizované implementácie aj pre morfologické operácie. Použil som shader pre morfologické otvorenie so zvolenou veľkosťou štrukturálneho elementu. Výsledok je vidieť na obrázku 6.5.

Segmentácia obrazu

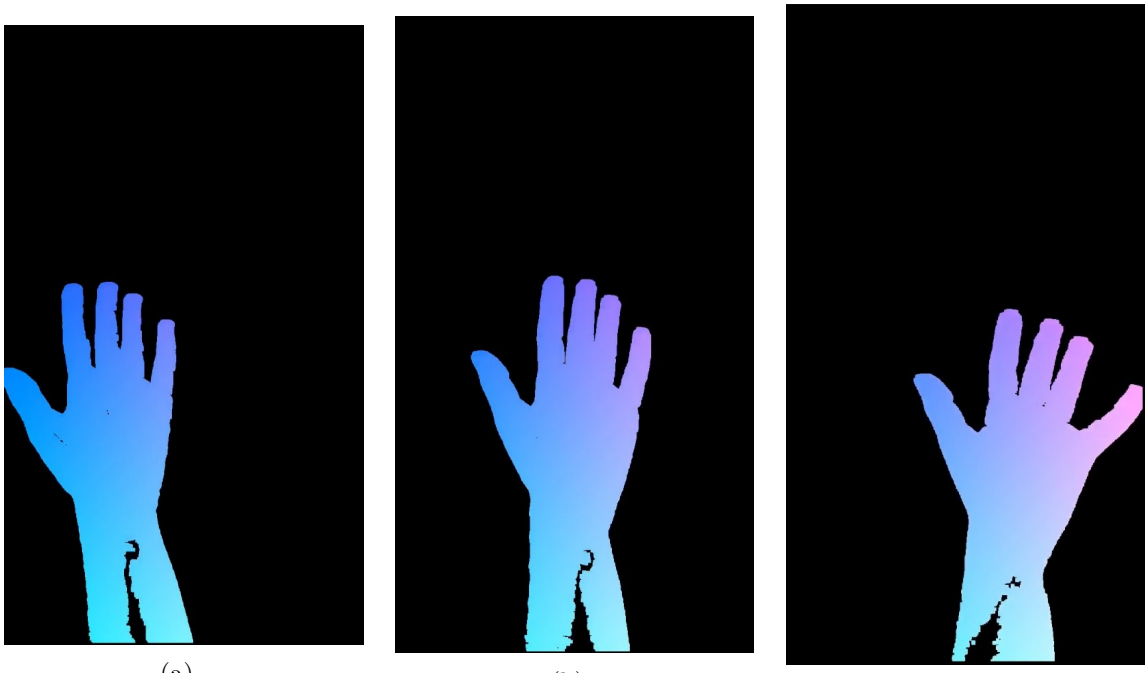
Mojim cieľom nebolo identifikovať všetky objekty na scéne a tak som v rámci zjednodušenia navrhol riešenie, ponúkajúce identifikáciu jediného objektu vo výslednom snímku. Obrázok 6.6 zobrazuje výsledok implementovanej segmentácie.

Pre každý pixel binárnej masky, tvoriacej výstup algoritmu ViBe, sa zoberie jeho hodnota (ktorá je buď 255 alebo 0) a pokiaľ je táto hodnota väčšia od nuly, do výslednej textúry vo formáte RGB sa uložia následovné hodnoty:

$$(r : x/(\text{šírka textúry}), g : y/(\text{výška textúry}), b : 255)$$

Operácia je znázornený na obrázku 6.7.a a ďalej je označovaná pod kódovým označením `Motioner`. Ďalšou operáciou je operácia redukcie, ktorej princíp je zobrazený na obrázku 6.7.b. Obe operácie vykonávajú vlastné `compute shadery`. Pre každý blok štyroch pixelov sa vypočíta priemer a uloží sa do menšej výslednej textúry. Redukcia sa vykonáva dokiaľ nie je výsledná textúra dostatočne malá na výpočet priemernej hodnoty na CPU. V mojom riešení je rozlíšenie nastavené na `1280x720` a redukcia je vykonaná 4-krát (80x45). Pomocou zmenšenej textúry sa na CPU vypočíta suma všetkých pixelov. Z výsledného sumarizovaného pixelu sa vypočíta stred pohybu a intenzita následovne:

$$x : \text{ P.red / (počet pixelov) / P.blue / (počet pixelov) } * \text{ (šírka textúry)}$$



(a) (b) (c)

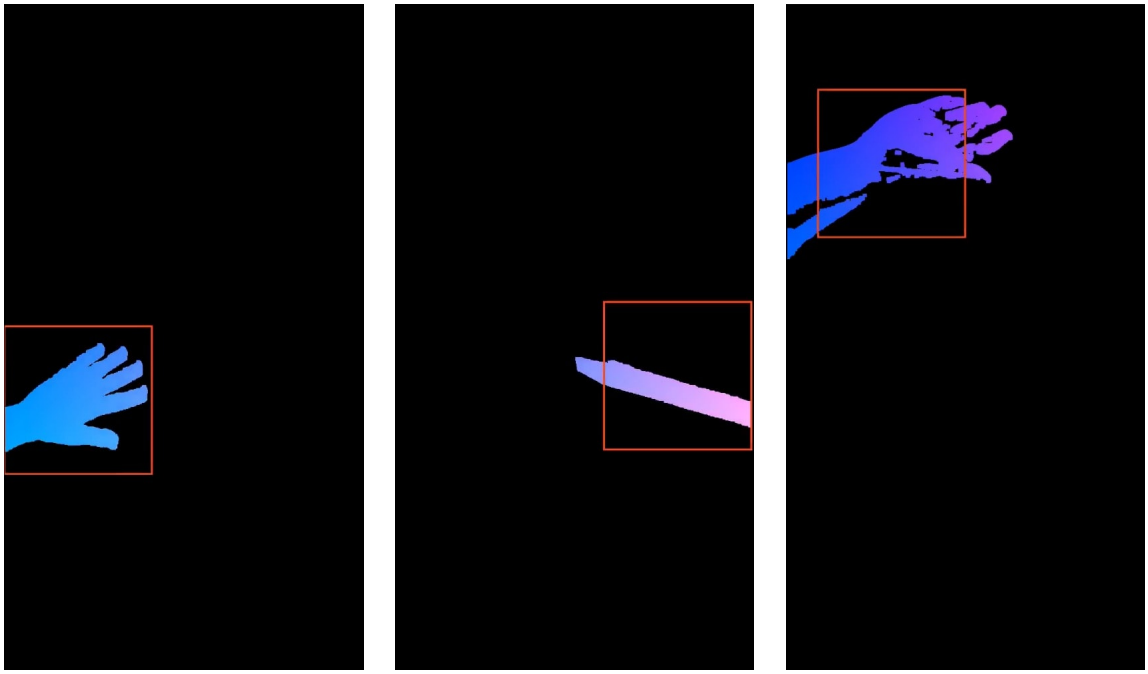
Obr. 6.5: Výsledok druhej iterácie, v ktorej bol prepísaný `fragment shader` na `compute shader` a taktiež bol algoritmus vylepšený o morfologické otvorené, ktoré prispelo k redukcii šumu a odstránenia malých objektov z výslednej masky.

$$y: \text{ P.green / (počet pixelov) / P.blue / (počet pixelov)} * \text{(výška textúry)}$$

$$\text{intenzita} : \text{P.blue / (počet pixelov)}$$

(a) (b)

Obr. 6.7: Obrázok a) znázorňuje proces rozšírenia pôvodnej binárnej masky na rozšírenú textúru obsahujúcu súradnice daného pixela pre výpočet výslednej hodnoty na CPU (`Motioner`). Obrázok b) znázorňuje principiálny spôsob redukcie snímku popísaný vyššie.



(a) (b) (c)

Obr. 6.6: Výsledok implementovanej segmentácie obrazu. Červený rámik znázorňuje pozíciu detekovaného objektu.

Popis výsledného reťazca pre detekciu pohybu

Výsledný reťazec pre detekciu je zložený z implementovaného ViBe algoritmu, ktorý je zapuzdrený do Metal enkóderu. Výstup je presmerovaný do `Motioner` enkóderu, ktorý predspracuje binárnu masku pre proces redukcie. Tento výstup je presmerovaný do enkóderu, ktorý ma na starosti redukciu textúry. Z redukovanej textúry sa vypočíta výsledná intenzita a poloha centra pohybu. Reťazec je znázornený na obrázku 6.8.

Obr. 6.8: Reťazec spracovania vstupného obrazu s využitím algoritmu ViBe.

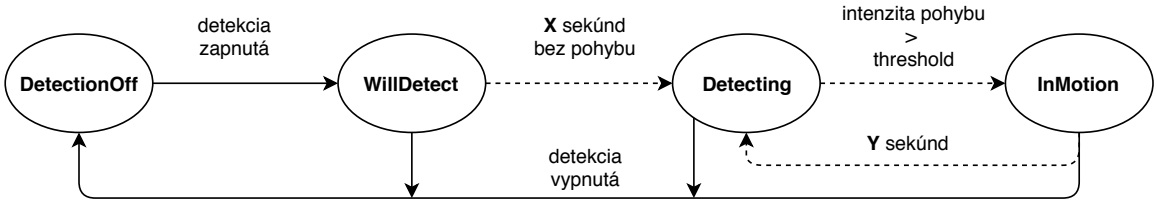
Algoritmy som sa snažil optimalizovať aby boli menej náročné na pamäť a výkon. Jednou z mnohých optimalizácií bola konverzia textúr na formát, ktorý je najvhodnejší pre ukladanie využívaných dát. Ďalšie optimalizácie zahŕňali využívanie dátového typu `half` namiesto dátového typu `int` alebo úprava veľkosti štrukturáleho elementu pri morfologickom otvorení. Pred optimalizáciou trvalo spracovanie jedného snímku **27ms**. Po optimalizácii klesla hodnota na **20.7ms**. Cieľom bolo dosiahnuť snímkovaciu frekvenciu 30fps, čo sa podarilo (48fps).

Vylepšenie aplikačnej logiky

Reťazec pre detekciu pohybu dáva na výstup intenzitu a súradnice centra pohybu. Tento proces som chcel obohatiť o stavy, na ktoré by mohla reagovať aplikácia a v prípade potreby

odosielať snímok s detekovaným pohybom alebo upozorniť používateľa o zvýšenom pohybe. Definoval som následovné stavy a ich význam (obrázok 6.9):

- **DetectionOff** (detekcia vypnutá) – Užívateľ detekciu neaktivoval.
- **WillDetect** (bude zapnutá detekcia) – Užívateľ aktivoval detekciu a do stavu detekovania sa dostane v prípade, že X sekúnd algoritmus nezaznamená žiadny pohyb. Jedná sa teda o proces stabilizácie.
- **Detecting** (detekuje) – Algoritmus je v stave aktívnej detekcie pohybu. V prípade, že algoritmus zaznamená intenzitu, prekračujúcu zvolený prah, prechádza sa do stavu `InMotion`.
- **InMotion** (v pohybe) – V tomto stave odošle aplikácia aktuálny snímok z kamery všetkým master jednotkám. Po X sekundách sa prejde do stavu `Detecting`.



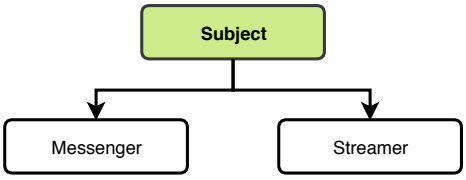
Obr. 6.9: Vylepšený mechanizmus detekcie vhodný pre použitie vo výslednej aplikácii. Plné hrany sú synchrónne, prerušované sú asynchrónne.

6.2 Realizácia modulu pre prenos audia a videa (VibeHomeCore)

Nasledujúca sekcia obsahuje súhrn najhlavnejších tried a ich popis spolu so zodpovednosťami daných tried. Taktiež obsahuje stručný popis dôležitých objektov, ktoré sú využité pri implementácii.

Subject

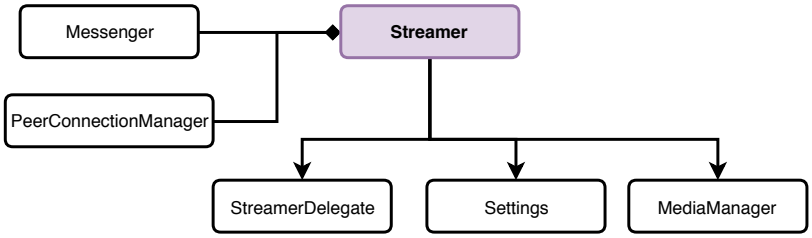
`Subject` reprezentuje jednotku, ktorá vysiela mediálny stream (`MasterSubject`) alebo jednotku, ktorá mediálny stream prijíma (`SlaveSubject`). Jej zodpovednosť je manažovať životný cyklus monitorovania. Používateľské rozhranie operuje len prostredníctvom tejto triedy. Súčasťou Subjectu je trieda `Streamer`, predstavujúca objekt, ktorý slúži na manažovanie RTC spojenia a vysielanie mediálnych dát. Trieda `Messenger` tvorí komunikačnú vrstvu, ktorá má na starosti kontrolu stavu slave jednotiek a taktiež udržiavanie týchto stavov. `Subject` poskytuje rozhranie pre príjem spracovaných a sformátovaných dát na najvyššej úrovni.



Obr. 6.10: Diagram triedy Subject a jej závislosti.

Streamer

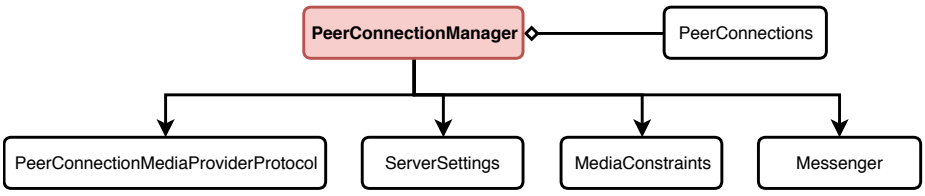
Trieda `Streamer` slúži ako tenký obal nad triedou `PeerConnectionManager`, ktorej zodpovednosť je udržiavanie audio a video prenosu od jedného účastníka k druhému. Rieši všetky časové limity a snaží sa udržať spojenie otvorené. Používa triedu `Messenger` na získanie všetkých aktuálnych stavov spojenia a informuje o nich ostatné `Subjecty`. `Streamer` notifikuje o svojich zmenách prostredníctvom delegátneho protokolu `StreamerDelegate`. Stopy pre audio a video sú brané z triedy `MediaManager`. Ak počas zahajovania audio relácie nie je dostupná audio stopa, `Streamer` zaháji vytvorenie nového RTC spojenia aby stopu získal a zaháji prenos audio streamu prostredníctvom triedy `PeerConnectionManager`. `Streamer` vyžaduje pre konfiguráciu triedy `Messenger` nastavenie MQTT serveru.



Obr. 6.11: Diagram triedy Streamer a jej závislosti.

PeerConnectionManager

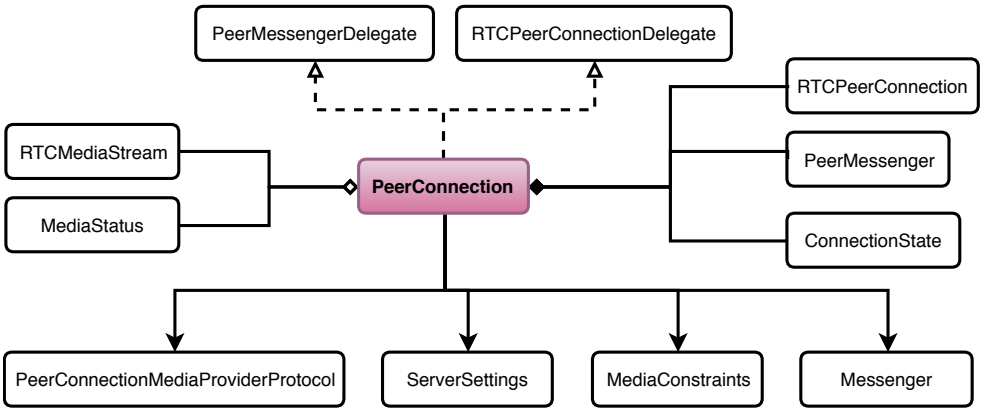
Trieda `PeerConnectionManager` riadi viacero spojení (*PeerConnections*). Udržiava si taktiež viacero mediálnych audio a video streamov a dokáže vysielať mediálny stream viacerým observerom. Pre každú instanciu triedy `PeerConnection` si udržiava separátny `Messenger` s určitým nastavením (subjectId, deviceId). Má na starosti prijímanie alebo odmietanie nových požiadaviek na spojenie. `PeerConnectionManager` nastavuje aktuálne RTC audio a video stavy prostredníctvom triedy `MediaManager`.



Obr. 6.12: Diagram triedy PeerConnectionManager a jej závislosti.

PeerConnection

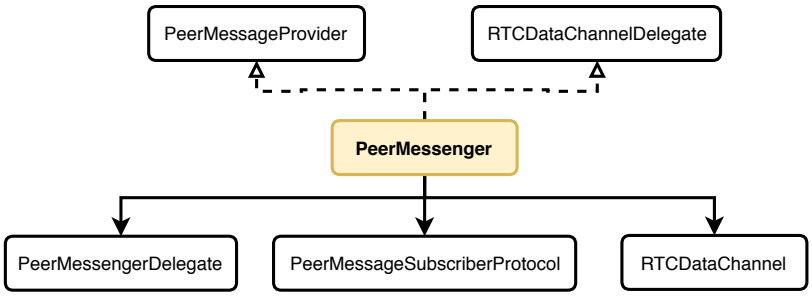
Trieda `PeerConnection` používa triedu `RTCPeerConnection` (popísanú v sekcii 3.1) na vytváranie RTC spojení. Vykonáva tzv. `offer dance`, popísaný na obrázku 3.8. V prípade chyby nastaví svoj stav na `failed` a trieda `PeerConnectionManager` na ňu zareaguje. Po tom, čo je trieda `PeerConnection` inicializovaná a SDP správy si klienti medzi sebou vymenili, je možné pridávať nové streamy. Streamy je možné zároveň aj odoberať.



Obr. 6.13: Diagram triedy PeerConnection a jej závislosti.

PeerMessenger

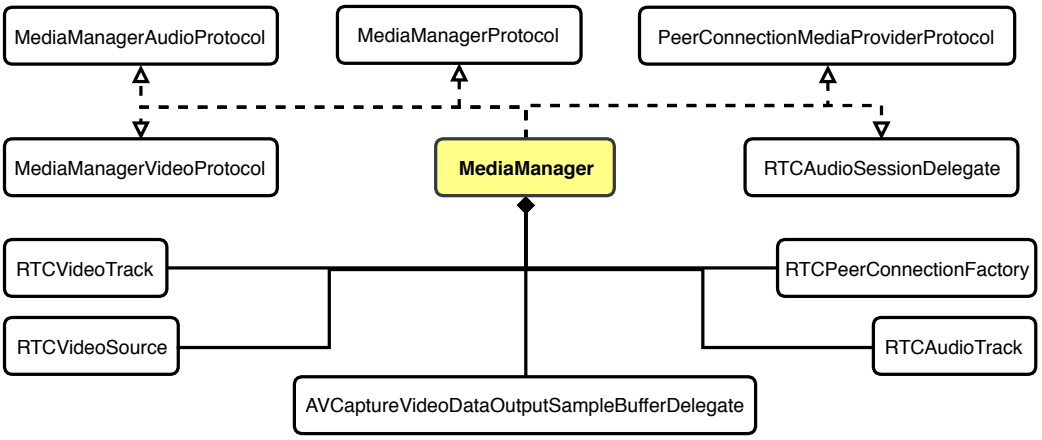
Trieda `PeerMessenger` je obsiahnutá v triede `PeerConnection`. Využíva sa pre zasielanie RTC dát prostredníctvom dátového kanálu (*peer-to-peer*). Dáta, tečúce cez signalizačný MQTT protokol sú teda zdvojované a tečú dvomi kanálmi. To sa hodí v prípade výpadku serveru MQTT. `PeerMessenger` poskytuje podobné aplikačné rozhranie pre prijímanie a odosielanie správ ako `Messenger`.



Obr. 6.14: Diagram triedy PeerMessenger a jej závislosti.

MediaManager

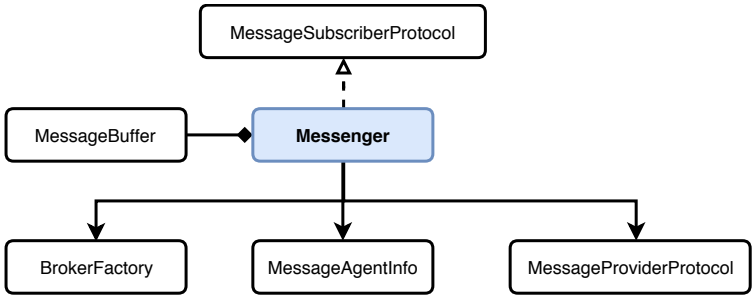
`MediaManager` využíva triedy `RTCAudioSession` a `AVCaptureSession` pre správu audio a video streamov. RTC frameworku poskytuje výlučný prístup a umožňuje tak meniť nastavenia zvukovej relácie. Taktiež rieši prácu s kamerou, vypínanie a zapínanie svetla, nastavenia parametrov kamery a podobne. Ďalšou zodpovednosťou tejto triedy je kontrola, či používateľ dovolil aplikácii využívať kameru a mikrofón.



Obr. 6.15: Diagram triedy MediaManager a jej závislosti.

Messenger

`Messenger` poskytuje triede `Subject` rozhranie pre zasielanie správ s využitím `MqttManageru`. `Subject` nepozná implementačné detaily zasielania správ a je mu poskytnutá abstrakcia pre zasielanie. Toto riešenie umožňuje v prípade zmeny signalizačného protokolu jednoducho implementovať komunikačné rozhranie a nemusí sa meniť implementácia `Subjectu`. `Messenger` sa prihlási k odoberaniu určitých typov správ a ak príjme správu, notifikuje `Subject` prostredníctvom delegátnej metódy. `Messenger` používa triedu `MessageBroker` pre zistenie typov prichádzajúcich správ a ich transformáciu z `MessageProtocolu` na konkrétny typ správy.



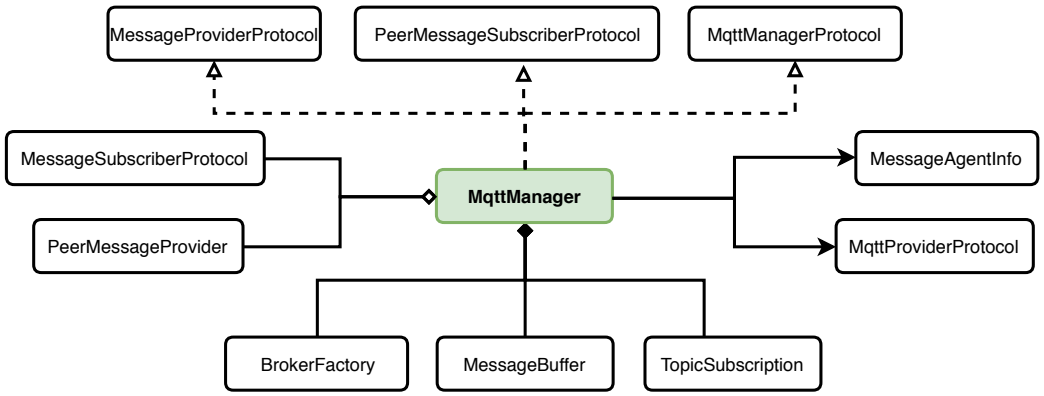
Obr. 6.16: Diagram triedy Messenger a jej závislosti.

MqttManager

Trieda `MqttManager` rieši všetku komunikáciu s MQTT serverom prostredníctvom triedy `MqttProvider` (tenká vrstva nad knižnicou `CocoaMQTT`). `MqttManager` udržiava zoznam `Messengerov` a `topicov`, na ktoré je prihlásený (`TopicSubscription`). `MqttManager` využíva iba jedno fyzické spojenie s MQTT serverom a v prípade vytvorenia nových `Messengerov` imituje jeho správanie – na novovytvorený `Messenger` sú odoslané všetky **retained** správy na ktoré je nový `Messenger` prihlásený. `MqttManager` sa fyzicky odhlási z odberu pre `topic` v momente, keď sa z neho odhlási posledný `Messenger`. V prípade straty spojenia sa trieda snaží spojenie nadviazať znovu. Ak sa to podarí, trieda sa znovu prihlási na všetky odbery, na ktoré bola pred výpadkom prihlásená.

`MqttManager` prevádza triedy typu `Message` na reťazce, ktoré sa posielajú prostredníctvom MQTT a taktiež prichádzajúce správy prevádza naspäť.

`MqttManager` obsahuje `MessageBuffer`, slúžiaci ako zásobník pre filtrovanie duplicitných správ, ktoré sa porovnávajú s poslednými desiatimi správami pre každý `topic` a iba unikátne správy môžu cez neho prejsť. V prípade využívania `PeerMessengeru` sú správy duplikované vždy, pretože sa posielajú dvoma cestami (MQTT a RTC dátový kanál).

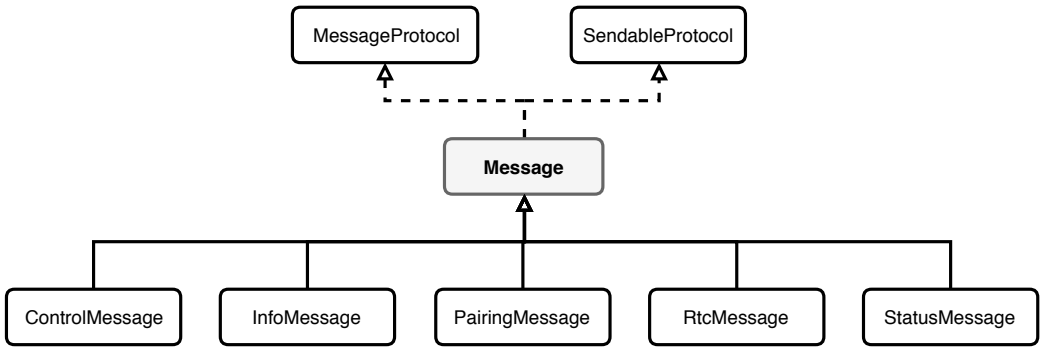


Obr. 6.17: Diagram triedy MqttManager a jej závislosti.

Message

`Message` reprezentuje dátový model pre akúkoľvek správu zasielanú cez inernetovú sieť. Zahŕňa päť hlavných typov:

- **ControlMessage** – Zahŕňa všetky typy správ pre ovládanie zariadenia, ako napríklad otočenie kamery.
- **InfoMessage** – Správa spojená s dostupnosťou **Subjectu**.
- **StatusMessage** – Správa informujúca o nejakom stave jednotky (úroveň hluku).
- **RtcMessage** – Správa spojená s RTC komunikáciou (ICE, SDP...).
- **PairingMessage** – Správa spojená s informáciami ohľadom párovania jednotiek.



Obr. 6.18: Diagram triedy Message a jej závislosti.

6.3 Realizácia výslednej aplikácie (VibeHome)

Realizácia spočíva v spojení modulu pre prenos audia a videa a taktiež detekčného modulu implementujúceho algoritmus ViBe. Samotná aplikácia však vyžaduje synchronizáciu dát zariadení a zachytených snímkov. Taktiež je potrebné pri detekovaní pohybu upozorniť používateľa notifikáciou, obsahujúcou daný snímok. V rámci synchronizácie dát je potrebné zabezpečiť, aby sa lokálne dáta uložili na zariadení. V tejto sekcii je ďalej popísaný vývoj používateľského rozhrania.

6.3.1 Prvý prototyp synchronizácie zariadení

Prvý prototyp aplikácie využíval technológiu Firebase[6] ako implementáciu serverového riešenia pre správu užívateľov a párovanie zariadení. Firebase využíva pre perzistenciu a spracovanie dát na servery NoSQL[7] vo formáte `JSON`. Príklad výslednej databázovej štruktúry je znázornený vo výpise 6.1. Detailným rozborom tohoto riešenia (dôvody pre deoptimalizáciu...) sa nezaoberám, nakoľko výsledné riešenie technológiu Firebase nepoužíva. Pre názorný účel však postačí nasledovný popis.

Pre spárovanie slave jednotky s master jednotkou som navrhol riešenie, kde server pre master jednotku vygeneruje náhodné štvormiestne číslo. Toto číslo sa zobrazí na master jednotke a na slave jednotke ho používateľ vyplní. Spolu s vygenerovaným kódom sa pre master jednotku vygeneruje číslo skupiny, v ktorej budú zariadenia spolu komunikovať signalizačným protokolom. Detaily ohľadom použitia `subjectId` a `groupId` sú vysvetlené v sekcií 6.2. Kód sa zobrazuje na master jednotke a zadáva na slave jednotke. Je to z dôvodu, že ak by chcel útočník na master jednotke sledovať zariadenie na ktoré útočí, stačilo by v správny okamžik uhádnuť štvormiestny kód a dostal by sa k mediálnemu streamu slave jednotky. Zodpovednosť akceptácie vygenerovaného kódu je však na slave jednotke. Pri zhode vygenerovaného kódu sa master jednotke zašle unikátny identifikátor (subjectId) danej slave jednotky.

V neskoršej fáze projektu, kedy pribudla potreba zasielania a ukladania zachytených snímkov, som usúdil, že Firebase nie je najhodnejšie zvolenou technológiou pre prípad použitia mojej aplikácie. Dozvedel som sa, že lepším riešením bude využiť technológiu CloudKit popísanú v sekcii 2.4. Dôvody pre prechod na CloudKit:

- **Potreba registrácie pri službe Firebase.** Pre využívanie používateľských účtov je pri službe Firebase vyžadovaná registrácia. Ekosystém Applu však ponúka možnosť využitia iCloud účtu, ktorý je potrebný aj na sťahovanie aplikácií z AppStoru. Ak teda používateľ stiahol moju aplikáciu z AppStoru, je vlastníkom iCloud účtu a nemusí sa do aplikácie prihlasovať. Všetky jeho dáta sú automaticky stiahnuté z jeho účtu.
- **Firebase spoplatňuje dátové úložisko nad rámec bezplatného balíčku.** Pri zvýšenom dátovom toku alebo prekročení určitého objemu dát je preto potrebné platiť. V prípade iCloudu s využitím frameworku CloudKit sú používateľské dáta uložené na účte používateľa. To môže priniesť používateľovi pocit bezpečia z používania aplikácie, ktorá používa synchronizáciu so servermi Applu. Ak používateľ zaplní svoje úložisko, môže si ho za mesačný poplatok u Applu rozšíriť.

[6] **Firebase** – https://firebase.google.com/

[7] **NoSQL** – je databázový koncept, v ktorom dátové úložisko a spracovanie dát používajú iné prostriedky než tabuľkové schémy tratičnej relačnej databázy.

- **Údržba serverového riešenia pri službe Firebase.** Pri zvýšenom objeme dát alebo zaplnení bezplatného úložiska dátami všetkých používateľov je treba za službu Firebase platiť. Taktiež je treba dávať pozor na stav serverov alebo udržiavať separátny serverový kód a riešiť prípadné bezpečnostné problémy spojené s implementáciou takéhoto riešenia. V prípade CloudKitu je táto zodpovednosť na firme Apple.
- **Jednoduchá implementácia zdieľania pri CloudKite.** CloudKit ponúka hotové riešenie zdieľania používateľských dát, popísané v sekcii 2.4. V prípad služby Firebase by bola nutná vlastná implementácia takéhoto riešenia spojená s ďalšou jeho údržbou.

```
1  ..."groups": {
2          "someKeyId−231212": {
3                  "subjId−123": {
4                          "name": "Living room",
5                          "state": 3}, // e.g. pripajanie
6                  "subjId−567": {
7                          "name": "Kitchen",
8                          "groupId": "someKeyId−231212",
9                          "state": 2} } // e.g. jednotka aktivna
10 },
11 "subjects": {"subjId−123": "someKeyId−231212",
12             "subjId−567": "someKeyId−231212"},
13 "codes": {"0983": {"groupId": "someKeyId−231212",
14                    "subjectId": "subjId−123",
15                    "timestamp": 123124342342} }...
```

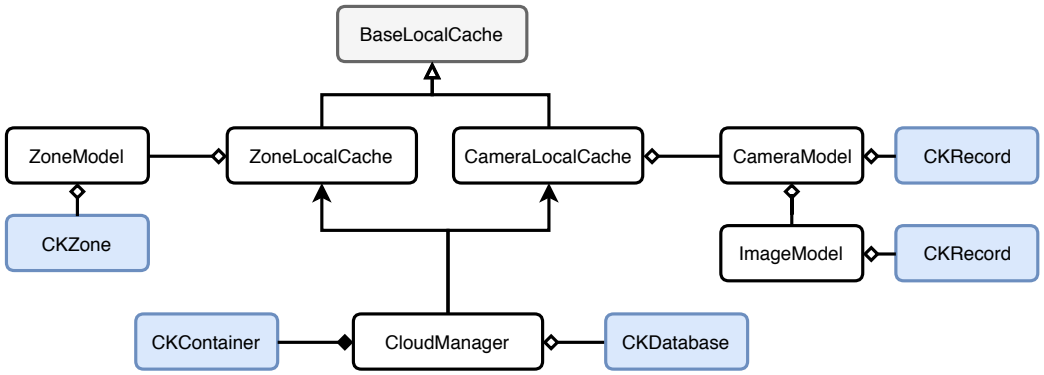
Výpis 6.1: Príklad databázovej štruktúry serverového riešenia implementovaného prototypu s využitím Firebase. Pre názorný účel je štruktúra zjednodušená.

6.3.2 Synchronizácia dát pomocou CloudKitu

Na obrázku 6.19 je znázornený diagram tried a ich závislostí. Predstavená štruktúra zabezpečuje synchronizáciu dát s používateľským iCloudom. Dáta sú uložené vo forme lokálnej `cache`. Hlavnou triedou, ktorá zastrešuje prístup k lokálnym `cache` objektom, je trieda `CloudManager`, ktorá je jedináčikom (*singleton*[8]). a pri spustení aplikácie sa musí nastaviť. Táto trieda si udržiava aj aktuálny zoznam databáz (privátnu a zdieľané). Základnou triedou je trieda `BaseLocalCache`, z ktorej dedí `ZoneLocalCache` a `CameraLocalCache` a ponúka rozhranie pre prihlasovanie sa na databázové odbery (*subscriptions*).

Prihlasovanie sa na odber pre špecifické udalosti, akými je napríklad prijímanie zdieľaných položiek alebo prijímanie zmenených a upravených položiek, má na starosti trieda `ZoneLocalCache`. Tá si udržiava aktualizačné tokeny (popísané na konci sekcie 2.4) a samotné zóny. Používateľovi sa po prvom spustení vytvorí v jeho privátnej databáze vlastná zóna. Ostatné zóny patria do zdieľanej databázy. Pri zmene práv alebo objektov v online CloudKit databáze si trieda sama aktualizuje svoje lokálne dáta. Rovnako je riešená nie len synchronizácia databáz alebo zón, ale aj zariadení a ich príslušných dát. Túto zodpovednosť má na starosti trieda `CameraLocalCache`, ktorá si udržiava najaktuálnejšiu kópiu zariadení, ich nastavení a taktiež zachytených snímkov. Väzba CloudKit záznamov je riešená kompozíciou, kvôli tomu, že CloudKit záznamy nemôžu využívať dedičnosť [1]. Pri prvej synchronizácii nie je aktualizačný token známy, takže pri prihlásení na odber sa vyžiadajú všetky dáta.

[8] **Singleton** – je názov návrhového vzoru, ktorý sa využíva v prípade, kedy je potrebná iba jedna instancia objektu počas behu programu.

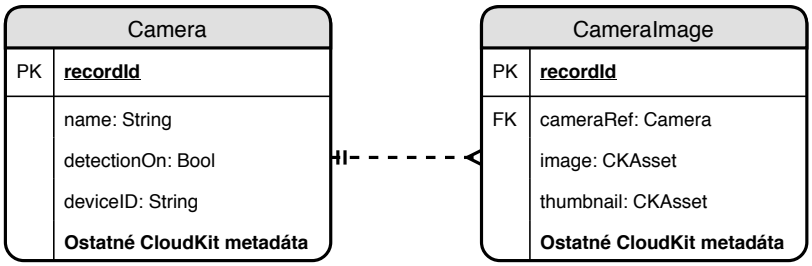


Obr. 6.19: Diagram tried a ich závislostí znázorňujúci riešenie synchronizácie dát pomocou CloudKitu. Triedy zvýraznené modrou farbou pochádzajú z frameworku CloudKit a ich popis sa nachádza v teoretických východiskách sekcie 2.4.

Obe triedy `ZoneLocalCache` a `CameraLocalCache` aktualizujú pomocou funkcionality odberov dáta lokálnej `cache`. Pre aktualizáciu používateľského rozhrania sa používa trieda s názvom `NotificationCenter`, z iOS frameworku `Foundation`. `NotificationCenter` poskytuje mechanizmus šírenia informácií (`broadcast`) naprieč celou aplikáciou. Na miestach, kde je treba používateľovi zobrazovať aktuálne dáta, sa vytvorí `observer`, registrovaný na triedu `NotificationCenter`, ktorá je jedináčik.

Dôležité dáta, ktoré konzumuje používateľ sa nachádzajú v triedach `Camera` a `CameraImage`. Tieto dáta spracováva a obsluhuje spomínaná trieda `CameraLocalCache`. Dáta nachádzajúce sa v týchto triedach sú znázornené v schéme na obrázku 6.20. Pri modifikácii záznamu `Camera` môže vzniknúť konflikt, spojený s oneskorenou synchronizáciou dát. Pri konflikte sa na strane klienta vyskytne chyba, obsahujúca pôvodný záznam, ktorý mal používateľ k dispozícii pri jeho manipulácii a zároveň nový záznam, pochádzajúci zo servera. Riešením konfliktu je výber záznamu zo servera.

Pri vypnutej aplikácii a vzdialenej zmene dát dôjde k notifikovaniu aplikácie formou `silent`[9] notifikácie, ktorá spustí synchronizáciu dát.



Obr. 6.20: Schéma CloudKit databáze. Okrem definovaných dátových položiek sa v databáze nachádzajú aj metadáta CloudKitu, obsahujúce záznamy ako identifikátor autora, dátum vytvorenia, modifikácie, autor poslednej modifikácie a ďalšie...

[9]**Silent notifikácia** – je notifikácia, ktorá sa vizuálne nezobrazí používateľovi, avšak zobudí aplikáciu pre možnosť ďalšieho spracovania.

6.3.3 Notifikácie

CloudKit pri prihlasovaní na odber zmien v databáze ponúka možnosť špecifikovať notifikáciu so statickým obsahom, ktorá príde v prípade úpravy, vymazaní alebo pridaní objektu na ktorý sa prihlasujeme. Nie je možné sa odkazovať na predom neznáme dáta, len na typ prichádzajúcich dát. V prípade riešenej aplikácie je potrebné v notifikácii zobrazovať detekovaný snímok pohybu.

Tento problém sa dá realizovať vytvorením aplikačného rozšírenia (*extension*), ktoré funguje nezávisle na aplikácii a spracováva všetky notifikácie, obsahujúce dáta pre zobrazenie. Rozšírenie bolo implementované vytvorením nového `targetu`[10], ktorý obsahoval jedinú triedu `NotificationService`, ktorá dedí z triedy `UNNotificationServiceExtension`. Táto trieda musela získať identifikátor pridaného snímku. Tento identifikátor slúži na vytvorenie dotazu, ktorý z CloudKit databáze stiahne záznam o kamere, pre uvedenie názvu kamery v notifikácii. Ďalším dotazom sa stiahne zachytený snímok a priloží sa k notifikácii, ktorá sa následne používateľovi zobrazí. Pri klepnutí na notifikáciu bola implementovaná navigácia do galérie daného zariadenia.

6.3.4 Perzistentné úložisko

Predstavené riešenie lokálnej `cache` je vyhovujúce do doby, dokiaľ nie je aplikácia ukončená užívateľom alebo systémom z dôvodu uvoľňovania prostriedkov pre ďalšiu aplikáciu. Preto som sa rozhodol implementovať perzistentné úložisko. Z výkonnostných dôvodov som siahol po knižnici `Realm`. Dôvody prečo je táto technológia pre perzistenciu dát rýchlejšia, než natívne riešenie postavené na SQLite, je popísané v článku [49].

Naivným prístupom som modifikoval modely zobrazené na obrázku 6.19. Pridal som do nich atribúty, do ktorých sa triedy spojené s CloudKitom serializovali, aby boli dáta kompatibilné s databázou Realmu. Problém nastal pri snímkoch. Dáta, obsiahnuté v triede `CKAsset` obsahujú len odkaz na dočasné úložisko, nie však samotné dáta snímku. Preto bolo potrebné dáta z odkazu načítať a separátne spolu s metadátami `CKAssetu` serializovať. V rámci ušetrenia dátového úložiska na zariadení som do perzistentnej databáze ukladal len náhľady snímkov. Snímok v plnom rozlíšení sa sťahoval na vyžiadanie pri prezeraní galérie.

Ďalším problémom, s ktorým som sa potýkal, bola práca s objektami databáze, prebiehajúca medzi vláknami aplikácie. `Realm` vyžaduje, aby k objektu načítanému z databáze alebo uloženému do databáze bol prístup iba z jedného vlákna. Operácia synchronizácie dát však prebieha na inom vlákne ako v prípade zobrazovania dát (hlavné vlákno) alebo spracovania snímku. Pôvodný naivný prístup k databázovým objektom musel byť reimplementovaný z prístupu predávania referencie objektu, na predávanie identifikátoru objektu. Vlákno, ktoré potrebuje prístup k objektom pre ďalšie spracovanie, dostalo namiesto referencie identifikátor objektu a podľa neho si z databáze daný objekt znovu vytiahlo.

6.3.5 Používateľské rozhranie (UI)

Prvé prototypy riešenia slúžili len na overenie technického riešenia pre prenos audia a videa. Ďalšia aplikácia a jej UI vzniklo pre potreby testovania a ladenia detekčných algoritmov. Výsledný design aplikácie priložený v prílohe B reflektuje grafický návrh z obrázku 5.5 sekcie 5.4. UI bolo implementované programovo čiastočne s využitím knižnice `SnapKit` popísanej v sekcii 2.5, ale taktiež pomocou `Storyboardov` popísaných v sekcii 2.3.

[10] **Target** – špecifikuje nový produkt aplikácie, jeho build nastavenia, triedy a zdroje potrebné pre vytvorenie takéhoto targetu. Aplikácia môže obsahovať viacero targetov (testy, Apple Watch, produkčnú verziu...).

6.3.6 Integrácia modulu na prenos audia a videa (VibeHomeCore)

Integrácia modulu na prenos audia a videa je veľmi jednoduchá. Spočíva v napojení používateľského rozhrania na rozhranie triedy `MasterSubject` alebo triedy `SlaveSubject` a ich inicializovaní. Následne sa používateľské rozhranie reprezentované triedou `UIViewController` príhlási na odbery zmien pre stavy, ktoré chce reflektovať používateľovi. V prípade ovládacích prvkov stačí zavolať príslušnú metódu triedy `Subject`. Je však treba reagovať na životný cyklus aplikácie a triedy `UIViewController` vysvetlenej na obrázku 2.2 a napríklad pri volaní metódy `viewDidDisappear()` ukončiť perzistentné spojenie.

6.3.7 Integrácia modulu na detekciu pohybu (VibeDetection)

Knižnica WebRTC, ktorá vo vnútri frameworku získava prístup ku kamere, neposkytuje žiadny spôsob ako sa dostať k surovým dátam senzoru zariadenia. Pre potreby integrácie modulu `VibeDetection` do danej aplikácie bolo treba modifikovať zdrojové kódy knižnice WebRTC. Pripravená verzia knižnice je k dispozícii prostredníctvom `Cocoapods`[11], avšak ak je treba knižnicu modifikovať, je potrebné získať zdrojové kódy[12], ktoré sú rozsiahle (∼4GB) a pomocou build skriptov v projekte vygenerovať výsledný framework, ktorý sa manuálne prilinkuje k projektu. Knižnica bola upravená a poskytuje možnosť priradenia delegátnej triedy, ktoré obsahujú metódy poskytujúce `framebuffer` zariadenia kamery. Upravená knižnica WebRTC bola prilinkovaná k modulu `VibeHomeCore` a surové dáta senzoru boli sprístupnené pomocou rozhrania triedy `MediaManager`. V aplikácii sa detekcia zapínala prostredníctvom rozhrania triedy `EnhancedVibeDetector`, ktorej stavy boli popísané na obrázku 6.9 a používateľské rozhranie reflektovalo stavy prostredníctvom delegáta (*rozhranie*) triedy `EnhancedVibeDetectorDelegate`. Pri detekovaní pohybu sa z obrázku vytvorí náhľad (*thumbnail*) a spolu s pôvodným obrázkom sa uloží do iCloudu cez CloudKit. Používateľ na master jednotke následne dostane notifikáciu obsahujúcu daný snímok.

6.4 Metriky kódu

V tejto kapitole boli popísané len najdôležitejšie časti riešenia implementovaných modulov a samotnej aplikácie. Veľa zdanlivo jednoduchých problémov je potrebné riešiť komplexne a implementovať tak veľké množstvo kódu. Z dôvodu obmedzeného rozsahu však nie je možné popísať detailnejšie časti implementácie. Názorná ukážka rozsiahlosti projektu je znázornená v tabuľke 6.1. Väčšina kódu je písaná v jazyku `Swift`, avšak grafické shadery sú písané v `Metal Shading Language`, ktorý je popísaný popísaný v sekcii 4.4.

Modul	Súborov	Prázdnych riadkov	Komentáre	Kód
Vibe	13	249	197	924
Vibe Shaders	3	37	33	129
VibeHome	46	752	554	3140
VibeCore	84	1562	1156	6802
Spolu	146	2600	1940	10995

Tabuľka 6.1: Tabuľka znázorňujúca rozsah zdrojových kódov pre jednotlivé moduly.

[11] **CocoaPods** – manažér závyslostí: https://cocoapods.org/
[12] https://webrtc.org/native-code/ios/

Kapitola 7

Testovanie

Táto kapitola predstaví všetky fázy a spôsoby testovania, ktoré boli použité v rámci danej aplikácie. V prvej časti 7.1 je možné sa dočítať, ktorý spôsob testovania sa použil počas celého vývoja až po finálne testovanie aplikácie. V druhej časti 7.2 je uvedené, ako prebiehalo testovanie prvého prototypu aplikácie a aké sú jeho výsledky. Následne je popísané finálne testovanie aplikácie a jej výdrži na zvolených zariadeniach. Na konci kapitoly sú na základe analýzy poznámok finálneho testovania vypísané návrhy na zlepšenie, ktorý by mohli byť vykonané pri budúcom vývoji.

7.1 Spôsob testovania

Testovanie počas celého vývoja aplikácie prebehlo celkovo na siedmych zariadeniach. Tri z nich sa používali ako zariadenia primárne: iPhone 7, iPhone SE, iPad 2017. Nasledujúce štyri zariadenia boli požičané od známych a používali sa na testovanie významnejších zmien v mojej aplikácii: iPhoneX, iPhone6S Plus, iPhone6S, iPhone7 Plus.

Usability testing

Usability testing alebo inak testovanie použiteľnosti je technika, ktorá sa používa na hodnotenie nejakého produktu alebo služby pomocou jeho testovania s/na reprezentatívnych používateľoch. Cieľom danej testovacej techniky je určiť prípadne výslednú spokojnosť používateľa s produktom. Obvykle sú účastníci daného testovacieho procesu moderátori (často vývojár aplikácie) s predpripraveným plánom testovania a náhodný používateľ (často predstaviteľ koncových používateľov). Taktiež sa tohoto procesu môžu zúčastniť aj pozorovatelia, ktorí pozorujú proces testovania a prispievajú tak k získavaniu výsledného obrazu pre celý proces. Testovanie použiteľnosti prebieha následovne: moderátor dáva používateľovi, ktorý nikdy pred tým nebol v styku s daným produktom, rôzne úlohy. Spolu s pozorovateľmi zaznamenáva verbálne a neverbálne poznámky používateľa behom procesu plnenia jednotlivých úloh. Po zhromaždení všetkých údajov a ich zotriedení sa vykoná následná analýza. Výstupom analýzy sú zistenia, čo presne mali vývojári produktu prerobiť alebo dokončiť, aby sa stal príťažlivejší a úspešný pre používateľa. Po vykonaní zmien sa uskutoční ďalšie kolo testovania. Toto sa vykonáva do doby, dokiaľ sa nezíska produkt, ktorý je pre používateľa uspokojivý [25, 14].

Priebežné testovanie aplikácie

Priebežné testovanie aplikácie prebiehalo iteratívne formou malých `usability` testov, ktoré sú popísané nižšie. Nešlo však o klasické testovanie použiteľnosti, nakoľko sa funkcie postupne testovali na dvoch alebo troch ľuďoch, ktorým sa po testovaní predložil krátky dotazník. Na základe týchto dotazníkov a skromnej spätnej väzbe účastníkov a taktiež vlastných predchádzajúcich skúseností, získaných pri tvorbe aplikácií, som následne vykonal zmeny a vylepšenia v danej aplikácii.

7.2 Testovanie prvého prototypu aplikácie

Prvý prototyp aplikácie bolo možné testovať s použitím dvoch zariadení. Pre komunikáciu medzi zariadeniami sa použil komunikačný model master/slave. V mojom prototype bola `slave` jednotka, ktorá vysielala mediálny stream a jednotka `master` stream prijímala.

Prototyp vypadal následovne 7.1 a mal naimplementovanú funkciu pre prenos audia a videa v reálnom čase, jednoduchú detekciu pohybu pomocou algoritmu `Running average`.

Testovanie prvého prototypu aplikácie prebiehalo v areále fakulty, kde som našiel študentov pre vykonanie testovania. Testovanie prebiehalo pár dní, počas doby návštev prednášok. Vo výsledku sa testovania zúčastnilo 14 ľudí. Po úvode a vykonaní samotného testovania mali respondenti vyplniť nižšie uvedené tvrdenia:

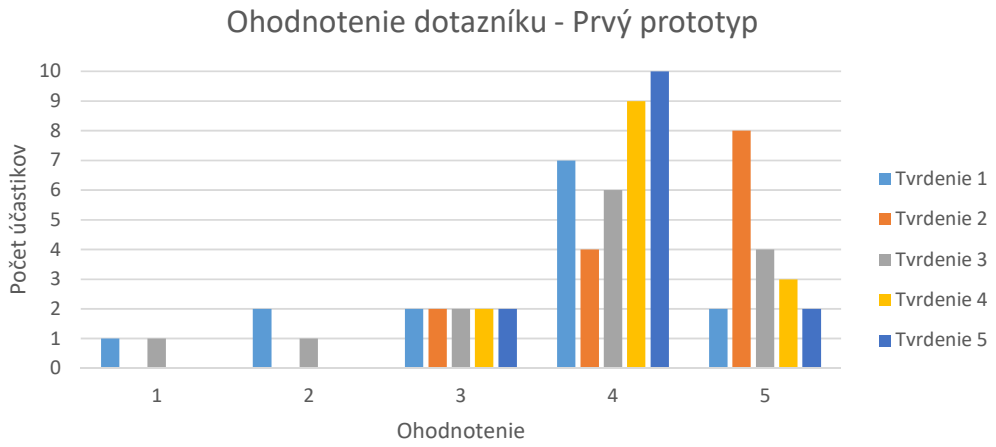
1. Celkový dojem z aplikácie mám pozitívny.
2. Páči sa mi nápad, ktorý sa snaží autor realizovať.
3. Danú aplikáciu považujem za praktickú a reálne použiteľnú v budúcnosti po jej dokončení.
4. Kvalita prenosu videa je pre 1. prototyp dostačujúca.
5. Detekcia pohybu je v danom prototype použiteľná v dostatočnej miere.

Odpoveď bolo možné zvoliť z 5 variánt, ktoré boli ohodnotené číslami s odpovedajúcim popisom: 1 – silne nesúhlasím; 2 – nesúhlasím, 3 – mám neutrálny názor; 4 – súhlasím; 5 — silne súhlasím. Výsledky daného dotazníku sú znázornené v grafe 7.1.

V rámci výsledkov testovanie bola od respondentov zozbieraná spätná väzba. Ich negatívne a pozitívne názory som utriedol a zhrnul do nasledujúcich bodov:

- Používateľom chýba galéria, v ktorej by bolo možné zobraziť snímky vytvorené pri detekovaní pohybu.
- Pri pomalom pohybe detekcia nefungovala úplne spoľahlivo.
- Používatelia spomenuli potencionálny prípad, kedy by chceli mať ako rodičia dohľad nad dieťaťom z viacerých jednotiek. V prototype bolo možné iba spojenie 1:1.

Po analýze spätnej väzby som prišiel k záveru, že hlavné zmeny, ktoré musím vykonať pre zlepšenie aplikácie sú implementácia galérie pre zobrazenie detekovaných fotiek. Taktiež ma to donútilo zamyslieť sa nad použitím zložitejšieho a spoľahlivejšieho algoritmu pre detekciu pohybu alebo vylepšiť výslednú aplikáciu o širšie možnosti monitorovacieho procesu (many-to-many). Všetky vyššie zhrnuté pripomienky sa následne iteratívne implementovali, testovali na vlastných zariadeniach. Vo výslednej variante sú pripomienky vyriešené.



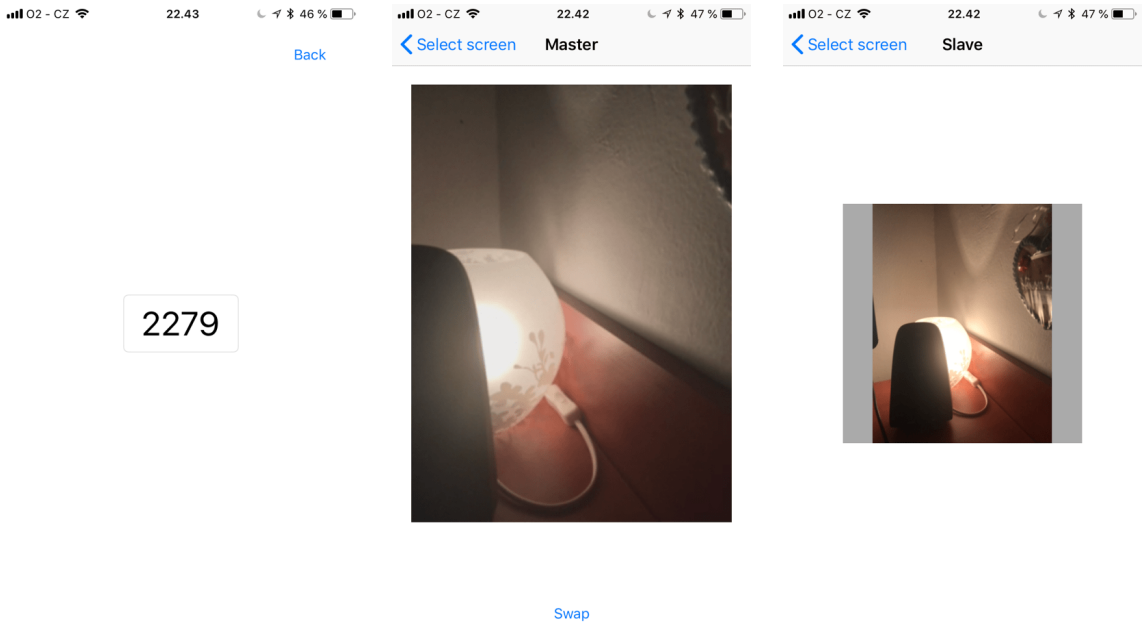
Tabuľka 7.1: Na grafe je jasne vidieť, že väčšina respondentov hodnotila celkový nápad aplikácie, jej použiteľnosť, perspektívu a funkcionalitu v danej etape veľmi kladne (hodnotením 4-5).

7.3 Finálne používateľské testovanie

Usability testovanie finálnej aplikácie prebehlo podľa respondentov hladko. Behom úvodu som každému respondentovi vysvetlil, čo je testovanie použiteľnosti a čo vlastne testuje, ako testovanie prebieha, aký je jeho prínos a ako bude testovanie trvať. Následne bolo treba skontrolovať, či si respondenti uvedomili, že behom testovacieho procesu majú hovoriť všetko nahlas. Po úvode sme sa s respondentmi pustili do samotného procesu testovania. Pre tento účel bol predom pripravený scenár (sada akcií), ktoré zahrňovali testovanie tých najdôležitejších častí aplikácie. Každá akcia obsahovala špecifické otázky, ktoré mi mali pomôcť získať reakcie o dôležitých súčastiach aplikácie. Následne mal na doplňujúce otázky reagovať číslom od 1 do 5, podobne ako v sekcii 7.2. Výsledky testovania sú znázornené v jednotlivých grafoch pod každou akciou zo scenára. Celkom sa testovania zúčastnilo 19 respondentov. Testovanie prebiehalo na aplikácii s jej pôvodným nastavením (prázdna aplikácia bez dát). Pripravený scenár akcií a tvrdení vyzeral následovne:

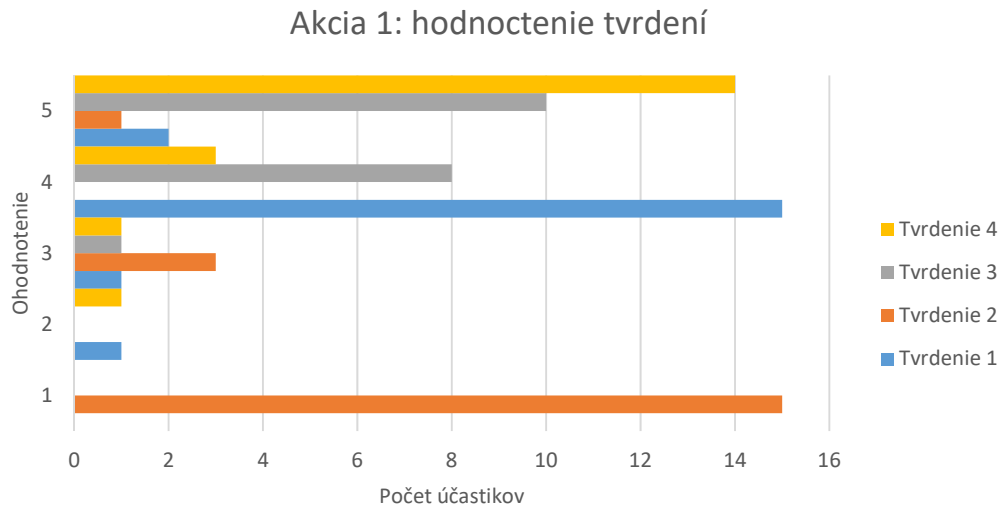
1). Spustite monitoring a začnite ho sledovať na druhom zariadení, následne ohodnoťte nižšie uvedené tvrdenia:

1. Párovací proces bol intuitívny.
2. Bol potrebný tutorial na vysvetlenie párovania.
3. Kvalitu prenášaného videa považujem za veľmi dobrú.
4. Celkový dojem z rýchlosti navigácie po aplikácii pri plnení úloh mam veľmi dobrý.



(a) Párovanie (b) Master jednotka (c) Slave jednotka

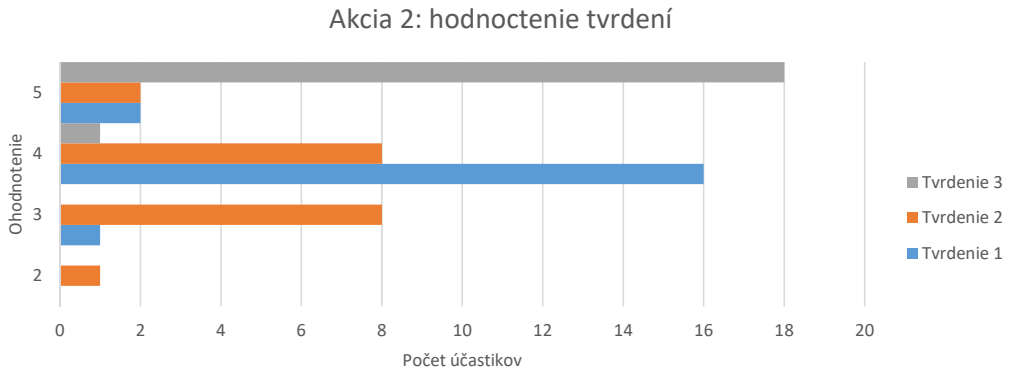
Obr. 7.1: Podoba aplikácie prvého prototypu. Obrázok a) zobrazuje proces párovania (generovanie párovacieho kódu). Obrázok b) zobrazuje master jednotku a tlačidlo na otočenie kamery. Obrázok c) zobrazuje slave jednotku a náhľad kamery.



Tabuľka 7.2: Znázornenie hodnotenia plnenia úloh v rámci akcie 1.

2). Spustite monitoring a zapnete prenos zvuku z monitorovacieho zariadenia, následne ohodnoťte nižšie uvedené tvrdenia:

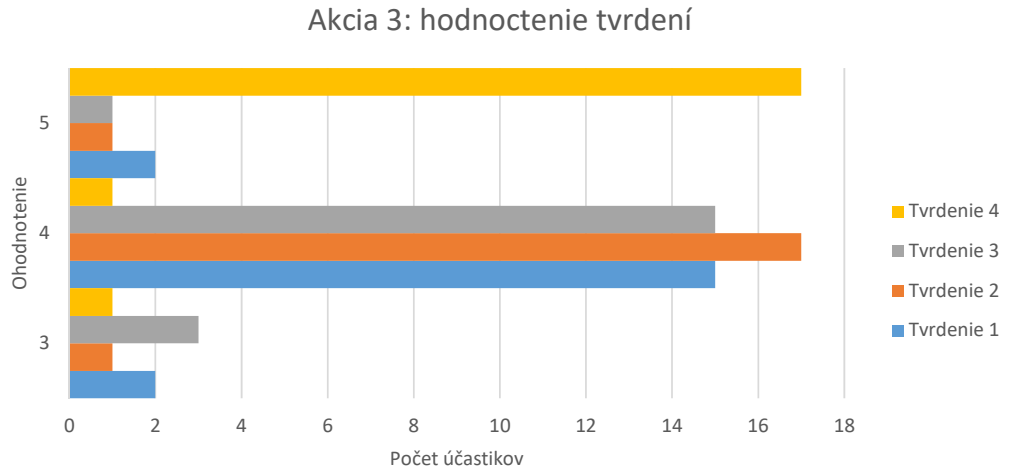
1. Hneď som prišiel na to, ako sa zapne odposluch audia z vysielacieho zariadenia.
2. Som si istý, že audio prenos beží v reálnom čase vďaka zmene statusu.
3. Kvalitu prenášaného audia považujem za veľmi dobrú.



Tabuľka 7.3: Znázornenie hodnotenia plnenia úloh v rámci akcie 2.

3). Spustite a následne vypnete detekciu pohybu na monitorovacom / vysielacom zariadení, vyvolajte pohyb pred kamerou a skontrolujte obsah galérie, následne ohodnoťte nižšie uvedené tvrdenia:

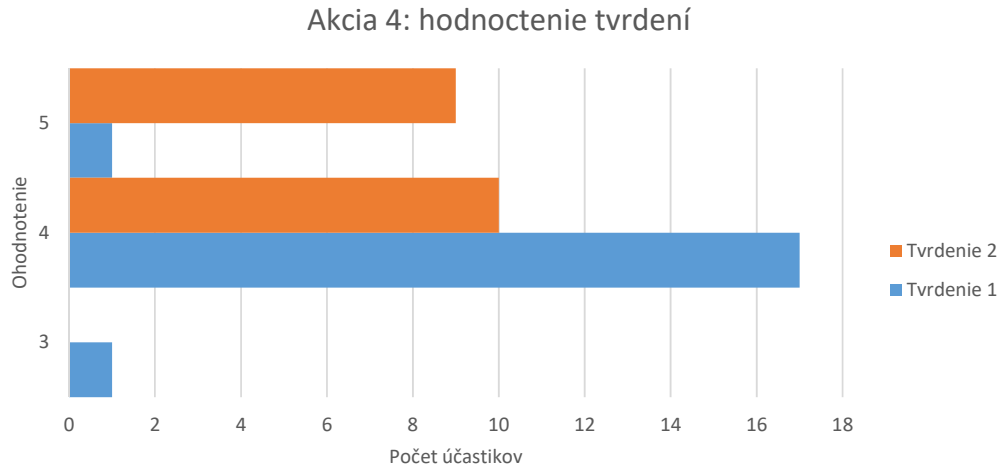
1. Hneď som prišiel na to, ako sa zapína detekcia pohybu na monitorovacom zariadení.
2. Hneď som prišiel na to, ako sa zapína detekcia pohybu na vysielacom zariadení.
3. Detekcia pohybu fungovala podľa predstáv.
4. Bolo jednoduché nájsť detekované snímky.



Tabuľka 7.4: Znázornenie hodnotenia plnenia úloh v rámci akcie 3.

4). Spustite monitoring a následne vymažte vysielaciu kameru, následne ohodnoťte nižšie uvedené tvrdenia:

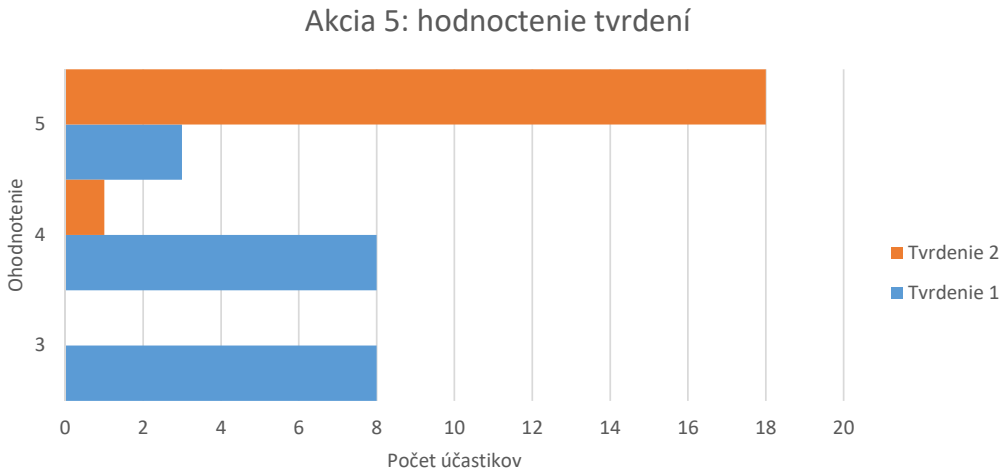
1. Tlačítko na vymazanie kamery som našiel hneď.
2. Po vymazaní kamery se stalo to, čo som očakával (vysvetlenie: vyskočilo odpovedajúce upozornenie na monitorovacom zariadení, prenos videa sa prerušil).



Tabuľka 7.5: Znázornenie hodnotenia plnenia úloh v rámci akcie 4.

5). Umožnite niekomu prostredníctvom zdieľania sledovať prenos videa z vysielajúceho zariadenia, následne ohodnoťte nižšie uvedené tvrdenia:

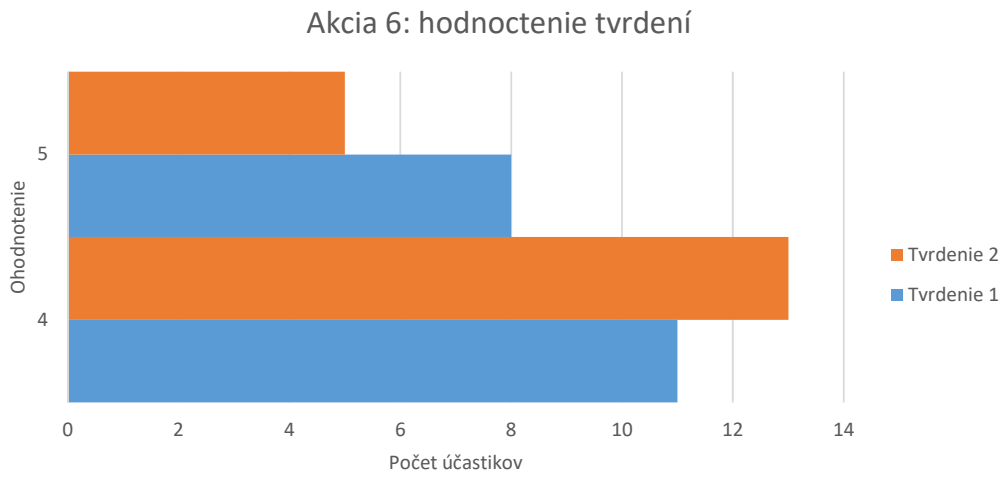
1. Proces zdieľania je jednoduchý a intuitívny.
2. Zdieľanie a sledovanie videa z vysielajúceho zariadenia prebehlo bez problémov.



Tabuľka 7.6: Znázornenie hodnotenia plnenia úloh v rámci akcie 5.

6). Celkový dojem z:

1. finálneho testovania aplikácie mam pozitívny a rád by som danú aplikáciu odporučil kamarátom.
2. výslednej aplikácie mam veľmi dobrý (aplikácia mi prijde ako intuitívna a konzistentná).



Tabuľka 7.7: Znázornenie hodnotenia plnenia úloh v rámci akcie 6.

Súhrn

Po analýze vyššie uvedených grafov je možné jednoznačne odvodiť, že výsledné testovanie aplikácie dopadlo veľmi pozitívne. Respondenti prakticky bez problému, rýchlo riešili zadané úlohy, čo je vidieť aj na ich hodnotení. Bolo aj pár hodnotení s nižšími známkami, ale bola to len malá časť respondentov. Zo spätnej väzby respondentov je zrejmé, že výsledný dojem znázornený na obrázku **??** je kladný.

7.4 Testovanie výdrže

Testovanie výdrže bolo vykonané v poslednej fáze vývoja aplikácie a používateľského testovania. Testovanie prebiehalo na troch zariadeniach s operačným systémom iOS 11.3.1: iPhone 7, iPhone SE, iPad 2017. Výdrž bola testovaná len v režime slave, nakoľko testovanie výdrže u master jednotky nemá zmysel, nakoľko nie je neustále spustená. Testované boli dva scenáre:

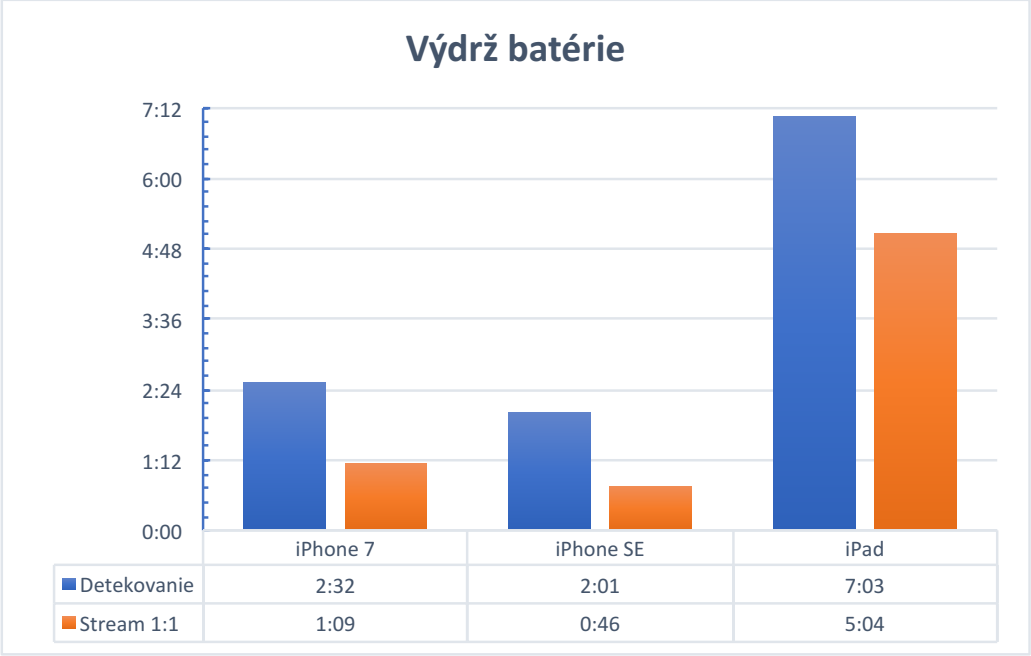
- Slave jednotka v režime so spustenou detekciou.
- Slave jednotka v režime prenosu audia a videa s vypnutou detekciou pohybu.

Výsledná vydrž zariadení je zobrazená v tabuľke 7.8. Počas prenosu sa na slave jednotke zobrazoval prenášaný obraz, čo je však potrebné len vo fáze nastavovania jednotky. Toto vylepšenie je spomenuté aj v nasledujúcich návrhoch na zlepšenie.

7.5 Návrhy na zlepšenie

Z analyzy výsledného testovania vyplynuli funkcie, ktoré by používatelia radi videli v aplikácii:

- **Pridanie podpory natáčania videa v okamžiku zachytenia pohybu**. Jedná sa o užitočnú funkciu, ktorá má určite zmysel. Otázkou však je, či by to zariadenie



Tabuľka 7.8: Výdrž zariadení v režime prenosu audia a videa medzi jednou master a slave jednotkou alebo v režime detekcie pohybu.

s aktívnou detekciou pohybu a aktívnym prenosom multimédií zvládlo po stránke výkonu.

- **Alarm pri prekročení nastavenej hranice zvuku.** Táto funkcia bola implementovaná v module aplikácie pre prenos audia a videa. Úroveň hlasitosti sa medzi jednotkami prenáša pri aktívnom spojení oboch jednotiek. Ak slave jednotka prekročí mnou nastavená úroveň zvuku, master jednotka zavibruje. To však neplatí v prípade, že neprebieha aktívny prenos. Implementácia tejto funkcie by spočívala v navrhnutí používateľského nastavenia pre upozornenie po prekročení hladiny zvuku a zaslaní notifikácie na druhé zariadenie.
- **V aplikácii chýba nastavenie minimálnej úrovne pohybu pre zaslanie oznámenia**. Táto hodnota je v aplikácii napevno nastavená a používateľ nemá možnosť, ako túto hodnotu nastaviť. Dôvodom pri návrhu bolo vytvorenie čo najjednoduchšieho UI, aby nebol používateľ zahltený mnohými zbytočnými voľbami. Toto nastavenie by však bolo veľmi vhodným rozšírením, ktoré by rozšírilo použiteľnosť aplikácie.
- **Vylepšenie výdrže slave jednotky pri streamovaní.** Istým riešením je počas doby streamovania znížiť programovo jas na najnižšiu možnú úroveň a taktiež zakázať zobrazenie streamovaného videa.
- **Bezpečnostné riziko**. Na slave jednotke je možné vymazať celú jednotku. Bolo by vhodné opatrenie kódom.

Kapitola 8

Záver

Cieľom tejto diplomovej práce bolo navrhnúť aplikáciu pre bezpečnosť domácnosti, umožňujúcu vzdialený dohľad nad domácnosťou prostredníctvom spárovaného mobilného zariadenia, čo sa úspešne podarilo splniť. Dôležitou časťou práce bola detekcia pohybu, ktorá patrí medzi problematiku oboru počítačového videnia. Výsledkom práce sú dva znovupoužiteľné moduly a výsledná aplikácia.

V rámci implementácie vznikol prototyp aplikácie, implementujúci algoritmus `Running average`, s využitím knižnice `OpenGL ES`. Ten sa však kvôli jeho nevhodným vlastnostiam stal nepoužiteľný v mojej práci. Preto som siahol pre populárny algoritmus ViBe, ktorý musel byť kvôli obmedzeniam implementovaný pomocou knižnice `Metal`. Algoritmus som implementoval ako znovupoužiteľný modul a jeho výsledná varianta bola vylepšená pomocou morfologických operácii.

Ďalším realizovaným modulom bol modul na prenos audia a videa. Pre tento modul som navrhol robustnú architektúru, ktorá ponúka možnosť prenosu peer-to-peer medzi viacerými zariadeniami (*many-to-many*). Súčasťou bolo aj prerobenie WebRTC knižnice a sprístupnenie delegátnych metód pre prístup k surovým dátam kamery zariadenia pre potreby analyzovania obrazu.

Tieto moduly boli integrované do výslednej aplikácie, ktorá má jednoduché a intuitívne rozhranie. Aplikácia úspešne implementuje synchronizáciu dát pomocou technológie CloudKit a dáta ukladá na lokálne zariadenie pomocou knižnice Realm. Jednotky sa párujú automaticky a zariadenia je možné zdielať s inými účtami iCloudu. V prípade zachytenia pohybu dostane prihlásený používateľ notifikáciu obsahujúcu zachytený snímok.

Aplikácia bola v rámci fáze prototypovania a vo výslednej fáze otestovaná na úzkom vzorku používateľov a boli navrhnuté zmeny pre ďalší možný vývoj.

Literatúra

[1] CKRecord. [Online; navštívené 22.04.2018].
URL https://developer.apple.com/documentation/cloudkit/ckrecord

[2] Internet Protocol, Version 6 (IPv6) Specification. [Online; navštívené 25.02.2018].
URL https://tools.ietf.org/html/rfc2460

[3] Introduction to WebRTC protocols. [Online; navštívené 21.01.2018].
URL
https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols

[4] Lifetime of a WebRTC session. [Online; navštívené 29.12.2017].
URL https:
//developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Session_lifetime

[5] Metal Shading Language Guide. [Online; navštívené 3.04.2018].
URL https://developer.apple.com/library/ios/documentation/Metal/
Reference/MetalShadingLanguageGuide/

[6] MQTT Introduction. [Online; navštívené 12.12.2017].
URL http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[7] OASIS Message Queuing Telemetry Transport (MQTT) TC. [Online; navštívené 5.01.2018].
URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

[8] RTCPeerConnection. [Online; navštívené 5.02.2018].
URL https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection

[9] SDP: Session Description Protocol. [Online; navštívené 3.03.2018].
URL https://tools.ietf.org/html/rfc4566

[10] A Study of WebRTC Security. [Online; navštívené 5.03.2018].
URL http://webrtc-security.github.io/

[11] Web Real-Time Communication (WebRTC: Media Transport and Use of RTP. [Online; navštívené 10.03.2018].
URL https://tools.ietf.org/html/draft-ietf-rtcweb-rtp-usage-25

[12] WebRTC connectivity. [Online; navštívené 5.01.2018].
URL
https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity

[13] WebRTC in the real world: STUN, TURN and signaling. [Online; navštívené 28.02.2018].
URL http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/#after-signaling-using-ice-to-cope-with-nats-and-firewalls

[14] What is usability testing? [Online; navštívené 15.05.2018].
URL https://www.experienceux.co.uk/faqs/what-is-usability-testing/

[15] Khronos Releases OpenGL ES 3.1 Specification. Mar 2014, [Online; navštívené 6.01.2018].
URL https://www.businesswire.com/news/home/20140317005673/en/Khronos-Releases-OpenGL-ES-3.1-Specification

[16] Working with Metal - Overview. 2014, [Online; navštívené 9.02.2018].
URL http://devstreaming.apple.com/videos/wwdc/2014/603xx33n8igr5n1/603/603_working_with_metal_overview.pdf

[17] Dec 2016, [Online; navštívené 7.04.2018].
URL https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/MetalProgrammingGuide/WhatsNewiniOS10tvOS10andOSX1012/WhatsNewiniOS10tvOS10andOSX1012.html

[18] Attack of the week: Datagram TLS. Aug 2016, [Online; navštívené 8.01.2018].
URL http://blog.cryptographyengineering.com/2012/01/attack-of-week-datagram-tls.html

[19] Introducing the MQTT Security Fundamentals. Dec 2016, [Online; navštívené 16.12.2017].
URL
https://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals

[20] *Reachability - Guides and Sample Code*. Květen 2016, [Online; navštívené 05.01.2018].
URL https://developer.apple.com/library/content/samplecode/Reachability/Introduction/Intro.html

[21] *What is Reactive Programming?* Prosinec 2016, [Online; navštívené 10.01.2018].
URL https://medium.com/@kevalpatel2106/what-is-reactive-programming-da37c1611382

[22] OpenGL ES Programming Guide - Application design. Mar 2017, [Online; navštívené 8.01.2018].
URL https://developer.apple.com/library/content/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLESApplicationDesign/OpenGLESApplicationDesign.html

[23] OpenGL ES Programming Guide - Best practices. Mar 2017, [Online; navštívené 3.01.2018].
URL https://developer.apple.com/library/content/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/BestPracticesforAppleA7GPUsandLater/

[24] *Zdrojové kódy WebRTC*. Leden 2018, [Online; navštívené 13.01.2018].
URL https://webrtc.googlesource.com/src

[25] Affairs, A. S. f. P.: Usability Testing. Nov 2013, [Online; navštívené 15.05.2018].
URL https:
//www.usability.gov/how-to-and-tools/methods/usability-testing.html

[26] ANSI, A. N. S. I.: Human System Interaction Ergonomics Standards. [Online; navštívené 14.03.2018].
URL https:
//webstore.ansi.org/ergonomics/human_system_interaction_ergonomics.aspx

[27] Barnich, O.; Droogenbroeck, M. V.: ViBE: A powerful random technique to estimate the background in video sequences. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, April 2009, ISSN 1520-6149, s. 945–948, doi:10.1109/ICASSP.2009.4959741.

[28] Barnich, O.; Droogenbroeck, M. V.: ViBe: A Universal Background Subtraction Algorithm for Video Sequences. *IEEE Transactions on Image Processing*, ročník 20, č. 6, June 2011: s. 1709–1724, ISSN 1057-7149, doi:10.1109/TIP.2010.2101613.

[29] Christogiannopoulos, G.; Birch, P. B.; Young, R. C.; aj.: Segmentation of moving objects from cluttered background scenes using a running average model. *SPIE Journal*, ročník 5822, 2005: s. 13–20.

[30] Cucchiara, R.; Grana, C.; Piccardi, M.; aj.: Detecting moving objects, ghosts, and shadows in video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 25, č. 10, Oct 2003: s. 1337–1342, ISSN 0162-8828, doi:10.1109/TPAMI.2003.1233909.

[31] Droogenbroeck, M. V.; Barnich, O.: Visual Background Extractor. Jan 2009, [Online; navštívené 15.01.2018].
URL https://patentscope.wipo.int/search/en/detail.jsf?docId=WO2009007198

[32] Fablet, Y.: Announcing WebRTC and Media Capture. Jun 2017, [Online; navštívené 28.03.2018].
URL https://webkit.org/blog/7726/announcing-webrtc-and-media-capture/

[33] Grigorik, I.: Browser APIs and Protocols: WebRTC - High Performance Browser Networking (O'Reilly). May 2016, [Online; navštívené 16.02.2018].
URL https://hpbn.co/webrtc/

[34] He, Z.; Liu, Y.; Yu, H.; aj.: Optimized Algorithms for Traffic Information Collecting in an Embedded System. In *2008 Congress on Image and Signal Processing*, ročník 4, May 2008, s. 220–223, doi:10.1109/CISP.2008.614.

[35] Heikkilä, J.; Silvén, O.: A real-time system for monitoring of cyclists and pedestrians. *Image and Vision Computing*, ročník 22, č. 7, 2004: s. 563–570.

[36] Lai, A. H. S.; Yung, N. H. C.: A fast and accurate scoreboard algorithm for estimating stationary backgrounds in an image sequence. In *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, ročník 4, May 1998, s. 241–244 vol.4, doi:10.1109/ISCAS.1998.698804.

[37] Liu, Y.; Yao, H.; Gao, W.; aj.: Nonparametric Background Generation. In *18th International Conference on Pattern Recognition (ICPR'06)*, ročník 4, 2006, ISSN 1051-4651, s. 916–919, doi:10.1109/ICPR.2006.868.

[38] Mishra, P. K.; Saroha, G. P.: A study on video surveillance system for object detection and tracking. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, March 2016, s. 221–226.

[39] Mittal, A.; Paragios, N.: Motion-based background subtraction using adaptive kernel density estimation. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, ročník 2, June 2004, ISSN 1063-6919, s. II–302–II–309 Vol.2, doi:10.1109/CVPR.2004.1315179.

[40] Mohanty, S.; Mohanty, A. K.; Carminati, F.: Efficient pseudo-random number generation for monte-carlo simulations using graphic processors. *Journal of Physics: Conference Series*, ročník 368, č. 1, 2012: str. 012024, [Online; navštívené 4.02.2018]. URL http://stacks.iop.org/1742-6596/368/i=1/a=012024

[41] Oliver, N. M.; Rosario, B.; Pentland, A. P.: A Bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 22, č. 8, Aug 2000: s. 831–843, ISSN 0162-8828, doi:10.1109/34.868684.

[42] Park, J.; Tabb, A.; Kak, A. C.: Hierarchical Data Structure for Real-Time Background Subtraction. In *2006 International Conference on Image Processing*, Oct 2006, ISSN 1522-4880, s. 1849–1852, doi:10.1109/ICIP.2006.312840.

[43] Pham, V.; Vo, P.; Hung, V. T.; aj.: GPU Implementation of Extended Gaussian Mixture Model for Background Subtraction. In *2010 IEEE RIVF International Conference on Computing Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, Nov 2010, s. 1–4, doi:10.1109/RIVF.2010.5634007.

[44] Piccardi, M.: Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, ročník 4, Oct 2004, ISSN 1062-922X, s. 3099–3104 vol.4, doi:10.1109/ICSMC.2004.1400815.

[45] Porikli, F.; Tuzel, O.: Human body tracking by adaptive background models and mean-shift analysis. In *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 2003, s. 1–9.

[46] Qiming, Z.; ChengQian, M.: A vehicle detection method in tunnel video based on ViBe algorithm. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, March 2017, s. 1545–1548, doi:10.1109/IAEAC.2017.8054272.

[47] Rahman, M. A.; Ahmed, B.; Hossian, M. A.; aj.: An adaptive background modeling based on modified running Gaussian average method. In *2017 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, Feb 2017, s. 524–527, doi:10.1109/ECACE.2017.7912961.

[48] Sigari, M. H.; Mozayani, N.; Pourreza, H.: Fuzzy running average and fuzzy background subtraction: concepts and application. *International Journal of Computer Science and Network Security*, ročník 8, č. 2, 2008: s. 138–143.

[49] Simjp: A Look Into Realm's Core DB Engine. [Online; navštívené 21.03.2018].
URL
https://academy.realm.io/posts/jp-simard-realm-core-database-engine/

[50] Singh, A.; Sawan, S.; Hanmandlu, M.; aj.: An Abandoned Object Detection System Based on Dual Background Segmentation. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, Sept 2009, s. 352–357, doi:10.1109/AVSS.2009.74.

[51] Spagnolo, P.; Leo, M.; D'Orazio, T.; aj.: Robust moving objects segmentation by background subtraction. In *The International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, 2004.

[52] Stauffer, C.; Grimson, W. E. L.: Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, ročník 2, 1999, ISSN 1063-6919, str. 252 Vol. 2, doi:10.1109/CVPR.1999.784637.

[53] sunny.hu: I have problems with Alfred's WebViewer. What should I do? Jan 2018, [Online; navštívené 27.03.2018].
URL https://alfred.camera/forum/t/i-have-problems-with-alfreds-webviewer-what-should-i-do/1246963

[54] Thurau, C.; Hlavac, V.: Pose primitive based human action recognition in videos or still images. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, June 2008, ISSN 1063-6919, s. 1–8, doi:10.1109/CVPR.2008.4587721.

[55] Uberti, J.; Dutton, S.: Plugin-free realtime communication. [Online; navštívené 12.01.2018].
URL http://io13webrtc.appspot.com/

[56] Wang, W.; Chen, D.; Gao, W.; aj.: Modeling background from compressed video. In *2005 IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, Oct 2005, s. 161–168, doi:10.1109/VSPETS.2005.1570911.

[57] Widyawan; Zul, M. I.; Nugroho, L. E.: Adaptive motion detection algorithm using frame differences and dynamic template matching method. In *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Nov 2012, s. 236–239, doi:10.1109/URAI.2012.6462984.

[58] Wren, C. R.; Azarbayejani, A.; Darrell, T.; aj.: Pfinder: Real-time tracking of the human body. *IEEE Transactions on pattern analysis and machine intelligence*, ročník 19, č. 7, 1997: s. 780–785.

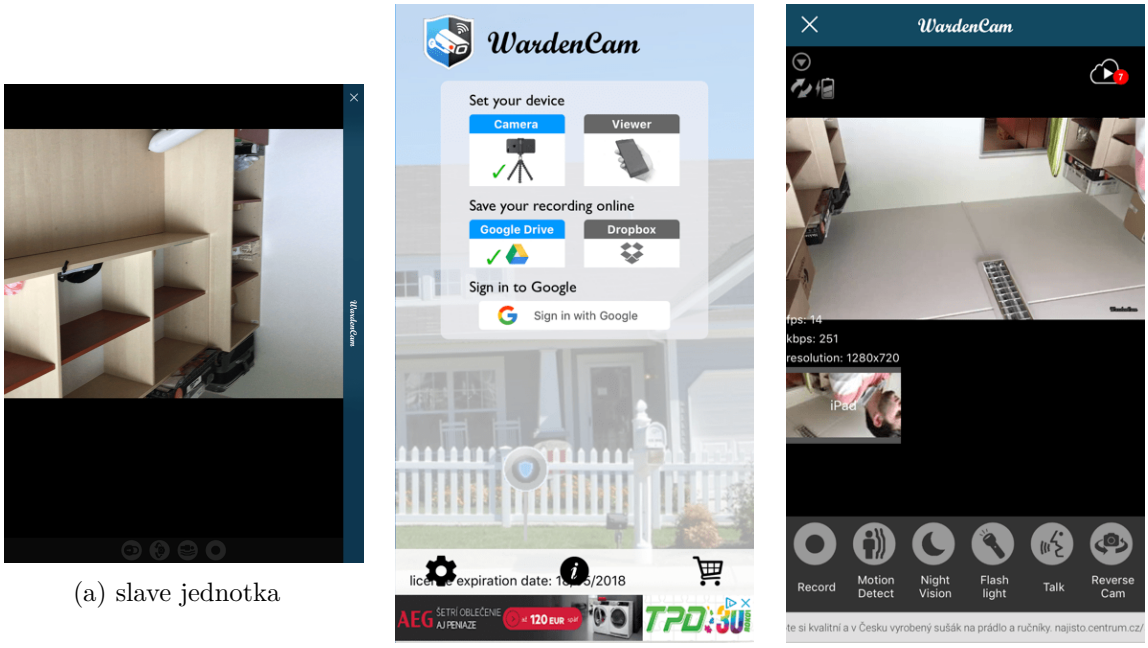
[59] Yi, Z.; Liangzhong, F.: Moving object detection based on running average background and temporal difference. In *2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering*, Nov 2010, s. 270–272, doi:10.1109/ISKE.2010.5680866.

Prílohy

Príloha A

Používateľské rozhranie ďalších najpopulárnejších aplikácií

Aplikácia WardenCam[1]



(a) slave jednotka

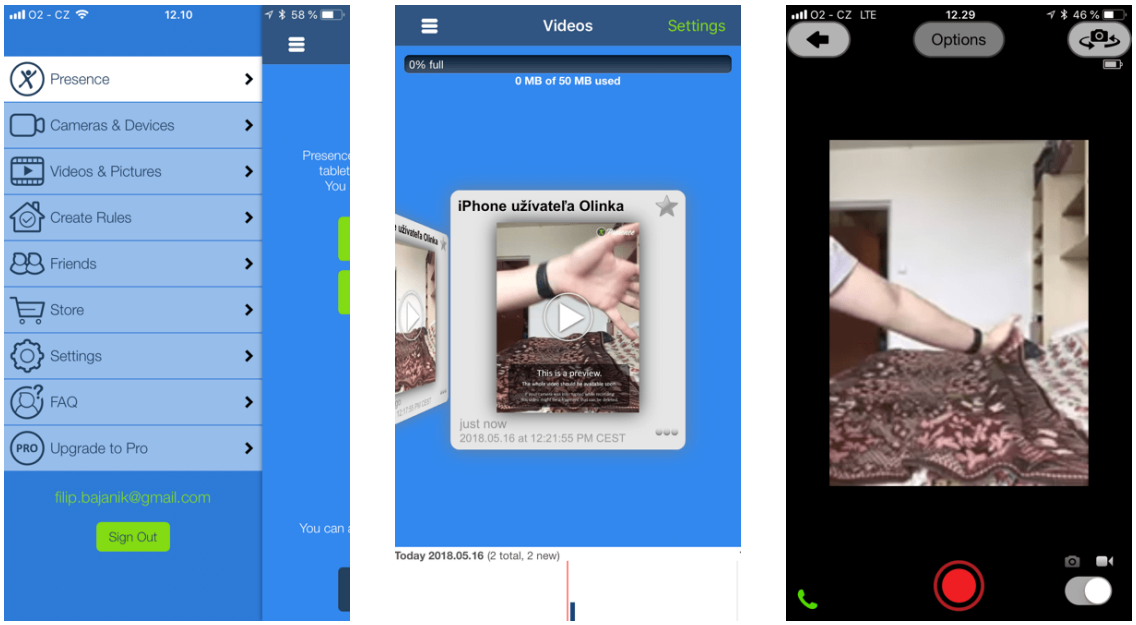
(b) úvodná obrazovka

(c) master jednotka

Obr. A.1: Ukážka používateľského rozhrania aplikácie WardenCam.

[1] https://itunes.apple.com/us/app/wardencam-video-surveillance/id914224766?mt=8

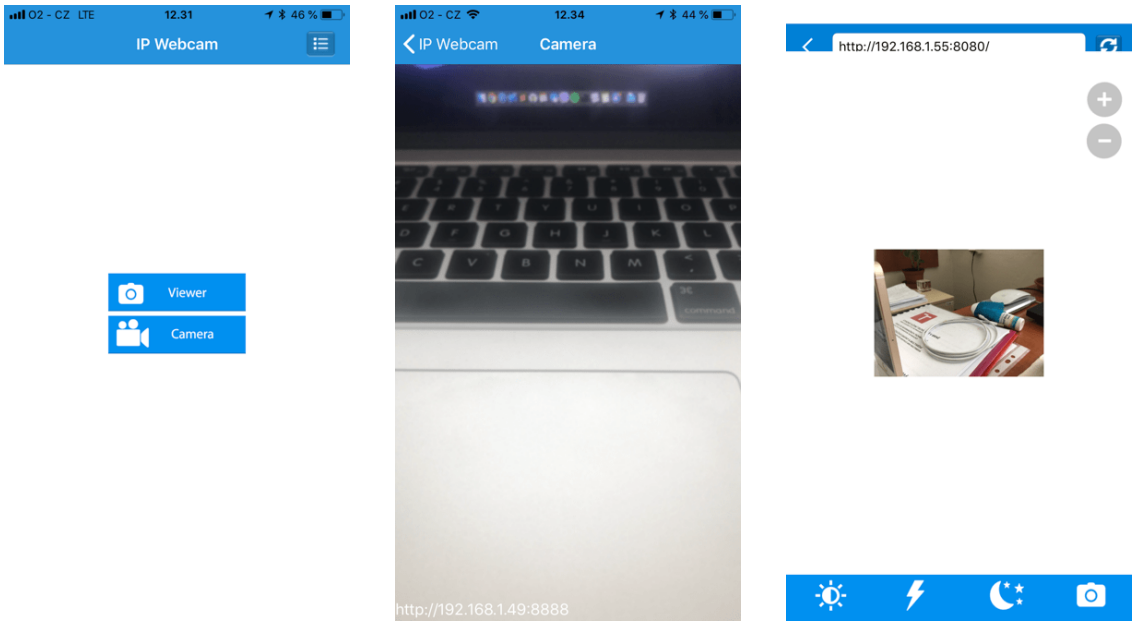
Aplikácia Presence Video Security[2]



(a) menu aplikácie (b) galéria (c) master jednotka

Obr. A.2: Ukážka používateľského rozhrania aplikácie Presence.

Aplikácia IP Webcam Home Security Camera[3]

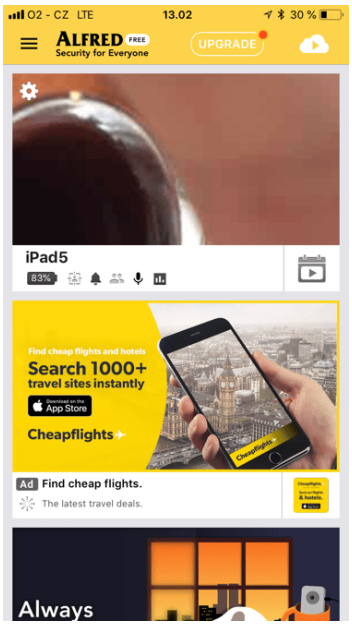


(a) úvodná obrazovka (b) slave jednotka (c) master jednotka

Obr. A.3: Ukážka používateľského rozhrania aplikácie IP Webcam.

[2] https://itunes.apple.com/us/app/presence-video-security/id618598211?mt=8
[3] https://itunes.apple.com/us/app/ip-webcam-home-security-camera/id1264454867?mt=8

Aplikácia Alfred Security Camera[4]



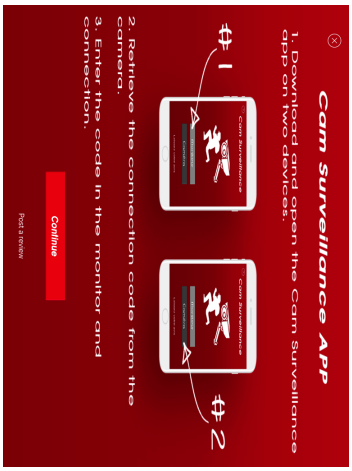
(a) úvodná obrazovka

(b) galéria

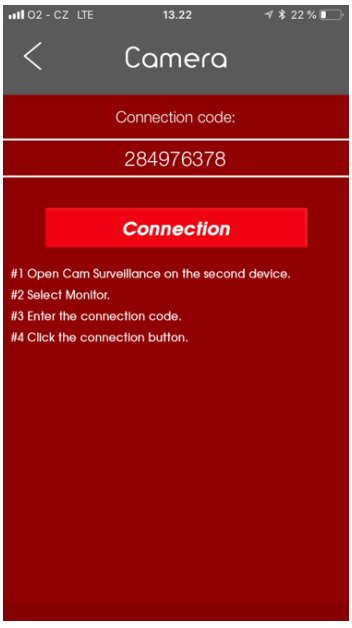
(c) master jednotka

Obr. A.4: Ukážka používateľského rozhrania aplikácie Alfred.

Aplikácia Security Camera – Surveillance Video[5]



(a) úvodná obrazovka (iPad)



(b) párovanie

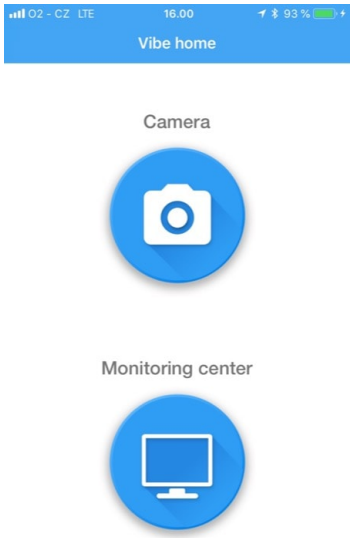
(c) master jednotka

Obr. A.5: Ukážka používateľského rozhrania aplikácie Security Camera – Surveillance.

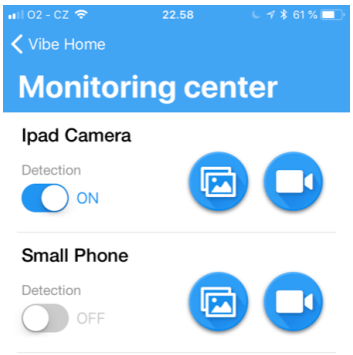
[4] https://itunes.apple.com/us/app/alfred-security-camera/id966460837?mt=8
[5] https://itunes.apple.com/cz/app/security-camera-surveillance-video/id1272756234?mt=8

Príloha B

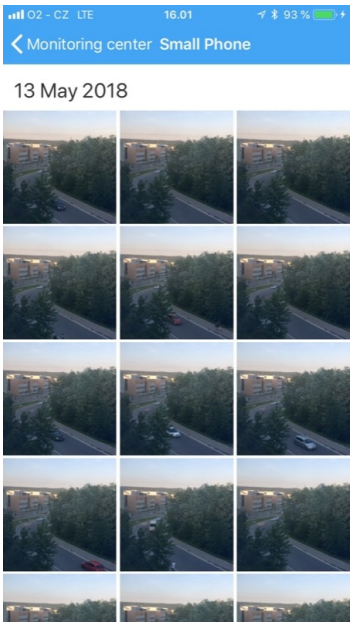
Podoba výslednej aplikácie



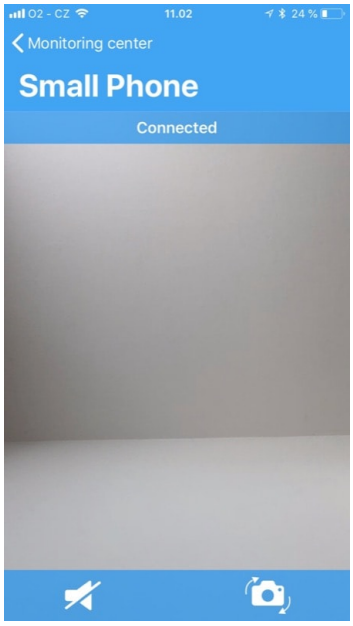
(a) Úvodná obrazovka



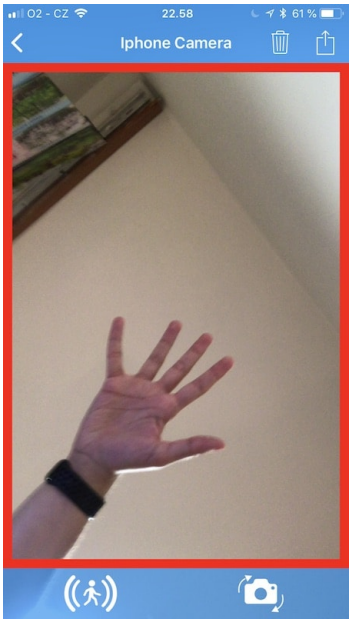
(b) Zoznam jednotiek



(c) Galéria snímkov



(d) Master jednotka



(e) Slave jednotka

Príloha C

Obsah CD

- Prezentačný plagát (**poster.pdf**)
- Krátka ukážka používania aplikácie (**video.mp4**)
- Jadro na prenos audia a videa (**VibeHomeCore**)
- Jadro pre detekciu pohybu (**VibeDetection**)
- Výsledna aplikácia (**VibeHome**)
- Patch pre WebRTC framework (**WebRTC**)