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

CHYTRÉ DIGITÁLNÍ ZRCADLO PRO INDIVIDUÁLNÍ SPORTY

SMART DIGITAL MIRROR FOR INDIVIDUAL SPORTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. TOMÁŠ OPICHAL**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Opichal Tomáš, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Inteligentní systémy
Název: **Chytré digitální zrcadlo pro individuální sporty**
Smart Digital Mirror for Individual Sports
Kategorie: Uživatelská rozhraní
Zadání:

1. Seznamte se s problematikou vývoje mobilních aplikací, zaměřte se na zpracování obrazu z kamery a na přenos a ukládání videa / obrazu.
2. Identifikujte vhodné případy užití systému, který bude fungovat jako chytré zrcadlo pro nácvik sportů a tance.
3. Prototypujte jednotlivé prvky navrženého systému, testujte s uživateli / testery. Zaměřte se na pořizování obrazu, přenos videa, zobrazování videa z více kamer, na ovládání / uživatelské rozhraní.
4. Navrhněte modulární a přitom dobře integrovaný systém sloužící jako chytré zrcadlo.
5. Implementujte navržený systém a jeho jednotlivé komponenty.
6. Testujte vyvíjený systém s testery / uživateli a iterativně jej vylepšujte.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; 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: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012
- Android Developers: https://developer.android.com/index.html
- Alan B Johnston, Daniel C Burnett: WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web, Digital Codex LLC, 2014

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 až 5.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 18. května 2021

Abstrakt

Cílem této diplomové práce je vytvořit praktické řešení pro individuální sportovce v podobě digitálního zrcadla, které se může stát alternativou k pozorování se v běžném zrcadle. Práce obsahuje a zdůrazňuje výhody digitálního zrcadla. Velká část práce se věnuje přenosu videa mezi různými zařízeními pro možnost spojení více obrazů na jednom displeji. Jádrem řešení je mobilní aplikace pro platformu Android, která funguje v různých režimech, jejichž volba závisí na způsobu použití aplikace. Aplikace byla průběžně testována se skupinou uživatelů, výsledkem bylo iterativní vylepšování funkcí aplikace a přidávání funkcí nových. Dále byla implementována webová varianta aplikace, což zvyšuje možnost využití digitálního zrcadla napříč různými platformami. Výsledky testování ukazují, že celé toto řešení je využitelné při praktickém nácviku sportovních prvků a aplikace má potenciál oslovit různá sportovní odvětví.

Abstract

The aim of this diploma thesis is to create a practical solution for individual athletes in the form of a digital mirror, which can become an alternative to observing yourself in an ordinary mirror. The work contains and emphasizes the benefits of a digital mirror. Much of the work is devoted to the transfer of video between various devices for the ability to combine multiple videos on a single screen. The core of the solution is a mobile application for the Android platform, which works in various modes, the choice of which depends on how the application is used. The application was continuously tested with a group of users, resulting in iterative improvements to the application's features and the addition of new features. Furthermore, a web variant of the application was implemented, which increases the possibility of using the digital mirror across different platforms. The results of testing show that this whole solution can be used in the practical training of sports exercises and the application has the potential to connect with various sports.

Klíčová slova

sport, přenos videa, peer-to-peer, mobilní aplikace, uživatelská zkušenost, UX, Android, WebRTC, Wi-Fi Direct, JSON, PHP, Angular

Keywords

sport, video streaming, peer-to-peer, mobile application, user experience, UX, Android, WebRTC, Wi-Fi Direct, JSON, PHP, Angular

Citace

OPICHAL, Tomáš. *Chytré digitální zrcadlo pro individuální sporty.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Chytré digitální zrcadlo pro individuální sporty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Opichal
18. května 2021

Poděkování

Rád bych poděkoval panu prof. Ing. Adamu Heroutovi, Ph.D. za jeho odbornou pomoc a přátelský přístup, který se pozitivně podepsal na mém odhodlání vytvořit práci co nejvyšší kvality. V neposlední řadě děkuji také svým nejbližším za jejich trpělivost a pochopení.

Obsah

Kapitola 1

Úvod

Lidé provozující technicky založené sporty mají často za cíl se neustále zdokonalovat v provedení jednotlivých sportovních prvků. Pozorování se v zrcadle může být jedním ze způsobů, jak si rychle ověřit, zda provedení daného prvku má takové parametry, jaké sportovec očekává. Ne vždy se ovšem jedná o vhodný způsob. Kontrola v zrcadle často vyžaduje například *otočit hlavu* směrem k zrcadlu, vyžaduje navíc vyšší dávku *koncentrace* – lidský mozek se v daný moment musí soustředit na koordinaci pohybu a zároveň zpracovávat vizuální informaci v zrcadle.

Cílem této práce je vytvořit nástroj, který bude sloužit lidem z oblasti individuálních sportů. Motivací k vytvoření takového nástroje je umožnit sportovcům okamžitou i zpětnou kontrolu správnosti a kvality provedení různých prvků spadajících do jejich sportovního odvětví. Zejména se jedná o sporty a cvičení jako jóga, fitness, sportovní tanec, balet, gymnastika, ale i tenis, stolní tenis nebo badminton.

Hlavní funkcí digitálního zrcadla je zobrazení video výstupu z tréninku sportovců a umožnění snadné konfigurace takového výstupu. Řešení by mělo nabízet různé režimy natáčení a zobrazování video výstupu, které budou vhodné pro jednotlivé uživatele a sportovní odvětví. Výstup by měl být zobrazován v reálném čase, aby měl sportovec po každém pokusu o provedení prvku možnost kontroly. Nástroj musí zajistit odpovídající kvalitu video výstupu a také přehledně zpracované uživatelské rozhraní.

Řešení bude zahrnovat funkci párování více zařízení, která umí zaznamenávat videa. To umožní zobrazit sportovci více video výstupů najednou. Na zvoleném zařízení s obrazovým výstupem pak lze pozorovat provedení daného prvku například ze dvou různých úhlů, což může výrazně zlepšit přehled sportovce o tom, jak kvalitně prvek provedl.

V rámci řešení diplomové práce jsem nejprve hledal vhodné technologie. Mimo jiné jsem experimentoval s knihovnou WebRTC, nejprve ve webovém prostředí, později na platformě Android. Výsledky experimentů vedly k vytvoření první verze mobilní aplikace *Mirrorer* obsahující základní funkcionalitu produktu, který byl iterativně testován skupinou uživatelů. Výsledkem řešení práce je produkt zahrnující mobilní a webovou aplikaci a webový signalizační server. Uživatelské testování ukazuje, že toto řešení má potenciál získat si své fanoušky a být pro sportovce inovativním pomocníkem při tréninku technických prvků.

Kapitola 2

Očekávaný výsledek

Řešení bude velmi pravděpodobně *webová* či *nativní* aplikace s uživatelským rozhraním umožňující zaznamenávání videa a jeho zobrazování v reálném čase. Následující cíle jsou **nezbytné** k vytvoření funkčního řešení stanoveného problému.

- Zajistit funkcionalitu a definovat protokol, kterým budou zařízení navazovat spojení.
- Umožnit přenášení videa v reálném čase mezi více zařízeními.
- Umožnit zobrazit jeden i více lokálních i vzdálených *streamů* na obrazovce zařízení – monitoru, tabletu, či jiného podobného.

Dále by se řešení mělo snažit o co nejlepší splnění níže uvedených cílů.

- V rámci řešení oddělit zařízení typu *kamera* a zařízení typu *displej*. Toto zvýší *modularitu* výsledného řešení, což povede k lepší organizaci kódu a pomůže uživatelům se v aplikaci rychleji zorientovat.
- Přinést uživatelům další funkce, které co nejlépe vystihnou jejich potřeby. Přitom myslet na co možná nejjednodušší použití těchto funkcí – řešení je určeno zejména sportovcům. Zvážit tedy hlasové ovládání, případně gesta a jiné inovativní technologie ovládání zařízení.

Nyní uvedu několik příkladů funkcí či rozšíření aplikace, která by mohla být prospěšná koncovým uživatelům. Ne všechny funkce musí být implementovány ve výsledném řešení, a zároveň je pravděpodobné, že během uživatelského testování přibudou funkce, které zmíněny nejsou.

- Zpoždění streamu – umožní sportovci ihned po provedení prvku zhlédnout pokus.
- Nahrávání streamu – pro možnost pozdějšího zhlédnutí pokusu.
- Nastavení kvality video výstupu.
- Pokročilá konfigurace zařízení typu displej – ve smyslu uspořádání jednotlivých obrazových výstupů na obrazovce a přizpůsobení jejich velikosti dle vlastních preferencí.
- Automatizovat proces párování jednotlivých zařízení.

2.1 Inspirace v existujících řešeních

Abych načerpal inspiraci a zároveň zjistil, jestli již dnes existují aplikace, které by umožňovaly sportovcům analyzovat exekuci daných cviků rozhodl jsem se, že se takové aplikace pokusím vyhledat. Smyslem tohoto průzkumu nebylo najít aplikace, která by zcela odpovídala mému řešení, ale spíše zjistit, jakým způsobem vývojáři k tomuto problému přistupují, což mi umožní vytvořit produkt, který bude svým řešením unikátní a obtížně nahraditelný.

2.1.1 Mirror

Jedním z řešení, která jsem objevil byl přístroj s názvem **Mirror**. Hlavním prvkem tohoto řešení je fyzické chytré zrcadlo, které obsahuje předinstalovanou aplikaci pro podporu sportovního tréninku. Zrcadlo tedy umí zobrazit klasický odraz sportovce a zároveň obrazový výstup z aplikace – navigační prvky aplikace, instruktážní videa apod. Fyzické zrcadlo je často zavěšeno na zdi podobně jako běžné zrcadlo. Hlavním případem užití tohoto systému je cvičení dle instruktážního videa v reálném čase – sportovec sleduje instruktáž, a zároveň může kontrolovat vlastní provádění cviků v odraze zrcadla, což ukazuje obrázek 2.1. Instruktážní videa jsou přímo součástí aplikace.

Vzhledem k povaze tohoto řešení jsem neměl možnost jej vyzkoušet osobně. Kromě podpory sportovního tréninku obsahuje také funkci propojení se zařízením které měří tepovou frekvenci sportovce. Dále umožňuje uživatelům vyfotit tzv. *selfie*, kterou mohou sdílet se svým okruhem přátel.

Tento produkt bych zařadil spíše do kategorie řešení, která širokou veřejnost motivují ke sportu a dělají sportovní činnost interaktivnější a zábavnější. Přidaná hodnota z pohledu kontroly a analýzy prováděných prvků oproti běžnému zrcadlu je minimální.

2.1.2 Coach's eye

Jedná se o mobilní aplikaci pro platformy Android a iOS. Aplikace umožňuje nahrávat, případně vkládat videa ze sportovního tréninku a umožňuje jejich zpětnou analýzu. V rámci analýzy lze, kromě běžných funkcí, video zpomalit a zakreslovat do něj objekty jako čáry, obdélníky, úhly apod. – toho se využívá při zakreslování rozdílů mezi skutečným a ideálním provedením daného prvku. Analýzu provádí zpravidla trenér, sportovec může sledovat analýzu přímo na zařízení trenéra. Analýzu lze ovšem i nahrát, což umožňuje interaktivní komunikaci mezi sportovcem a trenérem i v případě, kdy trenér není přítomen v době tréninku. Z tohoto důvodu obsahuje aplikace rovněž funkci pro sdílení nahraných videí.

Z pohledu cíle mé práce mě aplikace zaujala zejména možností získat zpětnou vazbu od trenéra i bez jeho přítomnosti. Aplikace ovšem neobsahuje funkci, která by umožňovala zpětnou kontrolu prováděných prvků aniž by muselo vzniknout hotové video – například metodou zpoždění video stopy.

Z mého pohledu se jedná o užitečnou aplikaci, která posouvá komunikaci mezi sportovcem a trenérem na další úroveň. Pokud však sportovec chce provádět kontrolu sám nad sebou, je tato aplikace srovnatelná s běžným přehrávačem videí.

2.1.3 CoachNow

Tato mobilní aplikace je obdobou aplikace **Coach's eye**. Obsahuje velmi podobné funkce, jako je zpomalení videa či zakreslování objektů do videa. Její hlavní přidanou hodnotou je možnost vytvoření skupiny uživatelů, kteří si mohou vzájemně sdílet a komentovat video

Obrázek 2.1: Ukázka produktu Mirror – cvičení dle instruktážního videa, které je spuštěno v prostředí chytrého zrcadla. Zdroj: https://www.mirror.co.

obsah. V jistém slova smyslu se jedná o jednoduchou sociální síť. Tento fakt posouvá aplikaci za hranice individuálních sportů. Trenér může analyzovat videa způsobem, který je podobný tzv. *elektronické tužce* známé z analýzy hokejových zápasů – zakreslit chyby v rozestavení hráčů na ploše, špatně načasovaný náběh apod.

Kapitola 3

Použité technologie a standardy

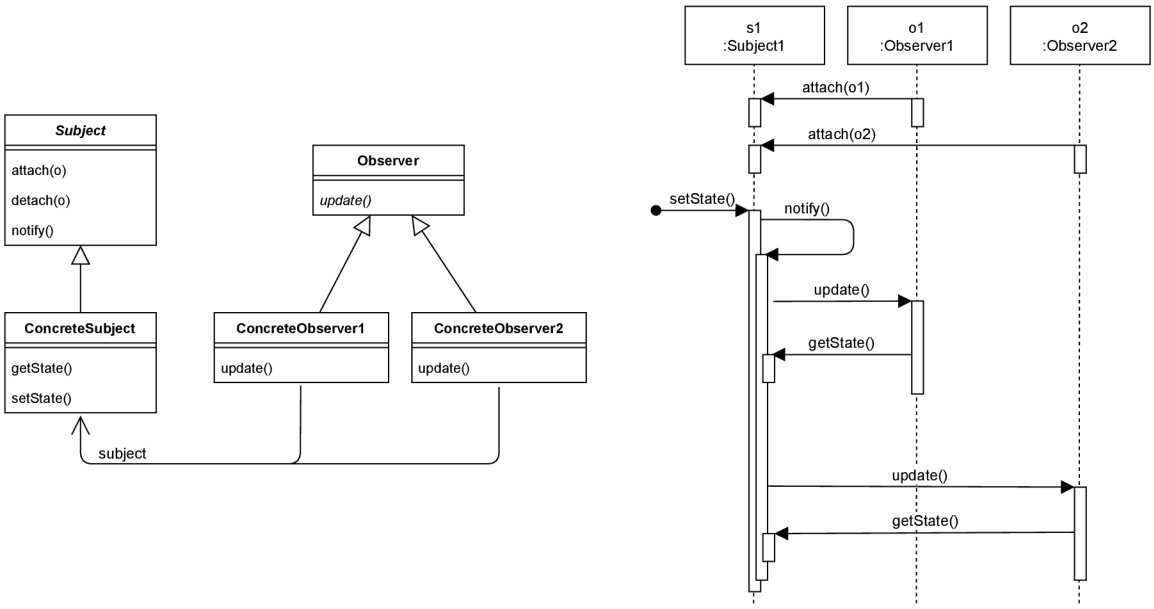
Vyjma krátkého úvodu k *událostem* tato kapitola pojednává zejména o technologiích používaných pro přenos souvislého toku dat v reálném čase a o komunikaci *peer-to-peer*. Vzhledem k povaze problému byla volba těchto technologií a jejich pokročilá znalost zcela zásadní. Jádrem celého řešení je *WebRTC*, což je relativně nový projekt, který používají i aplikace dnešních technologických gigantů, například *Google Meet*, *Facebook Messenger* nebo *Discord* [13]. O WebRTC bude hlavní část této kapitoly, uvedu však i několik základních poznatků k protokolu *RTP (Real-time Transport Protocol)*, kolem kterého bylo WebRTC vybudováno.

3.1 Události

V další části textu se budu často odkazovat na pojem událost, *event handler* případně *event listener*, proto nyní tyto pojmy objasním. Za událost může být považováno obecně cokoli, co může nastat během vykonávání programu, například akce uživatele, změna stavu na síťovém rozhraní nebo zpráva od operačního systému. Události nejčastěji vznikají v okolním prostředí programů a jejich vyvolání je asynchronní. Samotný výskyt události ovšem nepodmiňuje reakci daného programu – program může a nemusí na daný typ události reagovat. Využíváme-li události k chodu našeho programu, mluvíme o *událostmi řízené architektuře*.

Zpracování událostí je často implementováno prostřednictvím návrhového vzoru *observer pattern*. Problémem, který observer pattern [5] řeší, je: *„Definovat one-to-many závislost mezi objekty takovou, že pokud se stav jednoho objektu změní, všechny objekty na něm závislé jsou informovány a automaticky aktualizovány.“* Observer pattern definuje dvě třídy objektů – `Subject` a `Observer`. `Subject` je vlastním zdrojem událostí. Změní-li se stav objektu typu `Subject`, vyvolá se odpovídající událost, na základě čehož je `Observer` na změnu stavu upozorněn a může reagovat adekvátním způsobem. Aby `Observer` zprávu obdržel, musí se nejprve u objektu typu `Subject` registrovat, v diagramu na obrázku 3.1 je registrace znázorněna metodou `attach(o)`. Třídu `Observer` lze také chápat pod pojmem **event listener**. Ne vždy se však o pojmu event listener smýšlí jako o třídě objektů. Například v prostředí jazyka JavaScript je parametrem metody `addEventListener()` HTML DOM elementů pouze funkce, nikoli objekt. V jazyce Java je obvyklejší používat jako listenery objekty.

Event handler má pojmu event listener velmi blízko, jedná se ovšem o pouhou funkci, která má zpracování události na starosti. Event listener je chápán spíše jako celkový mechanismus zahrnující také naslouchání vzniku událostí. Například HTML DOM elementy



Obrázek 3.1: Diagram tříd a sekvenční diagram návrhového vzoru Observer Pattern [5].

mají obvykle řadu vlastností, kterým lze přiřadit vlastní funkci – event handler. Příkladem takových vlastností jsou `onclick`, `onfocus` nebo `onsubmit`.

3.2 Real-time Transport Protocol

RTP [17] je transportní protokol zajišťující přenos dat – například audia či videa – v reálném čase. Mezi služby RTP protokolu řadíme zejména rozlišování typu dat v paketech, číslování sekvencí, timestamping a monitorování doručení dat. RTP protokol funguje zpravidla nad protokolem UDP[1]. Protokol RTP sám o sobě tedy nezaručuje doručení paketů ve správném pořadí, avšak poskytuje informace, které mohou být použity k rekonstrukci tohoto pořadí. Mezi užitečné vlastnosti protokolu RTP lze zařadit:

- Zpravidla nízkou latenci.
- Pakety jsou sekvenčně číslovány. To umožňuje správné doručení paketů i na transportních vrstvách, které negarantují doručení paketů ve správném pořadí, případně doručení vůbec.
- Výše uvedené podněcuje užití protokolu RTP nad protokolem UDP.
- RTP není omezeno pouze na audiovizuální komunikaci, ale v zásadě lze využít v jakémkoliv spojitém datovém přenosu.

3.2.1 Quality of Service

Vzhledem k tomu, že protokol RTP je určený pouze k přenosu dat, nejčastěji se používá v kombinaci s protokolem *RTCP – RTP Control Protocol* [24]. Ten zpravidla poskytuje

[1]Protokol transportní vrstvy, který nezaručuje doručení paketu, nezachovává pořadí, ale díky nižší režii je zpravidla rychlejší než protokol TCP.

informace a statistiky spadající do kategorie *QoS – Quality of Service*. Mezi takové statistiky se řadí například počet přenesených paketů, ztráta paketu nebo zpoždění paketu. Tyto informace mohou být aplikací použity ke správnému nastavení parametrů datového toku. V případě přenášení video streamu lze například zvolit jiné rozlišení nebo video kodek. Druhou funkcí RTCP protokolu je poskytování informací o účastnících probíhající komunikace. Tyto informace mohou být v některých případech postačující, avšak často je implementován jiný protokol, který je pro řízení komunikace a kontrolu nad účastníky vhodnější.

3.3 WebRTC

WebRTC [15] je poměrně nová technologie umožňující širokou škálu *peer-to-peer*[2] komunikace, zahrnující například chatování, sdílení souborů, sdílení obrazovky, hry, audio hovory, video chat a další. Práce na standardizaci WebRTC začaly v roce 2011. Technologie je dostupná ve webových prohlížečích prostřednictvím aplikačního rozhraní[3], které je standardizované organizací *Wide Web Consortium (W3C)*. Standardizaci formátů a protokolů[4] používaných v jádru WebRTC má na starosti organizace *Internet Engineering Task Force (IETF)*.

S nástupem WebRTC se někdy mluví dokonce o revoluci webu navazující na *Web 2.0*, protože technologie umožňuje komunikaci v reálném čase přímo mezi webovými prohlížeči. WebRTC se ovšem neomezuje pouze na webové prohlížeče. Ruku v ruce s implementací v prohlížečích vznikají také standardní knihovny pro nativní užití v zařízeních *Android* a *iOS*, což ještě umocňuje škálu použitelnosti této technologie.

3.3.1 Navázání spojení mezi zařízeními

Jak již bylo zmíněno, jedním ze základních rysů WebRTC je komunikace peer-to-peer. Přesto je při navázání spojení nutné koordinovat výměnu informací mezi dvěma nebo více peery za pomocí dalších prostředků. Toto umožní peerům se vzájemně nalézt a ustavit mezi sebou přímý komunikační kanál. V rámci terminologie WebRTC vystupují tři základní typy serverů, které umožňují peerům ustavit spojení. Typy serverů nyní krátce uvedu a jejich roli ve WebRTC blíže popíši v následující části textu.

1. *Signalizační server* je jediný z typů, který je při navazování komunikace povinný. Jeho hlavním úkolem je zajistit peerům možnost *signalizace*, což spočívá v předávání informací například o síťovém okolí peera nebo o dostupných audio/video stopách.

2. *STUN (Session Traversal Utilities for NAT) server* je pomocný server, který pomáhá peerům překonat NAT[5] – jedná se o tzv. *NAT traversal*. Funguje v principu tak, že sdělí peerům jejich veřejné IP adresy, které jsou následně použity při signalizaci.

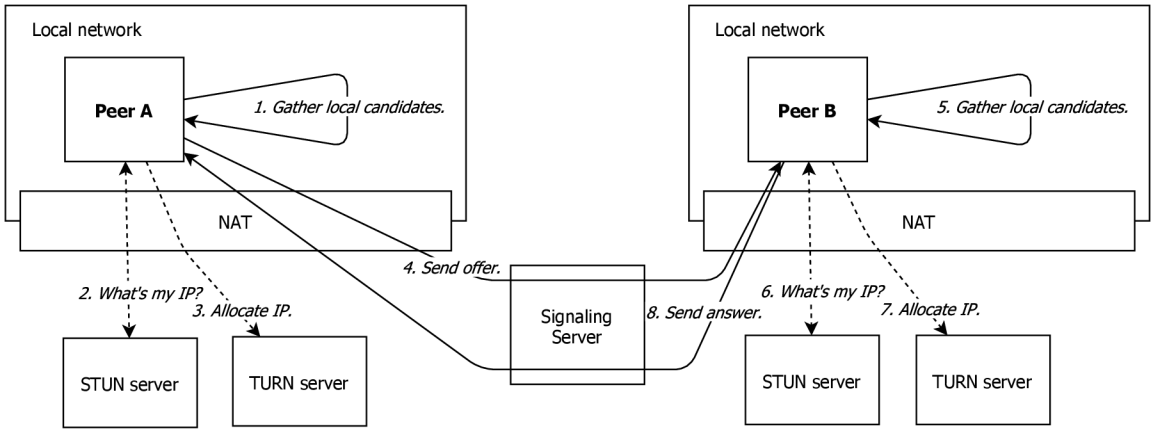
3. *TURN (Traversal Using Relays around NAT) server* je další pomocný server. TURN server nabourává koncept peer-to-peer, jelikož slouží k přeposílání paketů mezi oběma peery. Jeho použití není žádoucí, avšak někdy je nezbytné. Mechanismus, který zajišťují STUN servery může pro některé varianty NAT selhat – bude vysvětleno.

[2] Typ komunikace, kdy jsou data přenášena bezprostředně mezi dvěma komunikujícími subjekty, což je rozdílný koncept než například klient-server komunikace.

[3] https://www.w3.org/groups/wg/webrtc

[4] https://tools.ietf.org/wg/rtcweb/

[5] Network Address Translation – způsob úpravy IP adres, případně i portů na rozhraní lokální a veřejné sítě, zpravidla probíhá v routeru.



Obrázek 3.2: Hledání komunikačního kanálu mezi peery.

Jádrem navazování komunikace je *Interactive Connectivity Establishment (ICE)*. Jeho hlavním úkolem je překonání NAT, ale není to jeho jediná užitečná funkce. Pro účely popisu ICE protokolu je potřeba definovat pojem *ICE candidate*. ICE candidate je dvojice IP adresa, port a je potenciálním komunikačním prostředkem se vzdáleným peerem. Existují tři typy ICE candidate:

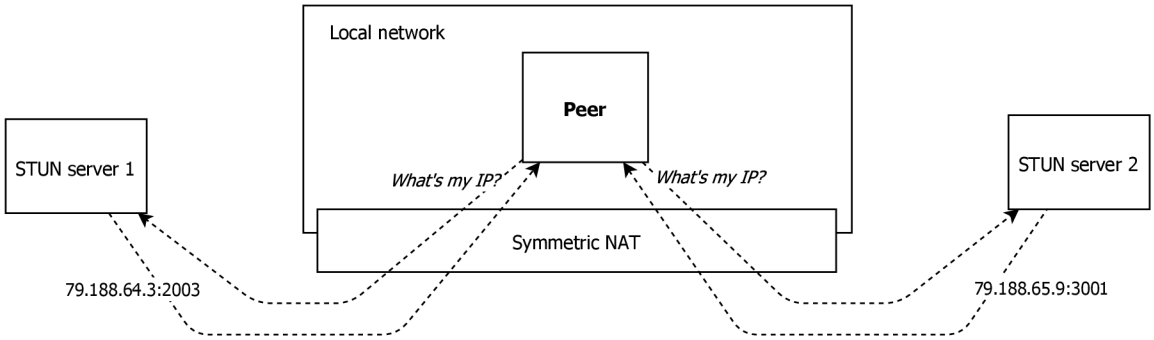
- Adresa přímo na síťovém rozhraní peera.
- Přeložená adresa na veřejné straně NAT – tzv. *server-reflexive adresa*.
- Adresa přidělená TURN serverem – tzv. *releayed adresa* [11].

Obrázek 3.2, vytvořený pro účely lepšího pochopení signalizačního protokolu, ukazuje dvě části navazování spojení mezi dvěma peery. V krocích 1-3 a 5-7 jsou získání odpovídající ICE candidates a v krocích 4 a 8 jsou zaslány – společně s dalšími informacemi – vzdálenému peerovi ve formě nabídky (offer) resp. odpovědi (answer). Pro výměnu informací, včetně těch o ICE candidates, byl vyhrazen termín *signalizace*. Pro tento účel musí být součástí WebRTC komunikace již zmíněný signalizační server, který funguje jako prostředník mezi peery před zahájením real-time komunikace.

V pozdějších vydáních WebRTC se objevuje také koncept *Trickle ICE*, který umožňuje asynchronní získávání ICE candidates, což může *v některých případech* výrazně urychlit zahájení komunikace mezi peery [10].

Nyní se vrátím k vysvětlení, proč je v některých případech nutné použít TURN server k přeposílání paketů mezi peery. Jedná se o problém související s typem NAT.

- *Full Cone NAT* mapuje všechny požadavky ze stejné IP adresy a portu na stejnou externí IP adresu a port. Jakýkoli vnější subjekt může posílat pakety vnitřnímu subjektu použitím externí IP adresy a portu.
- *Restricted Cone NAT* je stejný jako Full Cone NAT, avšak vnější subjekt s IP adresou **X** může posílat pakety vnitřnímu subjektu pouze pokud vnitřní subjekt předtím poslal paket na IP adresu **X**.
- *Port Restricted Cone NAT* je stejný jako Restricted Cone NAT, ale omezení zahrnuje také číslo portu. Nelze tedy zaslat vnitřnímu subjektu pakety z dané IP adresy a portu, pokud jsme na této IP adrese a portu předtím neobdrželi od vnitřního subjektu paket.



Obrázek 3.3: Ilustrace chování Symmetric NAT

- *Symmetric NAT* vytváří unikátní mapování na externí IP adresu a port pro každou dvojici vnitřní subjekt, vnější subjekt. Navíc pouze vnější subjekt, který obdržel paket od vnitřního může posílat pakety danému vnitřnímu subjektu.

Varianta Symmetric NAT nabourává myšlenku zjišťování veřejné IP adresy a portu, jelikož pro komunikaci s každým vnějším subjektem je třeba použít unikátní dvojici, což ilustruje diagram na obrázku 3.3. Pokud je peer skrytý za Symmetric NAT, nemůže spolehlivě sdělit svou externí adresu, což značně komplikuje hledání vhodných ICE candidates. Je-li navíc vzdálený peer skrytý za NAT typu Port Resticted Cone nebo Symmetric, nelze se vyhnout použití TURN serveru, který bude pakety mezi peery přeposílat. Zasláním STUN požadavku na jinou IP adresu může klient zjistit, zda se nachází za Symmetric NAT. Dále může pomocí příznaků požadavku nastavit, aby STUN server odpovídal z jiné adresy nebo portu než na který byl požadavek odeslán. Dostane-li na takové dotazy odpověď, lze odvodit, že se nachází za Full Cone NAT [22].

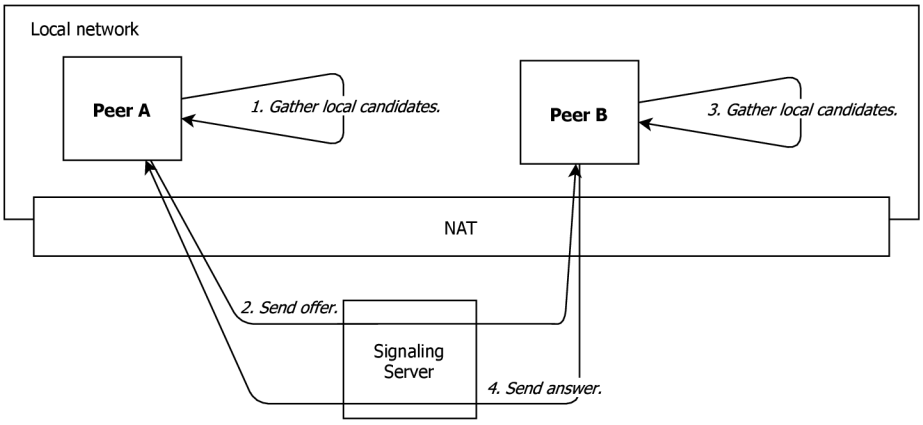
3.3.2 Spojení bez překonávání NAT

V předchozí části jsem popsal navazování komunikace co nejobecněji, nyní se pokusím více přiblížit tomu, jakým způsobem bude WebRTC použito v případě mého řešení. Z povahy problému totiž lze zajistit, že jednotliví peeři budou vždy v relativně nízké vzdálenosti od sebe, a proto je zajímavé sledovat, jakým způsobem je navazováno spojení v lokální síti, potažmo pomocí technologie *Wi-Fi Direct*.

V případě, že nepotřebujeme překonávat NAT, je navázání spojení poměrně přímočaré, jelikož odpadá komunikace se STUN a TURN servery. Pokud víme, že adresy peerů jsou vzájemně dosažitelné přímo z jejich síťových rozhraní, není třeba specifikovat STUN ani TURN server. V praxi to pak znamená velmi rychlé navázání spojení i bez využití konceptu Trickle ICE.

3.3.3 Signalizace

Doposud jsem se příliš nezabýval signalizačním serverem a jeho funkcemi podrobněji. Jak již bylo naznačeno, signalizační server je zodpovědný za předání různých typů informací jako například seznam audio/video stop nebo dříve zmínění ICE candidates. Je důležité zmínit, že signalizační server je v kontextu WebRTC chápán jako abstraktní pojem. WebRTC totiž nespecifikuje mechanismus, jakým mají být signalizační informace předány mezi peery. Signalizační server tedy nakonec nemusí být vůbec serverem v pravém slova smyslu, informace



Obrázek 3.4: Hledání komunikačního kanálu mezi peery v rámci privátní sitě

lze klidně zaslat emailem nebo i ručně přepsat. V praxi se často používá například *WebSocket*[6] mechanismus nebo varianta na bázi *HTTP protokolu*, například *HTTP polling*[7].

Veškerá signalizační data jsou předávaná ve formátu *Session Description Protocol (SDP)*. Jedná se o standardizovanou reprezentaci pro výměnu transportních adres, detailů o médiích a dalších *metadat*, popisujících jednotlivá sezení. Standard neobsahuje informaci o transportním protokolu, je tedy možné použít libovolný transportní protokol takový, aby vyhovoval konkrétnímu řešení. Toto zjevně koreluje s konceptem signalizačního serveru. Je důležité si uvědomit, že SDP je pouze formát dat. Účelem SDP není zajistit vyjednávání mezi účastníky, ale pouze umožnit standardní popis a sdělení signalizačních dat. Popis sezení pomocí SDP zahrnuje zejména následující:

- Název a účel sezení.
- Časový interval, po který je sezení aktivní.
- Média (audio, video, ...), která tvoří sezení.
- Informace potřebné k příjmu médií (adresy, porty, formáty, ...).

Budu-li konkrétnější, pomocí SDP lze rozlišovat média typu audio/video, specifikovat transportní protokol – například RTP+UDP+IP, nebo formát média (H.264, MPEG, ...). V neposlední řadě musí být také specifikovaná vzdálená adresa a port média, aby byl jeho příjem možný. Právě k tomuto účelu je důležitý proces zjišťování ICE candidates popsaný dříve [20].

3.3.4 Datové kanály

Vedle toku audiovizuálních dat v reálném čase obsahuje standard WebRTC také mechanismus pro posílání libovolných dat prostřednictvím *datových kanálů*. To, že jeden z účastníků navázaného spojení vytvoří datový kanál je indikováno vyvoláním události `datachannel` v `RTCPeerConnection` objektu vzdáleného peera. Ten má možnost nastavit datovému kanálu *event handler* pro každou z událostí:

[6] Komunikační protokol poskytující perzistentní spojení mezi klientem a webovým serverem.

[7] Technika, kdy klient pravidelně posílá dotazy na server s cílem získat nová data – v našem případě signalizační informace od vzdáleného peera.

- `open` – událost je vyvolána v okamžiku, kdy je transportní vrstva připravená přenášet datovým kanálem data,
- `message` – událost je vyvolána pro každou zprávu, která dorazí prostřednictvím datového kanálu a
- `close` – událost je vyvolána, pokud byl datový kanál uzavřen – například vyvoláním metody `close()` vzdáleným peerem.

Mezi obvyklé případy užití datových kanálů patří například předávání zpráv v textovém chatu nebo zasílání různých typů událostí. V mé práci jsem datové kanály využil zejména ke vzdálenému nastavení výstupu z kamery.

3.3.5 WebRTC na různých platformách

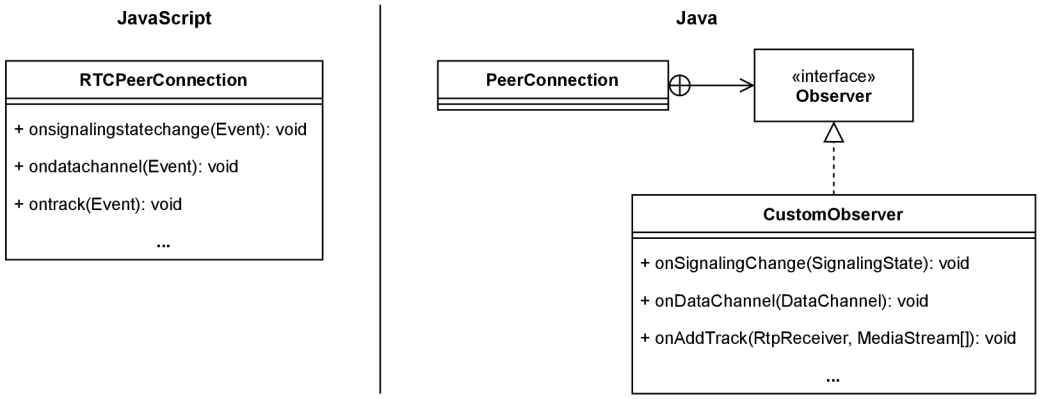
WebRTC se snaží umožnit vývoj bohatých vysoce kvalitních RTC aplikací pro prohlížeče, mobilní platformy a zařízení IoT[8] a umožnit jim komunikaci pomocí společných protokolů. Přes toto tvrzení považuji vývoj webových aplikací používajících WebRTC API za výrazně jednodušší proti vývoji na platformách Android nebo iOS, a to zejména z těchto důvodů:

- Webová varianta API[9] má velmi dobře zpracovanou dokumentaci včetně příkladů a popisu celé technologie. V případě vývoje nativních aplikací je třeba k pochopení API studovat zdrojové kódy API – často bez komentářů.
- Pro webovou variantu API existuje série ukázkových aplikací[10]. Lze tedy zkopírovat v podstatě hotový kód a použít ho správným způsobem pro vlastní aplikaci. Pro nativní knihovny neexistuje oficiální zdroj ukázkových příkladů, nejlepší varianta je inspirovat se v jiných projektech z různých zdrojů.
- Množství kódu, které je třeba napsat pro první funkční řešení ve webovém prostředí je výrazně menší oproti množství kódu v prostředí nativní mobilní aplikace. Proto považuji za nezbytné získat první kontakt s WebRTC prostřednictvím webového API.

Knihovna WebRTC je napsaná v jazyce C++. Nad touto knihovnou jsou dále vytvořena API pro webové aplikace (JavaScript), aplikace pro Android (Java) a aplikace pro iOS (Objective-C). Tato API jsou samozřejmě přizpůsobena možnostem konkrétního programovacího jazyka, obsahují však řadu společných rysů. Ve všech variantách je jádrem všeho objekt reprezentující spojení mezi dvěma peery – `(RTC)PeerConnection`. Dále mají všechna API svou reprezentaci pro přenášené médium – `VideoTrack` nebo `AudioTrack`, a také balíček pro operování s *datovými kanály*. Co se v jednotlivých API liší je přístup ke kameře a mikrofonu, jelikož jsou tato API používána v různých prostředích a operačních systémech.

Nyní se budu podrobněji zabývat průběhem komunikace po navázání spojení – tedy po nalezení vhodné dvojice lokálního a vzdáleného ICE candidate mezi kterými vzniká komunikační kanál. Zmíním zejména nejpodstatnější události, které během komunikace nastávají. Budu používat některé datové typy a názvy událostí z dokumentace W3C[11], tedy z webové varianty API.

[8] https://en.wikipedia.org/wiki/Internet_of_things
[9] https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
[10] https://webrtc.github.io/samples/
[11] https://w3c.github.io/webrtc-pc/ – k lednu 2021



Obrázek 3.5: Ilustrace rozdílu mezi JavaScript a Java WebRTC API.

Celé webové API je vybudováno okolo objektu třídy `RTCPeerConnection`. Objekt obsahuje řadu tzv. *event handler* funkcí (dále jen handler), což jsou vlastnosti objektu `RTCPeerConnection`. Tyto funkce jsou pak vyvolány v případě výskytu události svázané s daným handlerem. Jedním z užitečných handlerů třídy `RTCPeerConnection` je například `onconnectionstatechange`. Pokud této vlastnosti přiřadíme svou implementaci funkce, bude vyvolána při změně stavu spojení mezi peery. Můžeme tak reagovat na události navázání spojení, případně chyby při navazování apod. Dalším důležitým handlerem této třídy je `ontrack`, který je vyvolán, je-li v rámci spojení detekována nová audio či video stopa. Objekt stopy je předán v parametrech události, což umožní lokálnímu peerovi se stopou dále nakládat – například ji přehrávat v reálném čase, nahrávat pro pozdější využití apod. Dalšími typickými handlery třídy `RTCPeerConnection` jsou například `onsignalingstatechange` nebo `ondatachannel`.

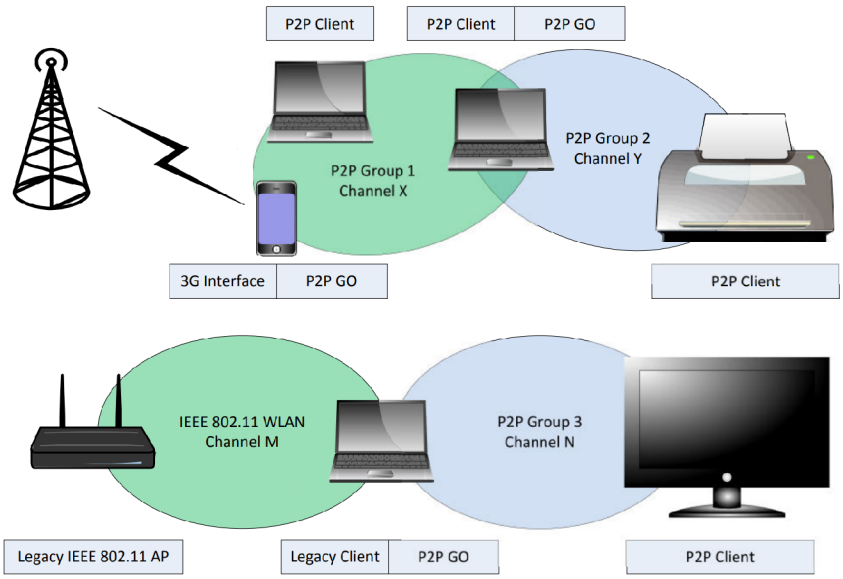
Pokud bychom pro ilustraci chtěli výše uvedené přirovnat k jazykovým konstrukcím v Java API, určeném pro platformu Android, zjistíme, že je zde středobodem objekt třídy `PeerConnection`. Poměrně zásadním rozdílem je, že ekvivalenty výše uvedených handlerů jsou v tomto případě definovány rozhraním `PeerConnection.Observer`. Toto rozhraní musí být implementováno vlastní třídou, jejíž objekt je použit při vytváření `PeerConnection` objektu. Java API je tedy v tomto méně flexibilní a často je vygenerován nepotřebný kód, jelikož ne vždy je třeba všechny handlery skutečně implementovat.

3.4 Wi-Fi Direct

Technologie *Wi-Fi Direct* má kořeny ve Wi-Fi standardu *IEEE 802.11*. Ve své podstatě umožňuje komunikaci dvou a více zařízení bez klasického *access pointu (AP)*. Typicky se klientské zařízení připojuje prostřednictvím AP do *WLAN sítě*[12]. V tomto modelu jsou před navázáním spojení jasně stanovené role – zařízení je buď AP nebo klient. Naproti tomu Wi-Fi Direct zařízení by mělo implementovat obě tyto role. Existují dokonce situace, kdy zařízení současně vystupuje jako AP i jako klient, například použitím různých frekvencí při komunikaci nebo časovým sdílením kanálu pomocí virtualizačních technik.

V rámci jednoho sezení Wi-Fi Direct hovoříme o tzv. *P2P Group Owner (P2P GO)* a *P2P Clients*. P2P GO zastává funkcionalitu access-pointu, group-owner je vybrán na

[12]Wireless Local Area Network – dvě nebo více bezdrátově komunikujících zařízení, která společně formují lokální síť.



Obrázek 3.6: Příklady podporovaných topologií a případů užití Wi-Fi Direct.

Vzdálenost	**Průměr [MB/s]**	**Odchylka [MB/s]**
0 m Wi-Fi P2P	4,105	0,110
0 m Wi-Fi	1,912	0,050
5 m Wi-Fi P2P	3,496	0,147
5 m Wi-Fi	1,906	0,051
10 m Wi-Fi P2P	2,215	0,098
10 m Wi-Fi	1,895	0,075

Tabulka 3.1: Výsledky měření propustnosti Wi-Fi a Wi-Fi P2P [23].

začátku komunikace v rámci *vyjednávání*. Jakmile je komunikace zahájena, mohou se do komunikační skupiny připojit další klienti, podobně jako do obecně známé WLAN [1].

Technologií Wi-Fi Direct má smysl se zabývat ze dvou důvodů. Protože se jedná o komunikaci typu peer-to-peer, nevnášíme další komplexity do WebRTC modelu. Platforma Android navíc zahrnuje podporu Wi-Fi Direct prostřednictvím balíčku *android.net.wifi.p2p*[13]. Narozdíl od jiných P2P technologií (například Bluetooth) poskytuje Wi-Fi Direct dostatečnou propustnost pro přenos multimediálních dat v reálném čase. Studie navíc ukazují, že v případě krátké vzdálenosti mezi komunikujícími zařízeními může být propustnost Wi-Fi Direct komunikace až dvojnásobná oproti běžně známé Wi-Fi s klasickým zařízením typu access point, což je důsledek komunikace přímo mezi zařízeními.

Implementace Wi-Fi Direct na platformě Android, jež je použita v této práci, je založena na objektu třídy `BroadcastReceiver`. `BroadcastReceiver` může být zaregistrován, aby přijímal všesměrové zprávy typu `Intent`. Pomocí objektu třídy `IntentFilter` lze při registraci specifikovat, které zprávy má konkrétní implementace `BroadcastReceiver` dostávat. V případě Wi-Fi Direct – na platformě Android se užívá spíše název Wi-Fi P2P – se jedná o 4 typy zpráv:

[13] https://developer.android.com/reference/android/net/wifi/p2p/package-summary

- `WIFI_P2P_STATE_CHANGED_ACTION` – Indikace, zda je Wi-Fi P2P rozhraní povoleno či zakázáno.
- `WIFI_P2P_CONNECTION_CHANGED_ACTION` – Indikace stavu spojení mezi jednotlivými peery včetně informace o tom, zda byla sestavena Wi-Fi P2P skupina a adresy group-owner zařízení. Příjem této zprávy je podmíněn povolením oprávnění s označením `ACCESS_FINE_LOCATION` a `ACCESS_WIFI_STATE`.
- `WIFI_P2P_PEERS_CHANGED_ACTION` – Indikace změny na seznamu dostupných zařízení. Této zprávě předchází volání metody `discoverPeers()` třídy `WifiP2pManager`. Před voláním metody a pro příjem této zprávy je nutné mít povolena oprávnění `ACCESS_FINE_LOCATION` a `ACCESS_WIFI_STATE`.
- `WIFI_P2P_THIS_DEVICE_CHANGED_ACTION` – Indikace změny parametrů tohoto zařízení, například změna názvu zařízení.

Některé čtenáře by mohlo překvapit, že technologie Wi-Fi Direct potřebuje ke svému chodu oprávnění `ACCESS_FINE_LOCATION`, tedy přístup k poloze zařízení. Ve skutečnosti zařízení opravdu nepotřebuje znát svou přesnou polohu, aby dokázalo skenovat své okolí. Důvod spočívá v existenci online databází, které obsahují data o přístupových bodech a jejich poloze – tyto databáze jsou tvořeny aplikacemi, které mají přístup k poloze[14]. Seznam okolních zařízení je pro danou oblast *unikátní*. Na základě informací z online databází a seznamu dostupných zařízení je tedy v některých případech možné určit polohu zařízení s přesností okolo 20 metrů. Skutečnost, že musí uživatel u aplikace, která s polohou vůbec neoperuje, potvrzovat a zapínat přístup k poloze, považuji za poměrně nešťastnou a špatný příklad uživatelské zkušenosti.

Jakmile je zaregistrován `BroadcastReceiver`, reagujeme na příchozí zprávy v handleru `onReceive(Context, Intent)`. Obvyklé akce, které pro jednotlivé zprávy vykonáváme jsou:

- `WIFI_P2P_STATE_CHANGED_ACTION` – V případě, že je Wi-Fi P2P rozhraní povoleno, spouštíme skenování okolí pomocí metody `discoverPeers()`. Skenování je třeba zahájit ve všech zúčastněných zařízeních, jinak nebudou zařízení vzájemně objevena.
- `WIFI_P2P_CONNECTION_CHANGED_ACTION` – V případě, že je spojení aktivní, vyžádáme si informace o Wi-Fi P2P skupině, například adresu group-owner zařízení. Po zjištění detailů o skupině je zpravidla možné zahájit P2P komunikaci mezi zařízeními. V případě, že je spojení neaktivní, můžeme provést navazující akce – například informovat uživatele o ztrátě spojení.
- `WIFI_P2P_PEERS_CHANGED_ACTION` – Na základě seznamu dostupných zařízení je třeba provést volbu, se kterým zařízením se chceme spojit. K dispozici máme **název zařízení** a **MAC adresu** síťového rozhraní zařízení. Volbu můžeme provést programově, často ale konkrétní zařízení volí uživatel na základě jeho názvu. Po provedení volby následuje volání metody `connect()` třídy `WifiP2pManager` pro navázání P2P spojení se vzdáleným zařízením.

[14] https://support.netanalyzer-an.techet.net/article/124-why-does-the-app-require-location-permission-for-wifi-signal-when-other-apps-dont

- `WIFI_P2P_THIS_DEVICE_CHANGED_ACTION` – Dle mého názoru poměrně nevýznamná událost, jejíž zpracování nemusí být nutně implementováno. Záleží však na konkrétním užití Wi-Fi Direct technologie.

Zmínil jsem metodu `connect()` – ta je volána s důležitým konfiguračním parametrem. Parametr kromě MAC adresy vzdáleného zařízení obsahuje také vlastnost `groupOwnerIntent`, která udává pravděpodobnost, že bude zařízení group-owner vytvořené Wi-Fi P2P skupiny. Zde si dovolím poznamenat, že Android dokumentace pro `groupOwnerIntent` přímo používá slovní obrat *„... sklon k tomu stát se group-owner zařízením ...“*. Vzhledem k tomu, že skutečnost, kdo se stane group-owner zařízením je často zcela zásadní ve vztahu k použitým komunikačním protokolům považuji obrat *„sklon k“* za velmi nešťastný. Vývojář by měl mít možnost s jistotou zvolit, které ze zařízení bude group-owner.

Další detaily v souvislosti s podporou Wi-Fi Direct ve WebRTC knihovně pro Android a signalizací pomocí Wi-Fi Direct zmíním v kapitole 7.1.2.

3.5 YUV formát

V rámci mé práce bylo nutné zjistit více o YUV formátu, který knihovna WebRTC používá pro vytváření a práci se snímky videa. YUV je formát rastrových obrázků založený na třech složkách – **jasu** a dvou **barevných složkách**. Narozdíl od známého formátu RGB jsou jednotlivé komponenty uloženy pro každý snímek odděleně (nejsou vzájemně prokládány).

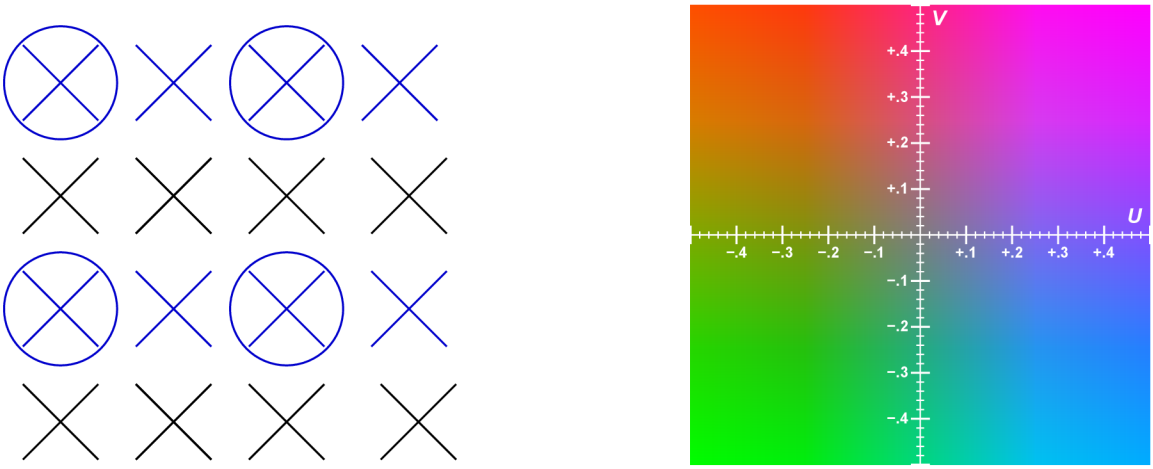
- Písmeno Y reprezentuje *jas*. Nejčastěji je pro jas vyhrazeno 8 bitů na pixel, velikost plochy Y v bajtech tedy odpovídá velikosti snímku v pixelech.
- Písmeno U reprezentuje *projekci modré barvy*.
- Písmeno V reprezentuje *projekci červené barvy*.

Barevné složky bývají často vůči jasové složce podvzorkovány, vzájemně však musí mít stejnou mohutnost. Pokud si výše uvedené složky grafu představíme skutečně jako plochy velikosti obrázku, můžeme hovořit o *horizontálním* a *vertikálním* podvzorkování. Pro definici hodnoty podvzorkování se poté užívá notace $A : B : C$, kde $A : B$ reprezentuje hodnotu horizontálního podvzorkování a $B : C$ hodnotu vertikálního podvzorkování. Nejedná se ovšem o klasický poměr, například pro formát YUV420 jsou barevné složky podvzorkovány v poměru 2:1 jak v horizontálním, tak ve vertikálním směru [6]. Toto detailněji ukazuje obrázek 3.7. Důležitým parametrem v mé práci byl počet bitů na jeden pixel. Pro formát YUV420, se kterým výsledná aplikace operuje, je tato hodnota 12 bitů na pixel.

3.6 Google Play Console

V průběhu vývoje mobilní aplikace **Mirrorer** byly jednotlivé verze aplikace sdíleny s testery prostřednictvím platformy Google Play Console. Ta slouží jako nástroj, kterým vývojář získá kontrolu nad životním cyklem celého vývoje. Mimo jiné umožňuje definovat 4 typy vývojových větví – interní testování, uzavřené testování, otevřené testování a produkční větev. Pro účely testování aplikace Mirrorer bylo použito uzavřené testování. To umožňuje specifikovat seznam testerů – prostřednictvím jejich účtu Google – kteří budou mít k aplikaci přístup. Testeři pak mohou cílovou aplikaci snadno instalovat a aktualizovat pomocí dobře známé aplikace Google Play[15]. Uzavřené testování je preferovanou metodou pro získávání

[15] https://play.google.com/store



Obrázek 3.7: **Vlevo** je vizuální reprezentace podvzorkování formátu YUV420. Každý kříž-ek reprezentuje jeden bajt jasové složky Y a jejich počet odpovídá počtu pixelů obrázků (nejsou podvzorkovány). Kružnice pak reprezentují jeden bajt libovolné barevné složky. Modré řádky ukazují, že horizontální podvzorkování je skutečně v poměru 2:1 – notace 4:2 říká, že v horizontálním směru jsou 2 bajty barvy na 4 bajty jasu. Vertikální podvzorkování je také v poměru 2:1 – notace 2:0 říká, že ve vertikálním směru je na 2 bajty barvy na lichých řádcích 0 bajtů barvy na sudých řádcích. **Vpravo** vidíme chromatickou plochu pro hodnotu $Y = 0{,}5$.

cílené zpětné vazby od uživatelů dle dokumentace platformy Android[16]. K dalším funkcím Google Play Console patří například:

- Nastavení aplikace – dostupnost v různých zemích, informace o obsahu (např. zda je aplikace vhodná pro děti), popis aplikace, snímky aplikace a další.
- Hlášení výkonu aplikace – například selhání aplikace.
- Zobrazení statistik aplikace – počet instalací, počet akvizicí, průměrné denní hodnocení a další.
- Nastavení monetizace – cena aplikace, produkty v aplikaci, propagační kódy a další.

[16] https://developer.android.com/distribute/best-practices/launch/test-tracks

Kapitola 4

Přístupy k vývoji mobilních aplikací

Mobilní aplikace jsou softwarovým řešením určeným pro mobilní telefony, tablety a v poslední době také tzv. *wearables*[1], které zažívají velký rozmach, především jako doplněk k jiným mobilním zařízením. Dvě nejrozšířenější platformy pro distribuci mobilních aplikací – Google Play a App Store – dnes obsahují miliony aplikací sloužící různým účelům. Mobilní aplikace, na rozdíl od jiných softwarových řešení, bývají často zdarma, případně dostupné za relativně malý poplatek [14]. Jedním z rysů mobilních aplikací je rychlá dostupnost a instalace, stejně jako snadné odstranění aplikace [19].

4.1 Porovnání přístupů

Existují tři hlavní přístupy k vývoji mobilních aplikací. Každý z nich má své silné a slabé stránky, a proto je třeba na začátku vývoje zvážit, který z nich bude danému účelu nejvíce vyhovovat.

4.1.1 Nativní aplikace

Jedná se o nejznámější způsob tvorby mobilních aplikací. Hlavním znakem tohoto přístupu je, že vývojáři se přizpůsobují dané platformě v tom smyslu, že svůj kód píší v programovacím jazyce, který daná platforma podporuje. Například *Objective-C* nebo *Swift* pro *iOS*, případně *Java* a *Kotlin* pro *Android*. Android a iOS jsou operační systémy úzce vázané na daný typ mobilního zařízení. Mezi výhody tohoto přístupu patří především fakt, že veškerá *API*[2] a další funkce dané mobilní platformy jsou dostupná přímo z vývojového prostředí. To může zásadně ovlivnit nejen schopnost využít optimálně všechny potřebné schopnosti daného zařízení, ale také *výkonnost* výsledného řešení, která bude v případě nativních aplikací bezesporu nejvyšší. Vývojové prostředí je také často vytvořeno přímo k účelu tvorby mobilních aplikací, a tedy obsahuje všechny nástroje potřebné k vývoji, testování a nasazení aplikace. Hlavní nevýhoda nativních aplikací vyplývá z toho, jak úzce jsou spjaty s danou platformou, což v praxi znamená, že vývoj pro N platforem nás bude stát N-krát více prostředků než vývoj pro jednu platformu. Pokud tedy plánujeme nabídnout aplikaci

[1] Malá elektronická zařízení, která lze nosit přímo na lidském těle, kde mimo jiné mohou sledovat a zpracovávat různé tělesné signály. Nejrozšířenějším typem jsou *chytré hodinky*.

[2] *Application Programming Interface* – soubor funkcí, tříd, protokolů a dalších prostředků, které společně definují, jak lze daný software používat, jaké používat formáty vstupů, co lze očekávat na výstupech aj.

Platforma	Počet aplikací	Počet stažení
App Store	2,2 mil.	140 mld.
Google Play	2,8 mil.	82 mld.
Windows Store	669 000	–
BlackBerry World	245 000	4 mld.

Tabulka 4.1: Statistiky největších platforem pro distribuci mobilních aplikací k roku 2018 [2].

široké veřejnosti, bude vývoj poměrně drahý, jelikož pravděpodobně budeme chtít pokrýt minimálně dvě největší platformy.

4.1.2 Mobilní webové aplikace

Mobilní webové aplikace jsou načítány prostřednictvím webového prohlížeče nainstalovaném v daném zařízení. Svého času se jednalo o jedinou možnost, jak vytvořit *third-party*[3] aplikace pro platformu iOS, neexistoval ani App Store – toto trvalo první rok od vydání prvního zařízení iPhone v červnu 2007. Hlavní předností tohoto způsobu vyvíjení mobilních aplikací je, že se jedná o multiplatformní přístup, jelikož standardní webové aplikace jsou všechny psané v jazyce (např. v *JavaScriptu*), který je aplikací standardu *ECMAScript*. Díky tomu mohou webové prohlížeče sloužit jako *relativně* jednotný interpret mobilní webové aplikace, což zjednodušuje zpřístupnění aplikace velkému množství uživatelů. Druhým faktorem, který ještě zrychluje vývoj webových aplikací je jednoduchost použitých technologií oproti složitějším nativním programovacím jazykům a jejich knihovnám. Nutné je také zmínit jednoduchost vydávání nových verzí nebo oprav, což v případě webových aplikací spočívá v prostém nahrání potřebných souborů na webový server. Mezi nevýhody tohoto přístupu řadíme zejména *nedostupnost* všech schopností mobilních zařízení (myšleno obecně – některá API jsou z prohlížečů dostupná). Co se týče *doručení* aplikace koncovým uživatelům – není běžné, aby uživatel hledal novou aplikaci zadáním URL do webového prohlížeče. Třetí nevýhodou je nižší výkonnost než v případě nativních aplikací, jelikož je kontext webového prohlížeče další vrstvou, která leží mezi uživatelem a danou platformou.

4.1.3 Hybridní aplikace

Princip *hybridních aplikací* spočívá ve využití komponenty, která bývá součástí jednotlivých platforem a je ve své podstatě webovým prohlížečem, vestavěným přímo do nativní aplikace. Tato komponenta se často nazývá *WebView*. Na hybridní aplikace lze tedy nahlížet jako na nativní aplikace, které pro interakci s uživatelem využívají především WebView. Ten tedy zpřístupní uživateli jádro aplikace psané webovými technologii a používá *wrapper* pro komunikaci s nativními technologiemi a různorodými API dostupnými na zařízení. Hlavní část kódu aplikace může být tedy sdílená napříč platformami, což je hlavní výhoda, kterou hybridní aplikace zdědily od webových aplikací. Zároveň je však potlačena nedostupnost některých nativních funkcí, protože hybridní aplikace má přístup ke schopnostem daného zařízení v plném rozsahu. I tento přístup má svá úskalí, výkonnost hybridních aplikací nemusí být optimální, jelikož hlavní část aplikace je opět vykonávána webovými technologii namísto nativního přístupu [2]. Druhou nevýhodou je absence grafických komponent dané

[3] Aplikace vyvíjené subjekty spadající mimo danou platformu.

platformy. V dnešní době ovšem existuje řada webových řešení, která tato grafická řešení dokáží zčásti nebo zcela nahradit jako například *Angular Material* nebo *Ionic UI Components*. Také záleží na úhlu pohledu, jelikož sdílením kódu definující uživatelské rozhraní snadněji docílíme jednotného designu uživatelského rozhraní napříč všemi zařízeními [7].

4.2 Volba přístupu a platformy

V úvodních fázích vývoje jsem experimentoval s WebRTC knihovnou ve webovém prostředí, jelikož dokumentace pro nativní verze knihoven jednoduše neexistuje. Vznikla dokonce i první použitelná webová verze, která uměla přenášet video stopu mezi prohlížeči.

Při úvahách, zda se vydat cestou webové či nativní aplikace bylo důležité předem odhadnout náročnost celého řešení z hlediska nutného výpočetního výkonu a přístupu k hardwarovým rozhraním daného zařízení, zejména pak ke kameře. Přestože se webová řešení stávají v posledních letech více a více konkurenceschopná nativním aplikacím, rozhodl jsem se nakonec vydat cestou nativní aplikace pro operační systém Android, a to z následujících důvodů:

- K dispozici je plný výkon daného zařízení bez kompromisů, což v případě přehrávání a obecně práce s video stopami považuji za velmi vhodné.
- Je možný přímý přístup k operační paměti a také k souborovému systému zařízení.
- Existence WebRTC knihovny pro Android, která obsahuje obdobnou funkcionalitu, jako webová varianta.
- Vlastní zkušenost s platformou, zkušenost v programováním v jazyce Java.

Minimální podporovaná verze systému android je verze 5.0 – Lollipop (verze vydána v roce 2014). Tato verze přinesla následující vylepšení, kterých mohlo být potenciálně využito při vývoji výsledné aplikace:

- Vylepšený výstup z kamery[4]. Nově je možno zaznamenávat snímky ve formátu YUV v režimu 30 FPS.
- Podpora rozhraní **OpenGL ES 3.1** pro práci s grafikou[5].
- Podpora nahrávání obrazovky zařízení[6].

Podpora starších verzí operačního systému Android nepřicházela v úvahu i z toho důvodu, že verze 5.0 je aktuálně také minimální verze pro užití WebRTC knihovny pro Android.

[4] https://developer.android.com/about/versions/lollipop#Camera
[5] https://developer.android.com/about/versions/lollipop#Graphics
[6] https://developer.android.com/about/versions/lollipop#ScreenCapture

Kapitola 5

Uživatel jako součást vývoje

Se zvyšujícími se nároky uživatelů na snadnou interakci s dnešními technologiemi vznikla řada konceptů, které zapojují uživatele přímo do procesu vývoje. Jedním z takových konceptů je *User-Centered Design (UCD)*. UCD věnuje zvýšenou pozornost potřebám, požadavkům a omezením koncových uživatelů produktu v každé fázi procesu vývoje. Na rozdíl od jiných metod se snaží optimalizovat produkt na základě toho, jak uživatelé mohou, chtějí a potřebují produkt používat místo toho, aby se snažil změnit chování uživatelů a přinutit je přizpůsobit se produktu [4].

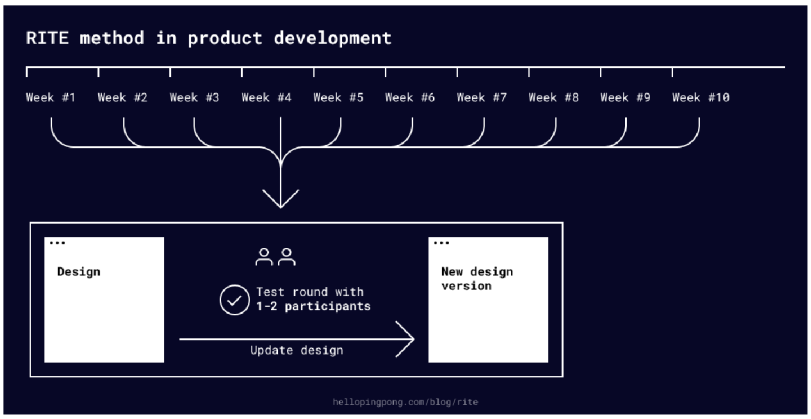
Použití přístupu, ve kterém je centrem všeho koncový uživatel má značné ekonomické i sociální výhody. Aplikace s vysokou úrovní použitelnosti jsou často úspěšnější jak technicky, tak i komerčně. Uživatelé jsou u takových produktů ochotni častěji si zaplatit nadstavbovou funkci, navíc jsou sníženy nároky na uživatelskou podporu. Mezi některé konkrétní výhody UCD patří:

- produkt je použitelný pro uživatele, jejichž technické dovednosti jsou na různé úrovni, což zpřístupní produkt více uživatelům,
- zlepšení uživatelské zkušenosti,
- snížení nepohodlí a stresu,
- zvýšení konkurenceschopnosti, například budováním značky (*brand image*) [9].

5.1 Uživatelské testování

Uživatelské testování je jednou z metod získávání zpětné vazby od uživatelů. Řeší otázku *intuitivní* interakce uživatele s uživatelským rozhraním a to zejména z pohledu uživatele, který často na produkt nahlíží a používá jej odlišným způsobem než jeho autoři.

Existuje několik druhů uživatelských testů. Testování se dělí na *moderované* a *nemoderované*. Jak již název napovídá, v případě moderovaného testování je u testování přítomný moderátor, který dohlíží na uživatele, může jim poskytovat zpětnou vazbu a provádět je procesem testování. Výhodou moderovaného testování je, že lze jít více do hloubky a zjistit od uživatelů více informací – například co by v dané situaci očekávali apod. Moderované testování je vhodnější, pokud pracujeme s menší testovací skupinou. Výhodou nemoderovaného testování je možnost získat zpětnou vazbu od velkého počtu uživatelů v krátkém čase.



Obrázek 5.1: Metoda RITE v rámci vývoje [12].

Guerilla testing je druh testování, kdy jsou účastníci vybráni náhodně na veřejně dostupných místech například v obchodním centru. Jedná se o rychlou a relativně levnou cestu, jak získat větší množství výsledků. Podobnou metodou je *laboratorní testování*, kdy mají všichni uživatelé relativně stejné podmínky, což může zlepšit vypovídající hodnotu získaných výsledků. Takové testování lze také pozorovat z pohledu třetí osoby, která může zaznamenávat detaily z průběhu testovacího sezení.

Nyní ještě k základním přístupům použitým v rámci interakce s uživatelem [21]:

- *Card sorting* metoda spočívá ve vytvoření kartiček s typem obsahu, který by se měl objevit ve výsledném produktu (např. webová aplikace). Uživatel se pak snaží zařadit kartičky do vlastních nebo předem připravených kategorií. Tato metoda se používá v raných fázích vývoje, což dá vývojářům základní představu o tom, jak by měl vypadat layout a navigační struktura aplikace.

- *A/B testing* je založený na porovnávání dvou různých verzí aplikace z důvodu zjištění, která z nich dává lepší výsledky. Používá se v případech, kdy chceme optimalizovat některé uživatelské procesy. U této metody je důležité testovat s větší skupinou uživatelů.

- *5-second test* klade důraz na první dojem – například, která část layoutu přijde uživatelům nejdůležitější, kterého tlačítka si všimli jako první apod.

5.1.1 Rapid Iterative Testing and Evaluation

RITE je praktická metoda uživatelského testování, která se zaměřuje na *rychlost*, *efektivitu* a *pravidelnost*. RITE se zaměřuje na popis procesu testování jako celku. Nezabíhá příliš do hloubky, pokud se bavíme přímo o průběhu jednotlivých testů s uživatelem.

Nejdůležitějšími aspekty této metody jsou rychlost a iterování. Uživatelské testování je opakováno zpravidla jednou týdně, ale lze zvolit i dvojnásobnou frekvenci testování, pokud jsme schopni zajistit rychlou změnu produktu. S ohledem na vysokou frekvenci testování je dostačující testovat například i s jedním nebo dvěma účastníky.

Problémy, které se při testování projeví lze klasifikovat do čtyř kategorií:

- Kategorie 1: **Jasná příčina a řešení, které lze snadno implementovat** – například změna textu, přeznačení tlačítek, přepsání dialogového okna atd.,

- Kategorie 2: **Jasná příčina a řešení, které je implementačně náročnější** – složitější funkcionalita, zásadní změny vzhledu či *codebase*,
- Kategorie 3: **Nejasná příčina a řešení** a
- Kategorie 4: **Jiné příčiny** – například problémy s testovacím skriptem nebo v interakci s uživatelem.

Opravy problémů z první kategorie jsou implementovány okamžitě a jsou tedy dostupné v příští iteraci testování. Opravy problémů druhé kategorie jsou zahájeny a jsou dostupné v příští nebo některé z dalších iterací. Pro problémy z kategorií 3 a 4 se snažíme získat další informace v nadcházejících iteracích a pokoušíme se je přesunout do první či druhé kategorie [18].

5.1.2 Minimum viable product

Pokud je prioritou, aby náš produkt – například mobilní aplikace – co nejvíce odpovídal představám koncových uživatelů, je vhodné vytvořit velmi jednoduché (levné) řešení, které bude obsahovat pouze nejmenší možnou funkcionalitu. Takové řešení nazýváme *minimum viable product*, což je první verze produktu, která by měla projít iterací uživatelského testování. Často je MVP využíván jako rozhodující faktor, zda má smysl pokračovat ve vývoji produktu. Uživatelské testování s MVP vytváří první zpětnou vazbu, a tedy udává směr, jakým se bude případný vývoj ubírat. MVP by neměl být chápán jako polovičaté řešení nízké kvality, naopak by měl ukázat hlavní smysl produktu a jeho přidanou hodnotu pro koncové uživatele v co nejlepším světle.

Jako jeden příklad za všechny může posloužit MVP cloudového úložiště *Dropbox*, kterým nebylo nic víc než prosté video vysvětlující, jak by Dropbox mohl v budoucnu fungovat. Původní autoři Dropboxu nalezli tedy snadnou a rychlou cestu, jak zjistit, zda o jejich produkt vůbec bude zájem.

5.2 Uživatelská zkušenost

Pojem *uživatelská zkušenost* – dále *UX* – je často chápán špatně či neúplně. Není to zdaleka o tom, že produkt má či nemá uživatelskou zkušenost. Každé uživatelské rozhraní má určitou úroveň UX. To, že UX neřešíme, neznamená, že výsledek bude bez UX, ale pravděpodobně její úroveň bude velmi nízká. Dalším důležitým poznatkem je, že UX neznamená popis prožitku nebo nálady uživatele. UX design je proces, který zahrnuje výzkum v podobě sledování uživatelů, jejich potřeb, rozvíjení nápadů, jejich aplikaci a měření v reálném světě.

5.2.1 Hlavní pilíře UX

Přestože je UX velmi komplexní disciplína, lze ji popsat pomocí pěti základních pilířů.

1. Psychologie. UX design často zahrnuje práci se subjektivními myšlenkami a pocity uživatele. Často je třeba se odpoutat od vlastního názoru na věc a více sledovat myšlenky uživatelů. Jakou má uživatel motivaci produkt používat? Co očekává po kliknutí na tlačítko? Je to něco co bude chtít udělat znovu? To jsou některé z mnoha otázek, které si je třeba položit při dělání UX designu.

2. Použitelnost. Na rozdíl od psychologie se jedná zpravidla o vědomý prožitek uživatele. Uživatel ví, když je něco obtížné nebo matoucí. Jako UX designéři bychom se měli snažit předcházet co nejvíce chybám, které mohou uživatelé udělat, pracovat v souladu s jejich očekáváním a používat *obvyklá* řešení.

3. Design. V UX designu je důležitější *funkčnost* designu než umělecká a líbivá řešení. UX design má za úkol podpořit důvěru uživatele v produkt, vést uživatele na správná místa a zajistit jednotnost a celistvost produktu.

4. Copywriting. Opět je nejdůležitější si uvědomit rozdíl mezi psaním textů, které mají podpořit značku a psaním textů, které vedou uživatele co nejjednodušší a nejpřímější cestou k výsledku. Pouze v druhém případě se jedná o UX design. Copywriting používáme, abychom uživateli předali podstatnou informaci, případně zmírnili jeho obavy z provedení některé akce.

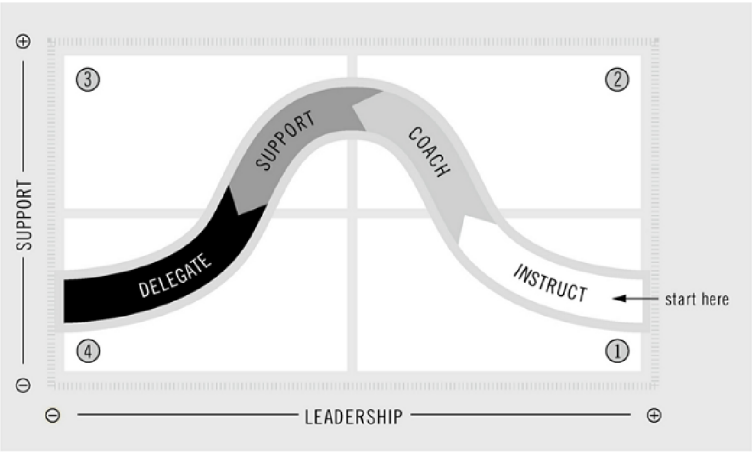
5. Analýza. Hlavní rozdíl proti jiným druhům designu. Analýzu je třeba dělat objektivně, nesnažit se dokázat *svou pravdu*, ale zjistit, co je opravdu funkční a naopak, co může design limitovat. Základem je shromáždit potřebná data a objektivně je interpretovat. Nelze si *domýšlet*, proč uživatelé něco dělají, je potřeba to *zjistit* [16].

Často skloňovaným pojmem v souvislosti s UX designem je *intuice* nebo *intuitivní design*. Vysoká míra intuitivnosti by měla být samozřejmostí, neexistuje důvod k tomu vytvářet něco, co není intuitivní. Je třeba si uvědomit, že v případě intuitivnosti se jedná o subjektivní pocit konkrétního uživatele. Co přijde intuitivní designérovi se nemusí shodovat s tím, co považuje za intuitivní většina uživatelů z cílové skupiny. *Cílovou skupinu* zmiňuji záměrně, nelze vytvářet intuitivní UX design bez správně stanovené cílové skupiny včetně definování *personas*[1].

Dále je třeba zmínit, že intuici (lze chápat i jako *instinkt*) získáváme v průběhu života. Intuice vychází z našich zkušeností – očekáváme něco na základě toho, co jsme předtím zažili. Lidé z různých věkových skupin, oborů nebo kultur mohou mít velice rozdílné instinktivní chování. Výše uvedené lze podtrhnout jednou z dostupných definicí intuice [3]: *„Intuition is compressed experience.“*

Pokud je úroveň intuitivnosti příliš nízká, vzniká něco, čemu se říká *unlearnable design*. S tímto se setkáváme při běžných činnostech i mimo oblast IT technologií. Jedná se o činnosti, při kterých relativně často opakujeme stejnou chybu, která je způsobená nevhodně zvoleným designem. Příklady nevhodného designu ukazuje například obrázek 5.2.

[1] Persona = smyšlená postava reprezentující typ uživatelů s podobnými vlastnostmi, kteří by mohli používat produkt podobným způsobem



Obrázek 5.2: Neintuitivní graf – nejvyšší hodnota je vlevo dole [25]

Kapitola 6

Návrh a vývoj aplikace

V této kapitole popíši základní model a hlavní komponenty, které bylo třeba implementovat. Vedle toho také zmíním některé implementační detaily, které bylo třeba vyřešit v průběhu vývoje.

6.1 Konceptuální model

Základními dvěma entitami mého konceptuálního modelu je člověk – sportovec, který provádí libovolné koordinační cvičení a zařízení s operačním systémem Android a aplikací *Mirrorer*, která vznikla jako výsledek této práce.

Jak ukazuje diagram na obrázku 6.1, Android zařízení může vystupovat ve třech různých režimech, kterými jsou:

- **Režim zrcadlo.** Tento režim lze využít v případě, že máme k dispozici pouze jedno zařízení. Obsahuje základní funkcionalitu pro nastavení a zobrazení výstupu z kamery. Režim je také vhodný k testování nové funkcionality, která nesouvisí s komunikací mezi zařízeními.

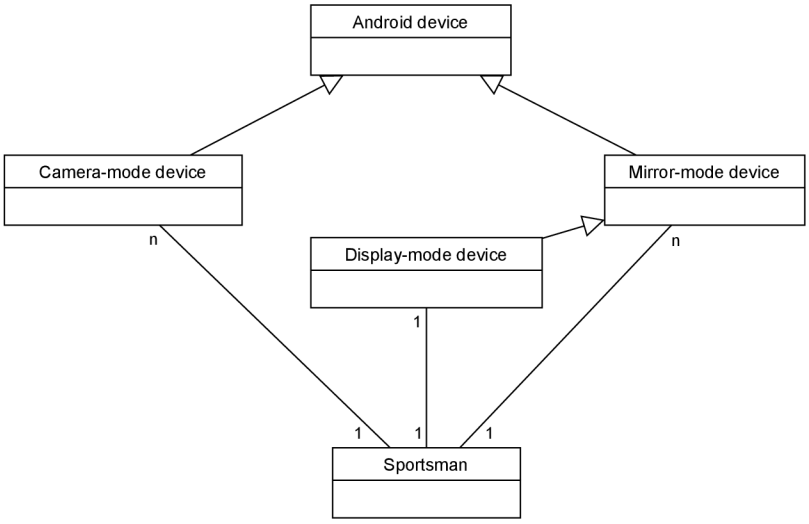
- **Režim kamera.** Zařízení v tomto režimu přistupuje ke kameře, zaznamenává vizuální výstup a odesílá ho v reálném čase do zařízení v režimu *displej*. Zároveň přijímá zprávy obsahující příkazy například k nastavení kamerového profilu.

- **Režim displej.** Zařízení v režimu displej kromě zrcadlení vlastního kamerového výstupu také přijímá data v reálném čase ze zařízení v režimu kamera. Dále je schopno odesílat zprávy s příkazy pro kamerové zařízení. Jedná se o rozšíření režimu zrcadlo, kdy přibývá další video stopa, kterou zařízení přijímá na svém síťovém rozhraní.

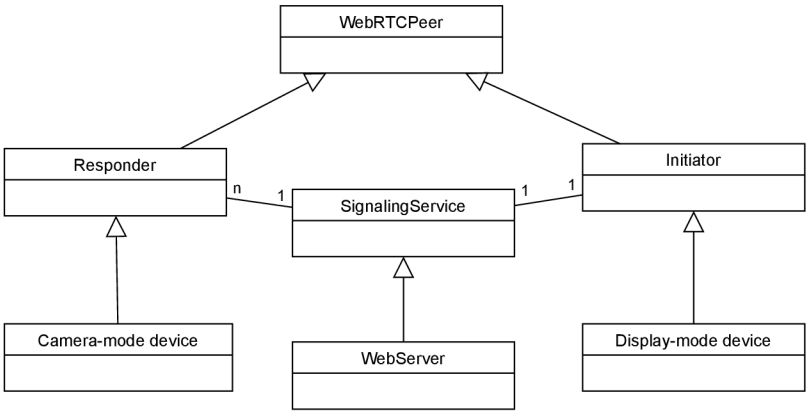
Obrázek 6.2 dále znázorňuje vztah režimů zařízení k signalizačnímu protokolu WebRTC. Pro zachování jednoduché architektury jsem se rozhodl, že zařízení v režimu displej bude vždy iniciovat signalizaci SDP protokolem, a tedy vytváří SDP zprávu typu offer. Naproti tomu kamerové zařízení v reakci na tyto zprávy vytvoří SDP zprávu typu answer.

Pokud by tedy řešení v budoucnu obsahovalo více kamerových zařízení, nezpůsobí toto nabourání stanoveného konceptu, iniciátor zůstane stále jeden.

Další důležitou entitou pro sestavení komunikace je signalizační server – v našem případě webový server, který implementuje REST API určené k výměně signalizačních informací.



Obrázek 6.1: Hlavní entity mého řešení



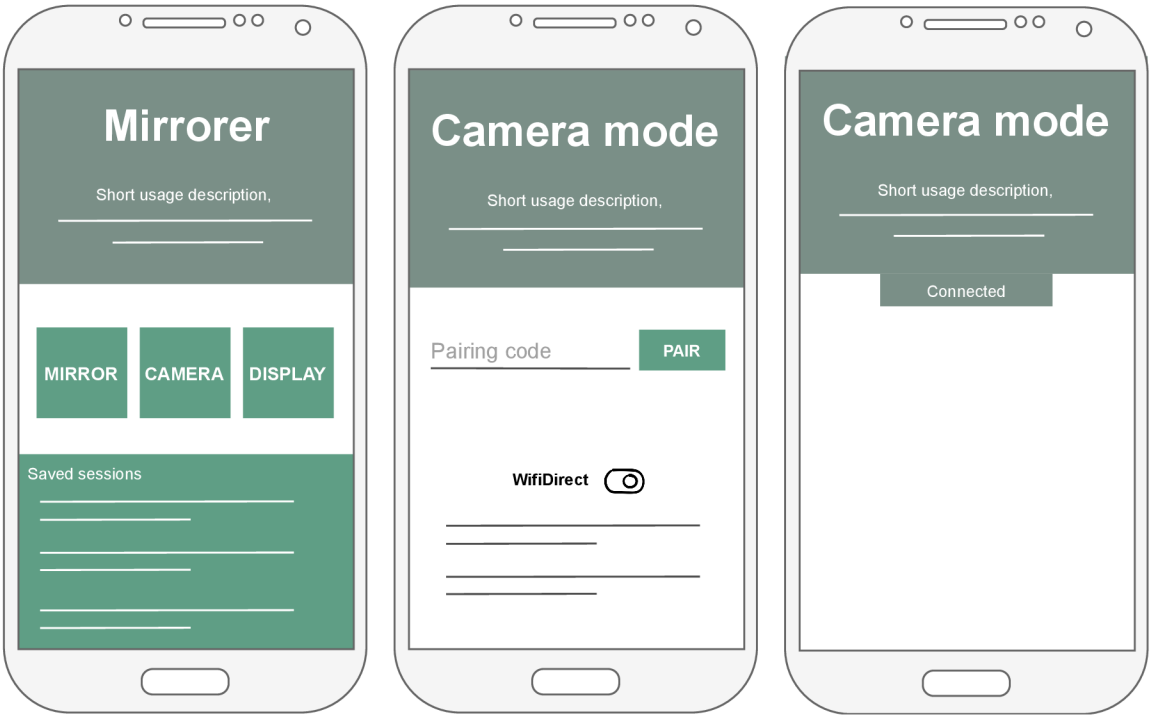
Obrázek 6.2: Vztah režimů zařízení k SDP protokolu

6.2 Návrh uživatelského rozhraní

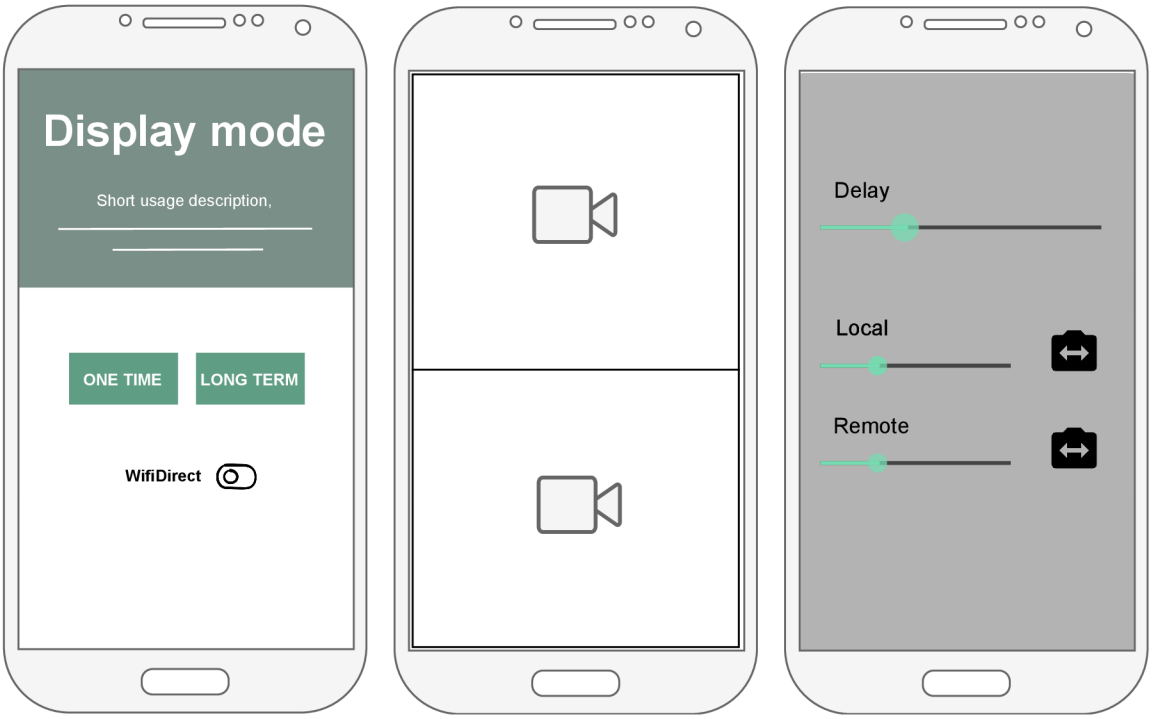
Před samotným začátkem programování je vhodné rozmyslet a navrhnout, jak by mohlo vypadat uživatelské rozhraní výsledné aplikace. Vývojář – v tomto případě spíše designér – by měl mít jakousi základní představu o tom, jakým způsobem bude uživatel výsledný produkt používat. Návrh by neměl být zpracován do přílišných detailů, aby existovala volnost pro připomínky a návrhy uživatelů během uživatelského testování [8].

6.2.1 Wireframes

Má-li designér vytvořit hmatatelný návrh uživatelského rozhraní, může se vydat cestou *wireframes* (česky *drátěné modely*). Wireframes jsou jakýmsi hrubým návrhem uživatelského rozhraní. Neobsahují zdaleka všechny texty, barvy a tlačítka, které bude obsahovat finální produkt. Wireframes slouží jako skvělý komunikační prostředek ve vývojovém týmu. Samotnému designérovi také pomáhají definovat základní prvky uživatelského rozhraní a uvědomit si, jak by celý proces užívání aplikace měl reálně vypadat. Výsledek finálního návrhu aplikace Mirrorer ukazují obrázky 6.3 a 6.4.



Obrázek 6.3: Wireframes – první část. **Vlevo** je úvodní obrazovka aplikace, která obsahuje název aplikace a krátký popis toho, jaké akce na této obrazovce lze provést. Následují volby režimů zrcadlo, kamera a displej, ve spodní části obrazovky je zobrazen seznam uložených dlouhodobých spojení. **Uprostřed** vidíte hlavní obrazovku režimu kamera, opět s krátkým popiskem toho, co se od uživatele očekává. Ve střední části obrazovky je vidět pole pro zadání párovacího kódu. Ve spodní části obrazovky lze nalézt přepínací tlačítko pro aktivaci vyhledávání Wi-Fi Direct peerů. Seznam nalezených peerů následuje níže. **Vpravo** je umístěna obrazovka režimu kamera po úspěšném navázání spojení, která obsahuje indikaci stavu spojení.



Obrázek 6.4: Wireframes – druhá část. **Vlevo** je hlavní obrazovka režimu displej, obsahující tlačítka pro generování párovacího kódu jednorázového nebo dlouhodobého spojení. Po vygenerování bude kód zobrazen. Tato obrazovka obsahuje také přepínací tlačítko pro aktivaci Wi-Fi Direct technologie. **Uprostřed** se nachází obrazovka, která reprezentuje stav aktivního přenosu lokální a vzdálené video stopy po úspěšném navázání spojení. Ve výchozím režimu by neměla obsahovat žádné ovládací prvky, aby mohla videa běžet v režimu *fullscreen*. **Vpravo** je znázorněn režim nastavení video stop, do kterého aplikace přejde po užití vhodného gesta. Mezi funkce obrazovky patří ovládání zpoždění video stop, přepínání kamer a nastavení kvality stop.

6.2.2 Minimum viable product

V rané fázi vývoje vznikl MVP – popsáno v sekci 5.1.2. První použitelná verze obsahovala

- výběr režimu aplikace – kamera či displej,
- jednoduchý párovací mechanismus a
- jednoduché nastavení video výstupu.

Úvodem bylo třeba rozhodnout, zda bude vyvíjena jedna aplikace, obsahující tyto režimy, nebo se budou souběžně vyvíjet dvě různé aplikace pro režim kamera a displej. Po krátké úvaze jsem se vydal cestou jedné aplikace. Hlavními výhodami tohoto přístupu jsou:

1. Lepší uživatelská zkušenost – jedna aplikace obsahuje kompletní funkcionalitu.
2. Kompatibilita WebRTC knihovny je zaručena – oba režimy budou používat vždy stejnou verzi knihovny.
3. Možnost úspory v psaní kódu díky využití sdílené *codebase*[1].

Součástí MVP-verze nebyla pouze samotná aplikace Mirrorer, ale také základní implementace webového signalizačního serveru. Server přijímá požadavky protokolu `HTTP`, konkrétně `GET` a `POST`, kterými jsou zapisovány a získávány informace o peerech ve formátu SDP. Signalizační server byl nezbytnou součástí první verze celého řešení. Bez něj by nebylo možné implementovat párovací mechanismus, založený na přepisu krátkého kódu z jednoho zařízení na druhé. Detailnější popis implementace signalizačního serveru lze nalézt v sekci 7.1.

V poslední řadě byly do uživatelského rozhraní přidány základní ovládací prvky pro nastavení kvality a zpoždění video stop a přepínání kamery. Tyto byly přidány, aby i hrubá verze aplikace uměla něco více než být jakousi *vzdálenou kamerou*.

6.3 Rapid Iterative Testing and Evaluation

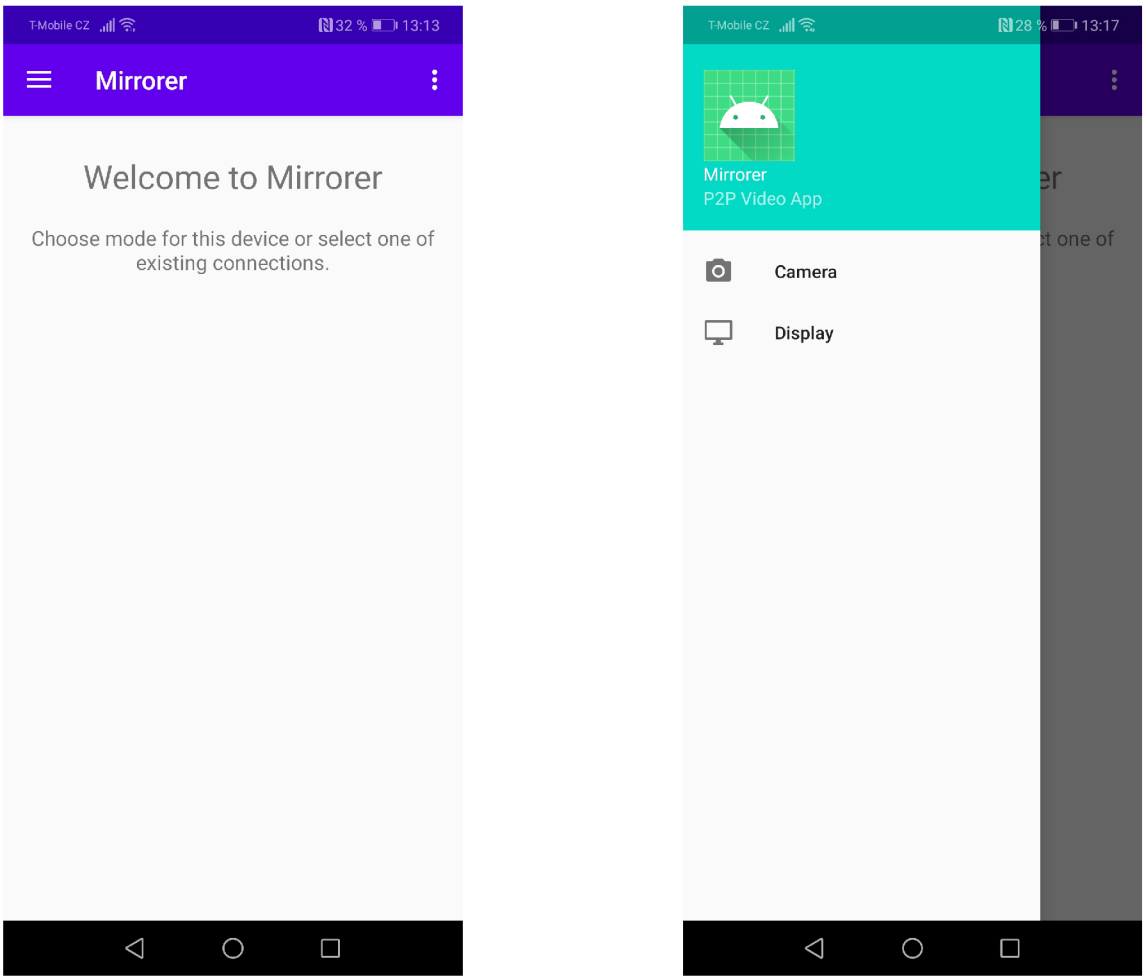
Po vytvoření MVP následovalo iterativní uživatelské testování. Testování probíhalo v přibližně dva týdny dlouhých cyklech, jejichž výstupem byla nová verze aplikace Mirrorer dostupná vybraným uživatelům v Google Play. V následující části se pokusím vybrat největší milníky, které udávaly směr vývoje a vedly k současnému stavu aplikace.

6.3.1 Režim zrcadlo

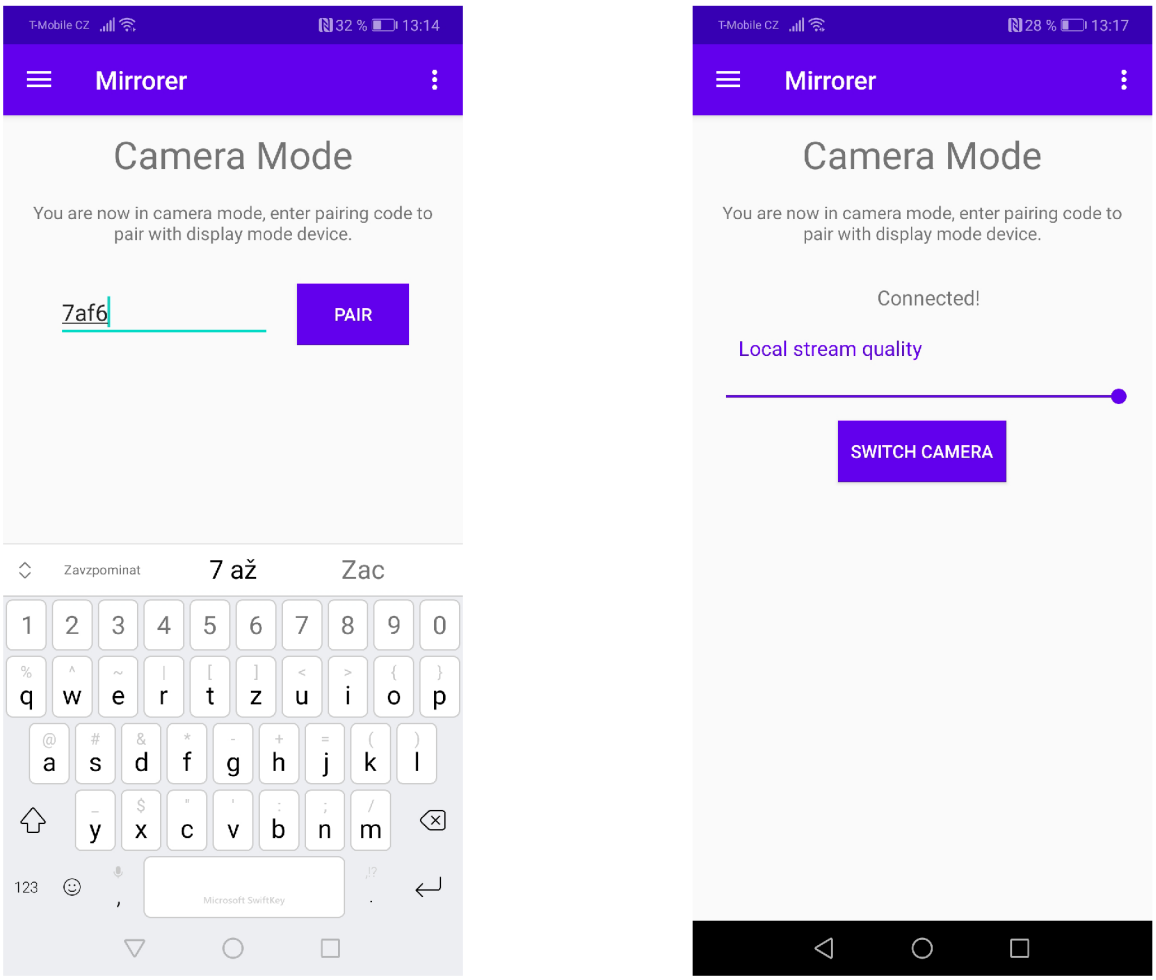
Jedním z prvních nedostatků výsledného řešení byla nutnost disponovat dvěma zařízeními s operačním systémem Android pro možnost vyzkoušení a užívání aplikace. Ukázalo se, že ne vždy bude potřeba použít dvě různá zařízení s nezávislým video výstupem. Některým uživatelům postačí pouze jeden pohled. Pro tyto účely byl implementován třetí režim aplikace – zrcadlo. V tomto režimu nedochází k párování zařízení a přenosu video stopy síťovými prostředky, ale pouze k zrcadlení vlastního výstupu z kamery. Funkce režimu zrcadlo by měly být totožné jako v režimu displej, samozřejmě bez přítomnosti vzdálené video stopy.

Také se ukázalo, že se tento režim skvěle hodí k ladění chyb a testování nových funkcí, které nejsou přímo spjaty s komunikací mezi zařízeními. Rozhodnutí vytvořit režim zrcadlo v rané fázi vývoje považuji za důležitý bod na cestě k současné verzi aplikace.

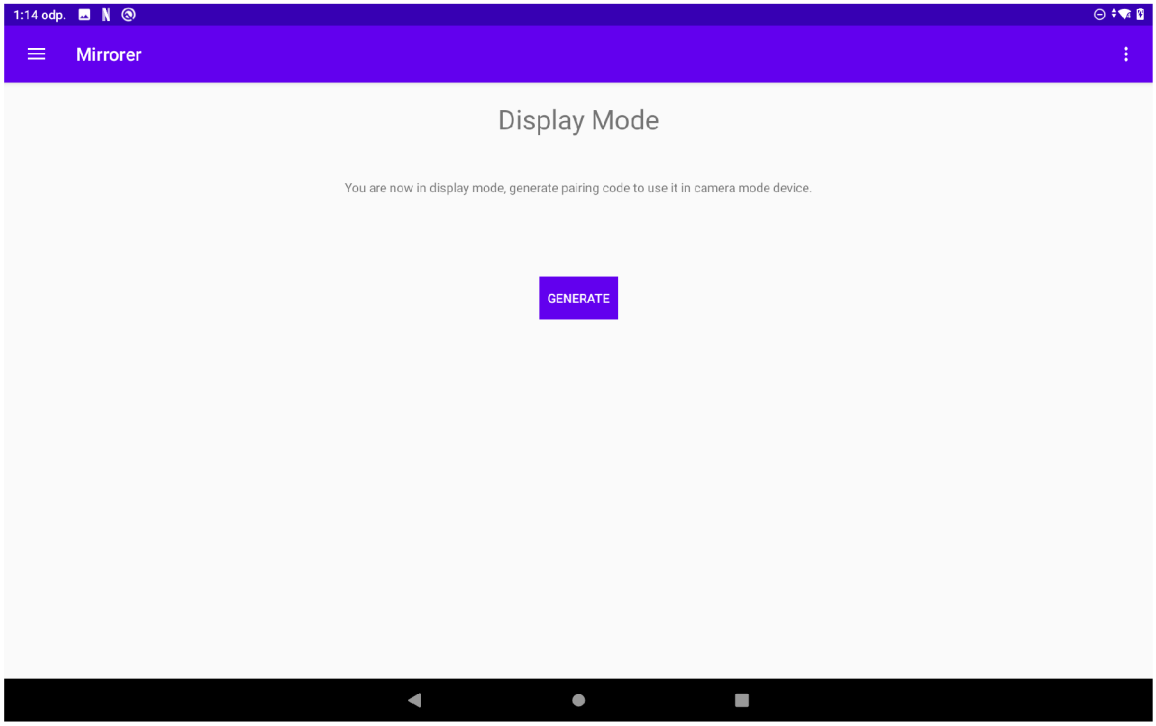
[1] https://en.wikipedia.org/wiki/Codebase

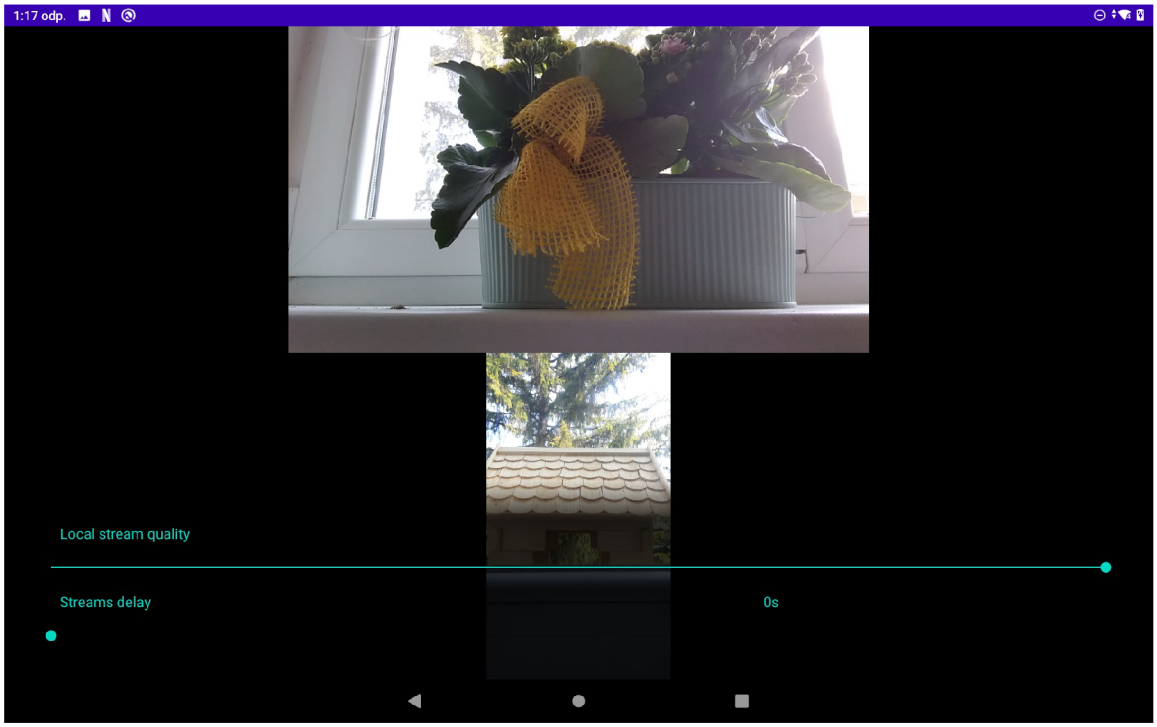


Obrázek 6.5: Snímky obrazovek z první hrubé verze aplikace Mirrorer. Bylo snahou vytvořit minimalistické uživatelské rozhraní, které bude uživatele navádět k výběru režimu. **Vlevo** je vidět úvodní obrazovka aplikace, **vpravo** potom *navigation drawer* s možností výběru režimu.

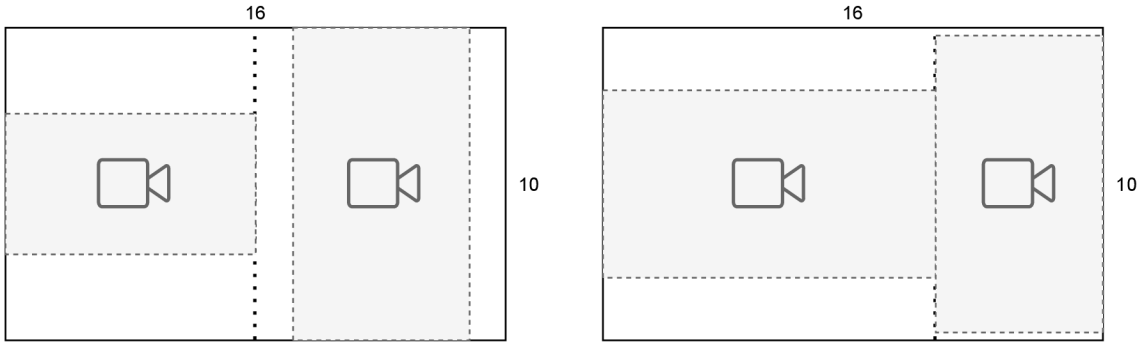


Obrázek 6.6: Snímky obrazovek z první hrubé verze aplikace v režimu kamera. **Vlevo** vidíte proces zadávání párovacího kódu. Ten je nutné nejprve vygenerovat ve vzdáleném zařízení, operujícím v displej režimu. Po zadání a stisknutí tlačítka `PAIR` je zahájena výměna informací mezi kamera a displej peerem. **Vpravo** je zobrazen stav po úspěšném navázání spojení. Obrazovka obsahuje základní ovládací prvky společně s informací o stavu spojení.





Obrázek 6.7: Snímky obrazovek z první hrubé verze aplikace v režimu displej. **Nahoře** je úvodní obrazovka režimu displej, obsahující popis a tlačítko pro generování párovacího kódu. Nelze tedy provést jinou akci než vygenerovat kód pro vzdálené kamera zařízení. Po úspěšném navázání spojení se displej peer přepíná do streamovacího módu, viz obrazovka **dole**. Tento mód vykresluje lokální a vzdálenou video stopu a při poklepání na displej zobrazí dodatečná nastavení.



Obrázek 6.8: Ukázka rozdělení obrazovky na tabletu s poměrem stran displeje 16:10 (nejčastější u Android tabletů) pro video stopy o poměru stran 16:9. Rozdělení displeje v poměru 2:1 (vpravo) je ve vyobrazeném příkladu efektivnější – plocha je využita ze 72 % oproti 58 % v případě rozdělení 1:1 (vlevo).

6.3.2 Efektivnější využití plochy displeje

Dalším podnětem, který vzešel z uživatelského testování bylo zvýšit využití plochy zařízení v režimu displej pro vykreslování videí. První verze aplikace pracovala pouze s video výstupem formátu 16:9 při rozdělení plochy displeje pro lokální a vzdálenou video stopu v poměru 1:1. Lepšího vyplnění dostupné plochy bylo docíleno následujícími funkcemi:

1. Možnost zvětšit jeden z video výstupů, tak že plocha obrazovky je rozdělena v poměru 2:1. Zvětšování výstupu je aktivováno použitím gesta *double-tap* na jedno z videí. Přínos této funkcionality ukazuje obrázek 6.8.

2. Nastavit video výstupu z kamery požadovaný poměr stran. Například při nastavení poměru stran 3:4 u obou video stop bude plocha tabletu o poměru stran 16:10 využita téměř z 94 %. Za účelem zjištění, které poměry stran daná kamera podporuje, byl využit balíček `android.hardware.camera2`, konkrétně třída `CameraCharacteristics`[2], která umí poskytnout právě seznam podporovaných formátů výstupu z kamery. Více o využití podporovaných formátů se dočtete v sekci 7.1.3.

Kombinací těchto dvou přístupů získává uživatel značnou volnost při nastavování zobrazení video stop.

6.3.3 Přiblížení videa

V praxi se ukázalo, že ne vždy mají uživatelé kamerová zařízení dostatečně blízko na to, aby kamera zachytila prováděný prvek dostatečně detailně. Toto bylo vyřešeno implementováním funkce *pinch to zoom* – přiblížení video výstupu obecně známým gestem *pinch*[3], užívaným například také ve většině aplikací sloužících jako fotoaparát nebo kamera. Přiblížení funguje v režimu zrcadlo i v režimu displej, ve kterém lze přibližovat jak lokální, tak vzdálené video. Vedlejším efektem této funkce je také další zefektivnění využití plochy displeje. Přiblížením videa – a tedy jeho zvětšováním v kontextu plochy displeje – dochází ke zmenšování až úplnému zmizení černých pruhů okolo videa.

[2] https://developer.android.com/reference/android/hardware/camera2/CameraCharacteristics

[3] Pomocí pohybu dvou prstů po displeji zvětšujeme (prsty od sebe) či zmenšujeme (prsty k sobě) hodnotu přiblížení.

6.3.4 Všechny ovládací prvky v režimu displej

V prvních verzích aplikace byla možnost nastavovat *lokální* video výstup pomocí ovládacích prvků – přepínání kamery, nastavení kvality videa nebo nastavení poměru stran. Během uživatelského testování se zjistilo, že bude vhodné zvolit jedno ze zařízení za **master**, přičemž toto zařízení bude schopné ovládat kromě svého lokálního video výstupu také video výstup vzdálený. Z povahy komunikace vyplynulo, že master zařízením bude vždy zařízení v režimu displej. Toto je vhodné ze dvou důvodů:

- Umožnit uživateli vidět výsledek ihned po změně nastavení.
- Zařízení v režimu kamera může v budoucnu být potenciálně více – master zařízení by ale mělo být pouze jedno.

Pro implementaci vzdáleného ovládání kamerového zařízení byly využity *datové kanály*. O komunikačním protokolu a využití datových kanálů naleznete více v sekci 7.1.4.

6.3.5 Opětovné přehrání zpožděné stopy

Zpoždění video stopy se ukázalo býti skvělým nástrojem pro zpětnou kontrolu předváděných prvků. Často však během testů docházelo k situacím, kdy chtěl uživatel vidět svůj pokus znova, aby zkontroloval více detailů. Ten však byl již dávno přepsán novými daty. Hledal jsem proto cestu, jak uživatelům umožnit pokus dočasně *nahrát* a zpětně zobrazit. Navíc, jak je vysvětleno v sekci 7.1.5, paměti nebylo nazbyt a nepřicházelo tedy v úvahu ukládat současně výstup z kamery a již přehrané snímky. Nabídlo se poměrně elegantní řešení, který vycházelo z následujících skutečností:

- Uživatel zpravidla nastavuje délku zpoždění na o něco větší dobu, než je trvání prováděného prvku. To proto, aby krátce po odcvičení prvku měl dostupný záznam na displeji.
- Uživatel nepotřebuje vidět záznam toho, jak sleduje cvik na displeji – tyto snímky nemají žádnou hodnotu. Během sledování záznamu tedy nemusíme výstup z kamery uchovávat.

Problém byl tedy vyřešen tak, že po odcvičení prvku lze aktivovat režim *playback loop*, ve kterém je potlačeno ukládání aktuálního výstupu z kamery. V tomto režimu jsou data snímků, které již máme v paměti přehrávána ve smyčce. Jedná se o čerstvě implementovanou funkcionalitu, v budoucnu by mělo být možné tento výsek videa uložit do zařízení, případně sdílet s trenérem a podobně.

Pro účely aktivace režimu *playback loop* bylo vytvořeno jediné tlačítko, které se nachází přímo na obrazovce s video výstupy, nikoli v režimu zobrazení ovládacích prvků. To proto, že přístup k tlačítku musí být co nejjednodušší, aby uživatel neztratil čas vyvoláním obrazovky s ovládacími prvky. Tlačítko je zobrazeno pouze v případě nastavení zpoždění délky alespoň 1 s, v ostatních případech nemá tato funkce příliš velký význam.

6.3.6 Webová varianta displej zařízení

V závěrečné fázi dosavadního vývoje aplikace Mirrorer se ukázalo, že by bylo vhodné se zamyslet nad podporou více platforem. Například ti uživatelé, kteří vlastní pouze mobilní telefon se systémem Android mohou aplikaci užívat pouze v režimu zrcadlo. Video navíc

Funkce	Android displej	Webový displej
Párování – Webový server	Ano	Ano
Párování – Wi-Fi Direct	Ano	Ne
Zpoždění stop	Ano	Ano
Playback loop	Ano	Ne
Max. počet kamera peerů	1	Teoreticky neomezeno
Datové kanály	Ano	Ano
Vzdálené ovládání	Přepínání kamery Kvalita videa Poměr stran	Přepínání kamery
Dlouhodobá spojení	Ano	Ne
Otáčení videa	Automatické	Ruční

Tabulka 6.1: Tato tabulka srovnává funkce režimu displej mezi Android a webovou variantou řešení. Párování prostřednictvím Wi-Fi Direct nebylo implementováno, jelikož přístup z webového prohlížeče k Wi-Fi rozhraní daného zařízení dle mých znalostí v současné době není možný. Naopak datové kanály včetně demonstrativní funkce přepínání kamery byly úspěšně implementovány. Implementace zpoždění a nutnost ručního otáčení videa budou rozebrány v další části textu. O implementaci funkcí na platformě Android se dočtete v kapitole 7.

vidí na poměrně malém displeji[4]. Proto jsem se rozhodl vytvořit webovou variantu aplikace, která by fungovala v režimu displej. Výhodou bylo, že v rané fázi vývoje jsem experimentoval právě s webovou variantou WebRTC, a mohl jsem tedy na tyto experimenty navázat. Pro samotnou implementaci řešení jsem použil framework Angular[5] – to mi umožnilo aplikaci lépe modularizovat, a také využít grafické komponenty knihovny Angular Material[6].

Webová varianta displeje demonstruje, že WebRTC standard lze užít pro komunikaci mezi různými platformami. Dovolím si poznamenat, že se v nynějším stavu nejedná o plnohodnotnou náhradu displejového režimu aplikace Mirrorer pro Android, zejména proto, že webová varianta začala vznikat poměrně krátkou dobu před odevzdáním této práce. Tabulka 6.1 srovnává implementované funkce displej režimů mezi Android a webovou aplikací.

Největší výzvou při implementaci webové varianty displeje bylo najít nový přístup k vyřešení zpožděného přehrávání video stop. V prvních fázích bylo experimentováno s vlastností `playoutDelayHint` objektu `RTCRtpReceiver`, která dokáže nastavit zpoždění daného přijímače v sekundách. Jedná se ovšem o experimentální vlastnost[7], její podpora v prohlížečích je velmi omezená. Pokud už je vlastnost podporována, záleží na konkrétním prohlížeči, jaké maximální zpoždění umožní. Nebylo mým cílem nechat uživatele napospas jejich prohlížeči, a tak jsem zvolil jinou cestu. Ta spočívá v použití rozhraní `MediaRecorder`[8], které umožňuje nahrávat a zpětně přehrát tok audiovizuálních dat – v mém případě stačilo předat nahrávači

[4] S výjimkou uživatelů, jejichž zařízení podporuje sdílení obrazovky, např. prostřednictvím USB-C rozhraní.

[5] https://angular.io/

[6] https://material.angular.io/

[7] https://groups.google.com/g/discuss-webrtc/c/wtuhQu6c1KY

[8] https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder

objekt třídy `MediaStream` obsahující tok dat dané video stopy. Při spouštění nahrávání lze specifikovat, v jakých časových intervalech se má generovat událost `dataavailable`, která s sebou nese nahraná data z posledního časového úseku. Naneštěstí MediaRecorder API negarantuje, že každý blok dat obdržený z této události lze přehrát sám o sobě[9]. Video stopu, kterou lze přehrát jako celek získáme teprve spojením všech dílčích bloků dat.

Výsledkem dalších experimentů bylo použití **dvou** různých `<video>` elementů, které budou střídavě přehrávat právě nahranou stopu. Problémem je, že ve chvíli kdy zastavuji nahrávání a spouštím nové, přicházím o několik snímků videa, protože restart nahrávání nepatří k triviálním operacím. Toto způsobovalo jemné zaseknutí těsně před záměnou `<video>` elementů. Abych docílil co nejhladšího přechodu, pokusil jsem se použít dvě instance třídy `MediaRecorder` a načasovat souběžné nahrávání stopy tak, abych nepřišel o žádné snímky videa. Úspěšný jsem však byl pouze v případě **lokální** video stopy. Zpětně přehrávaná data získaná tímto způsobem ze **vzdálené** video stopy byla různými způsoby poškozena. Toto vedlo k dalším experimentům – nyní již pouze s jednou instancí `MediaRecorder`.

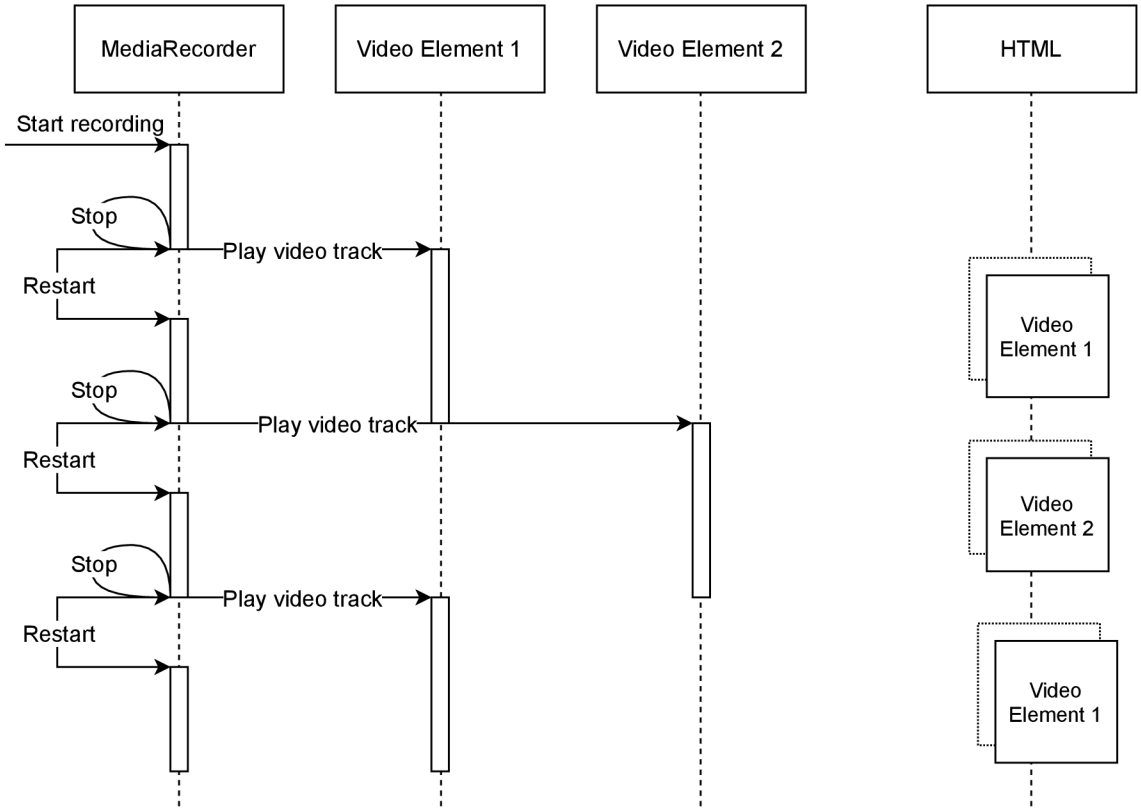
Zjistil jsem, že zpomalit přehrávanou video stopu ve webovém prostředí je dnes velmi jednoduché – slouží k tomu vlastnost `playbackRate` objektu `HTMLMediaElement`, hodnota vyjadřuje relativní změnu rychlosti přehrávání vůči původní (např. hodnota 0,5 přehrává stopu poloviční rychlostí). Díky zpomalení přehrávání získávám více času pro `MediaRecorder` a přechod při záměně videí je téměř nepozorovatelný. Tímto způsobem řešení sice stále přicházím o snímky videa, výhodou ale je, že nedochází k zasekávání na posledním snímku každé dílčí video stopy. Pro lepší představu o této funkcionalitě jsem připravil sekvenční diagram na obrázku 6.9, rovnice (6.1) pak ukazuje výpočet hodnoty `playbackRate` nutné k hladkému přechodu mezi video stopami.

$$\mathcal{R} = 1 - \frac{\Delta}{\mathcal{D}} \tag{6.1}$$

Rovnice (6.1) ukazuje, že hodnota $\mathcal{R}$, tedy relativní rychlost přehrávané stopy závisí na aktuálním zpoždění $\mathcal{D}$, které je zároveň délkou nahrávání dílčích stop. Dále ve výpočtu figuruje parametr Δ, který nastavujeme na očekávanou dobu trvání restartu nahrávání video stopy. Čím je parametr nižší, tím bude změna rychlosti videa menší, zároveň ale hrozí větší riziko zaseknutí na posledním snímku dílčí stopy. V praxi byl parametr nastaven na hodnotu 30 ms. Výpočet dále ukazuje jemně paradoxní situaci, kdy při nastavení **vyššího zpoždění** dojde k **menšímu zpomalení** videa, což je optimálnější z hlediska uživatelské zkušenosti. Ovšem i při nastavení zpoždění na pouhé $2\,s$ dojde ke zpomalení přehrávání videa na 98,5 % původní rychlosti, což je velmi obtížně pozorovatelný rozdíl.

Při nahrávání stopy videa dojde bohužel ke ztrátě informace o jeho orientaci, proto je uživateli ve webové variantě displeje umožněna manuální změna orientace otáčením videa v 90° intervalech.

[9] Toto bylo ověřeno v nejnovějších verzích běžně užívaných prohlížečů.



Obrázek 6.9: Sekvenční diagram znázorňující implementaci zpoždění jedné video stopy ve webovém řešení displejového zařízení. Objekt `MediaRecorder` opakovaně spouští nahrávání stopy. S dokončením nahrávání předává data jednomu z `<video>` elementů a spouští další nahrávání. Prodleva, znázorněná akcí *Restart*, je záměrně zveličena. V praxi se jedná o časový úsek v řádu desítek milisekund. Mezi elementy `<video>` je přepínáno nastavováním CSS vlastnosti `z-index`.

Kapitola 7

Výsledný produkt

V této kapitole popisuji výsledek mé práce. Než přistoupíme k detailům implementace, shrnu základní body celého řešení, které byly implementovány.

1. Webová implementace signalizačního serveru na bázi protokolu HTTP.
2. Mobilní aplikace **Mirrorer** s režimy zrcadlo, kamera a displej.
3. Párování mobilních zařízení prostřednictvím webového signalizačního serveru.
4. Párování mobilních zařízení prostřednictvím přímé signalizace prostřednictvím Wi-Fi Direct.
5. Ukládání dlouhodobých spojení pro možnost automatického navázání spojení.
6. Nastavení zpoždění video stop v režimech zrcadlo a displej.
7. Možnost aktivování *playback loop* v režimech zrcadlo a displej.
8. Ovládání kamerového výstupu ve všech režimech – přepínání kamery, nastavení kvality a poměru stran.
9. Vzdálená komunikace a ovládání prostřednictvím *datových kanálů*.
10. Webová aplikace pro režim displej.

7.1 Implementační detaily

V této části budou popsány některé netriviální implementační detaily mého řešení.

7.1.1 Signalizace – LAN

Signalizací v režimu LAN je myšlena situace, kdy jsou obě zařízení připojena ke stejné lokální síti – zpravidla prostřednictvím Wi-Fi a mají také přístup ke globální internetové síti kvůli komunikaci s webovým serverem. Kromě této varianty byla implementována také alternativní signalizace pro variantu *Wi-Fi Direct*, která bude popsána v další části. Hlavním smyslem signalizace je předat si vzájemně informace o okolní síti a zvolit vhodná síťová rozhraní k přenosu dat v reálném čase. Jak již bylo zmíněno, SDP zprávu typu offer vytváří

vždy zařízení v režimu displej. Displej zařízení zasílá tuto zprávu ve formátu JSON prostřednictvím REST API na webový server. Obsah zprávy je na serveru uložen do souboru a server generuje odpověď s náhodnou sekvencí hexadecimálních znaků – *hash*. Hash slouží:

- Jako dočasný identifikátor P2P komunikace.
- Jako klíč k přístupu k SDP souboru pro zařízení v kamerovém režimu.

Displej zařízení následně zahájí iterativní dotazování na přítomnost odpovědi na serveru. Po úspěšném přečtení SDP offer a zaslání SDP answer kamerovým zařízením je tato zpráva vrácena v odpovědi displej zařízení a na základě informací vyplývajících z SDP protokolu je sestavena komunikace. Výše uvedené je podrobněji znázorněno v diagramu na obrázku 7.1.

Po získání souboru SDP answer displej zařízením jsou soubory ze serveru odstraněny, aby mohl být hash v budoucnu opět využit pro signalizaci. V případě rozšíření aplikace, a tedy vyšší frekvenci signalizace by bylo třeba implementovat mechanismus na mazání SDP souborů, které nebyly zařízeními získány, a tedy nebyly automaticky smazány. Výše uvedený mechanismus popisuje signalizaci ve variantě *One Time*. Jedná se tedy o jednorázové spojení dvou zařízení. V této variantě je hash 4-místný, což v ideálním případě umožňuje až 65536 právě navazujících spojení. Kromě samotných dat SDP protokolu je v souboru také časové razítko, aby bylo možné určit, zda jsou data v souboru aktuální, a tedy platná pro právě navazující spojení. Tento prvek implementace je jakousi záchranou brzdou v případě, že dojde ke kolizi. Časové razítko v souboru má platnost 10 minut – staré nesmazané soubory s SDP daty tedy nemohou být použity, dokud nejsou aktualizovány. Příklady souborů ukazují výpisy 7.1 a 7.2.

V rámci zlepšení uživatelské zkušenosti je však implementována také varianta dlouhodobých sezení – *Long Term*, jejichž hash je po navázání spojení uložen v paměti zúčastněných zařízení pro příští využití k signalizaci. Oproti jednorázové variantě je v tomto případě hash délky 6 znaků – až 17 milionů uložených dlouhodobých spojení.

```
{
    "type": "offer",
    "sdp": "v=0\r\no=- 1019545947847706806 2 IN IP4 127.0.0.1\r\ns=-\r\n ...",
    "timestamp": "2021-05-01 14:38:55"
}
```

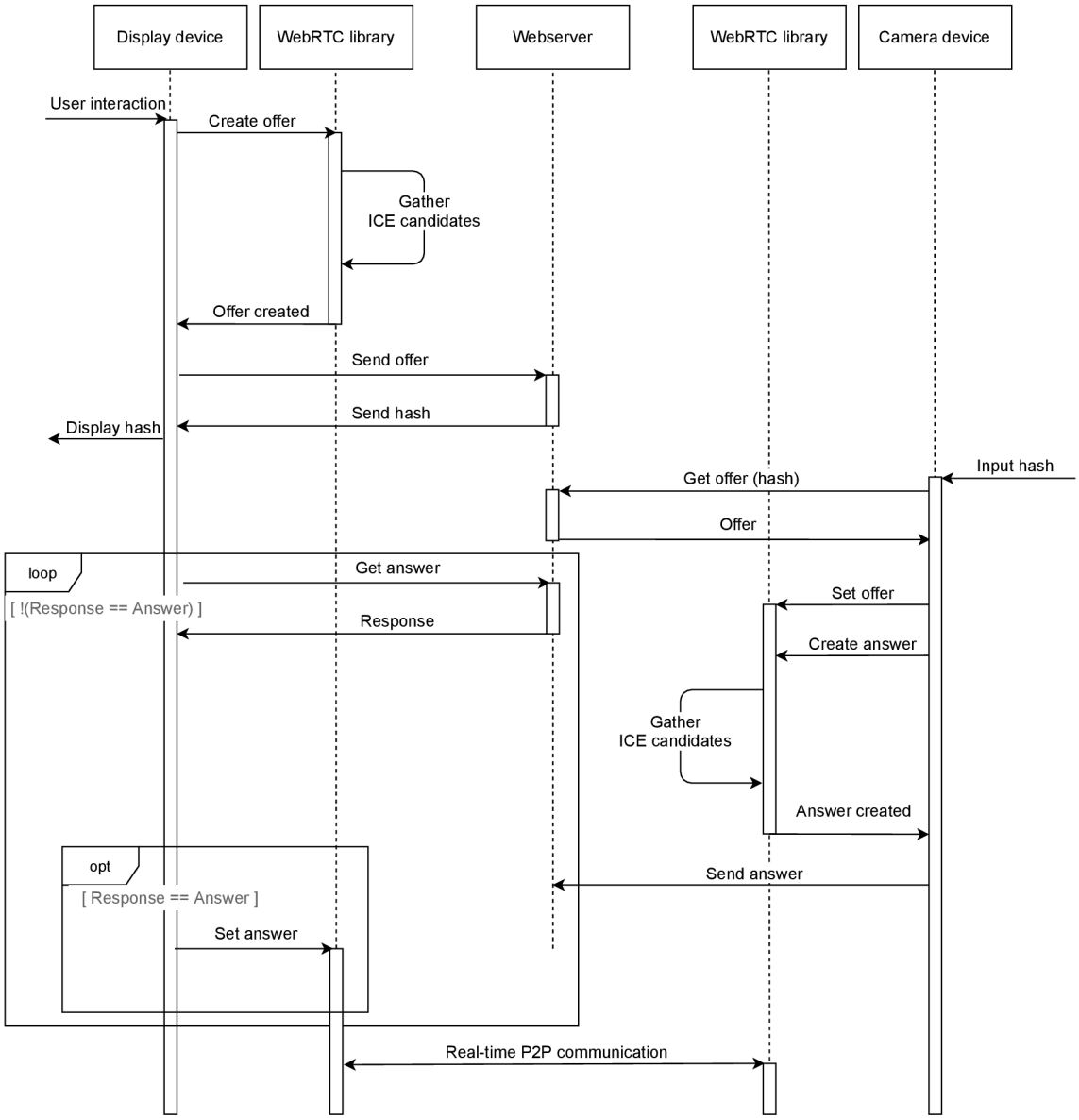
Výpis 7.1: Soubor s SDP zprávou typu offer.

```
{
    "type": "answer",
    "sdp": "v=0\r\no=- 1871110692182772699 2 IN IP4 127.0.0.1\r\ns=-\r\n ...",
    "timestamp": "2021-05-01 14:39:26"
}
```

Výpis 7.2: Soubor s SDP zprávou typu answer.

Někteří čtenáři by si mohli položit otázku, jak je zajištěna bezpečnost tohoto řešení. Může během procesu signalizace dojít k záměně souborů, které by způsobilo spárování jiných zařízení, než bylo zamýšleno? Pojďme si shrnout, jaké jsou předpoklady úspěšného navázání spojení:

1. Zařízení, která chceme spárovat se musí nacházet na stejné lokální síti. Toto je dáno vynecháním STUN serverů z implementace signalizace, podrobnosti lze nalézt v sekci 3.3.2.



Obrázek 7.1: Sekvenční diagram signalizace aplikace Mirrorer prostřednictvím webového serveru.

2. Od zapsání dat na server do jejich načtení nesmí uběhnout více než 10 minut, po této době se data zneplatňují.

Výše uvedené podmínky samy o sobě téměř vylučují možnost záměny souborů se signalizačními daty. I pokud by k záměně došlo, budou spojena zařízení ze stejné lokální sítě, což považuji za méně závažnou variantu.

7.1.2 Signalizace – Wi-Fi Direct

Předpokladem úspěšné signalizace prostřednictvím Wi-Fi Direct kanálu je navázání tohoto typu spojení. Zde jsem se potýkal s nedokonalostí Wi-Fi Direct balíčku dostupném na platformě Android. Přehledná a relativně stručná dokumentace na vývojářském portálu[1] rozhodně neodpovídá skutečnosti a osobně zde vidím velký potenciál ke zlepšení této platformy. Pokud bych návod použil tak jak je podán, pravděpodobnost úspěšného navázání spojení by se blížila nule. Při implementaci jsem řešil zejména 3 problémy:

1. Volání metod z balíčku `android.net.wifi.p2p` často neuspěje a vrací chybový stav `BUSY`.

2. Bylo velmi nejisté, zda se zařízení vzájemně vůbec naleznou – tedy zda zařízením dorazí zpráva `WIFI_P2P_PEERS_CHANGED_ACTION` a bude zjištěna přítomnost daného peera.

3. Po úspěšném navázání spojení bylo *group owner* zařízení vybráno spíše náhodně, nastavení vlastnosti `groupOwnerIntent` konfiguračního objektu – popsáno v závěru sekce 3.4 – na *maximální* či *minimální* hodnotu nemělo v praxi příliš význam.

Během vývoje se ukázalo, že o něco vhodnější bude použít specifičtější variantu navázání Wi-Fi Direct spojení, a to za použití *registrovaných služeb*. Lze vytvořit vlastní instanci služby obsahující základní informace o službě (název, protokol a další). Jedním z typů služeb na platformě Android je tzv. Bonjour služba[2] – implementovaná třídou `WifiP2pDnsSdServiceInfo`. Výhodou použití služby byla možnost *filtrovat* nalezená zařízení a ponechat pouze ta, která nabízejí konkrétní typ služby. Služba je registrována v displej zařízení metodou `addLocalService()` objektu typu `WifiP2pManager`. Pro objevení služby bylo třeba v kamerovém zařízení implementovat třídu `DnsSdServiceResponseListener` – ta obdrží informace o nalezené službě včetně **MAC adresy zařízení**. Následuje postupné volání metod `addServiceRequest()` a `discoverServices()`, které spustí mechanismus objevování okolních Wi-Fi P2P služeb.

Mechanismus nabízení a objevování služeb bohužel zdědil výše uvedené problémy od prostého Wi-Fi Direct spojení. Abych zvýšil pravděpodobnost objevení služby kamerovým zařízením, upravil jsem algoritmy dle doporučení jiných vývojářů[3] následujícím způsobem:

- V zařízení v roli **displej** je před voláním metody `addLocalService()` přidáno volání `clearLocalServices()`. Navíc je po přidání služby periodicky v intervalu několika sekund volána funkce `discoverPeers()`, která zřejmě iniciuje vysílání informací o službě.

[1] https://developer.android.com/guide/topics/connectivity/wifip2p
[2] https://www.andriydruk.com/post/mdnsresponder/
[3] https://stackoverflow.com/questions/26300889/wifi-p2p-service-discovery-works-intermittently/31641302

- V **kamerovém** zařízení se ukázalo býti užitečným volat periodicky sekvenci funkcí

  1. `WifiP2pManager.removeServiceRequest()`,
  2. `WifiP2pManager.addServiceRequest()`,
  3. `WifiP2pManager.discoverServices()`.

  Tento přístup znamenal zvýšení pravděpodobnosti objevení služby.

Smyslem objevení služby je víceméně získání **MAC adresy** vzdáleného zařízení, kterou je nutné znát pro další fázi protokolu. Po úspěšném objevení služby kamerovým zařízením je vzdálené displejové zařízení přidáno do seznamu peerů. Uživatel z tohoto seznamu vybírá dle názvu zařízení. V reakci na volbu uživatele je volána metoda `connect()` s konfiguračním parametrem typu `WifiP2pConfig`, který mimo jiné obsahuje výše zmíněnou vlastnost `groupOwnerIntent`. V tomto případě byl `groupOwnerIntent` nastaven na hodnotu 0, čímž má být indikováno, že se kamerové zařízení nechce stát *group owner* zařízením. Často bohužel nastával pravý opak. Naštěstí objekt `WifiP2pManager` poskytuje také metodu `createGroup()`, která vytvoří Wi-Fi Direct skupinu s daným zařízením jako *group owner*. Tato cesta vypadala nadějně, bohužel jsem ale došel ke zjištění, že pokud displej zařízení takto skupinu vytvoří, klesne pravděpodobnost úspěšného objevení služby kamerovým zařízením opět k nule. Řešením by bylo nechat zodpovědnost na uživateli, aby skupinu vytvořil až po objevení displejového zařízení. Vzhledem k tomu, že jedním z cílů aplikace byla vysoká úroveň uživatelské zkušenosti, rozhodl jsem se toto vyřešit na pozadí algoritmem zaznamenaným na obrázku 7.2.

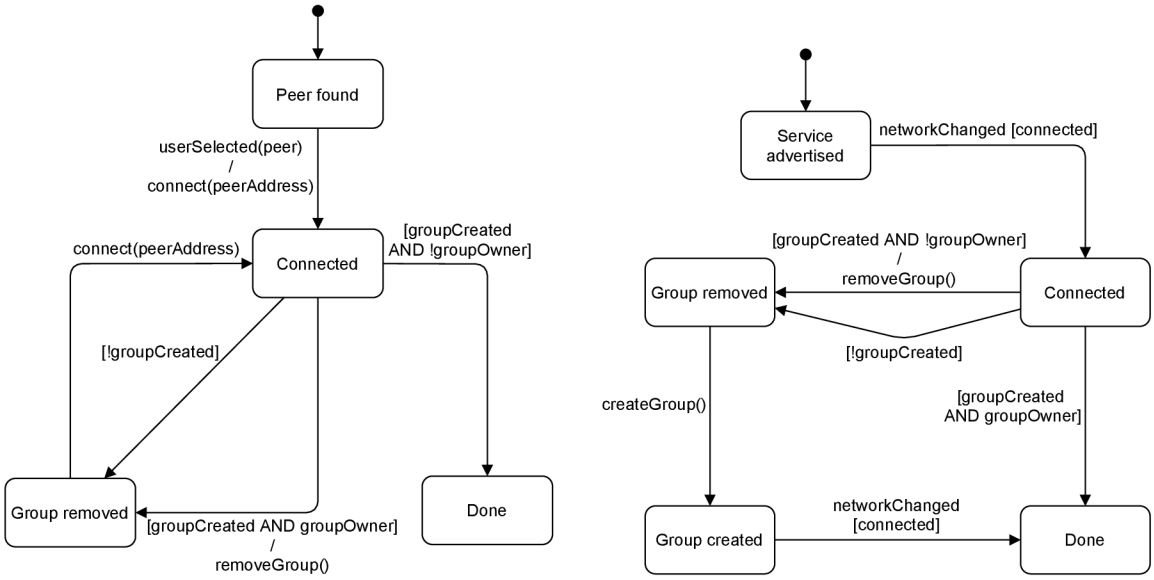
V případě, že se uživatel rozhodne využít k přenosu dat technologii Wi-Fi Direct, je potřeba k signalizaci přistoupit odlišným způsobem. Pokud uživatel aktivuje režim Wi-Fi Direct, je třeba počítat s tím, že nemusí mít přístup k lokální síti a globální internetové síti – například pokud cvičí ve venkovním prostředí. Využití webového signalizačního serveru zde tedy nepřipadá v úvahu. Navíc i pokud by se signalizační data podařilo předat (například prostřednictvím mobilních dat), spojení by přesto nebylo sestaveno, protože zařízení nesdílí stejnou lokální síť. Signalizační data jsou mezi peery předána *napřímo* komunikačním kanálem, který umožnilo vytvořit Wi-Fi Direct spojení.

Signalizační protokol pro Wi-Fi Direct variantu vychází z předpokladu, že displej zařízení je vždy zároveň *group-owner* zařízením. V praxi to znamená, že obě strany jsou bezprostředně pro sestavení spojení schopny zjistit pouze IP adresu displej zařízení. Z tohoto důvodu figuruje displej zařízení v komunikaci jako **server**, který poslouchá na interně daném portu, dokud se nepřipojí **klient** – kamerové zařízení. Displej zařízení je v tomto případě také samo o sobě *signalizačním serverem*. Vzhledem k tomuto faktu se během komunikace ušetří 2 kroky – zápis zprávy typu offer na server a čtení zprávy typu answer ze serveru. Celá signalizace je popsána sekvenčním diagramem na obrázku 7.3.

Po výměně signalizačních dat je sestaveno P2P spojení v reálném čase. Jsou-li zařízení zároveň připojena ke stejné lokální síti, může následně přenos video stopy probíhat prostřednictvím této sítě. Pravděpodobnější ale je, že budou data zasílána v rámci Wi-Fi Direct spojení.

7.1.3 Formát výstupu z kamery

Vzhledem k tomu, že maximální délka zpoždění video stop závisí na velikosti rozlišení snímků videa, bylo nezbytné nabídnout uživatelům možnost toto rozlišení nastavovat. Možná



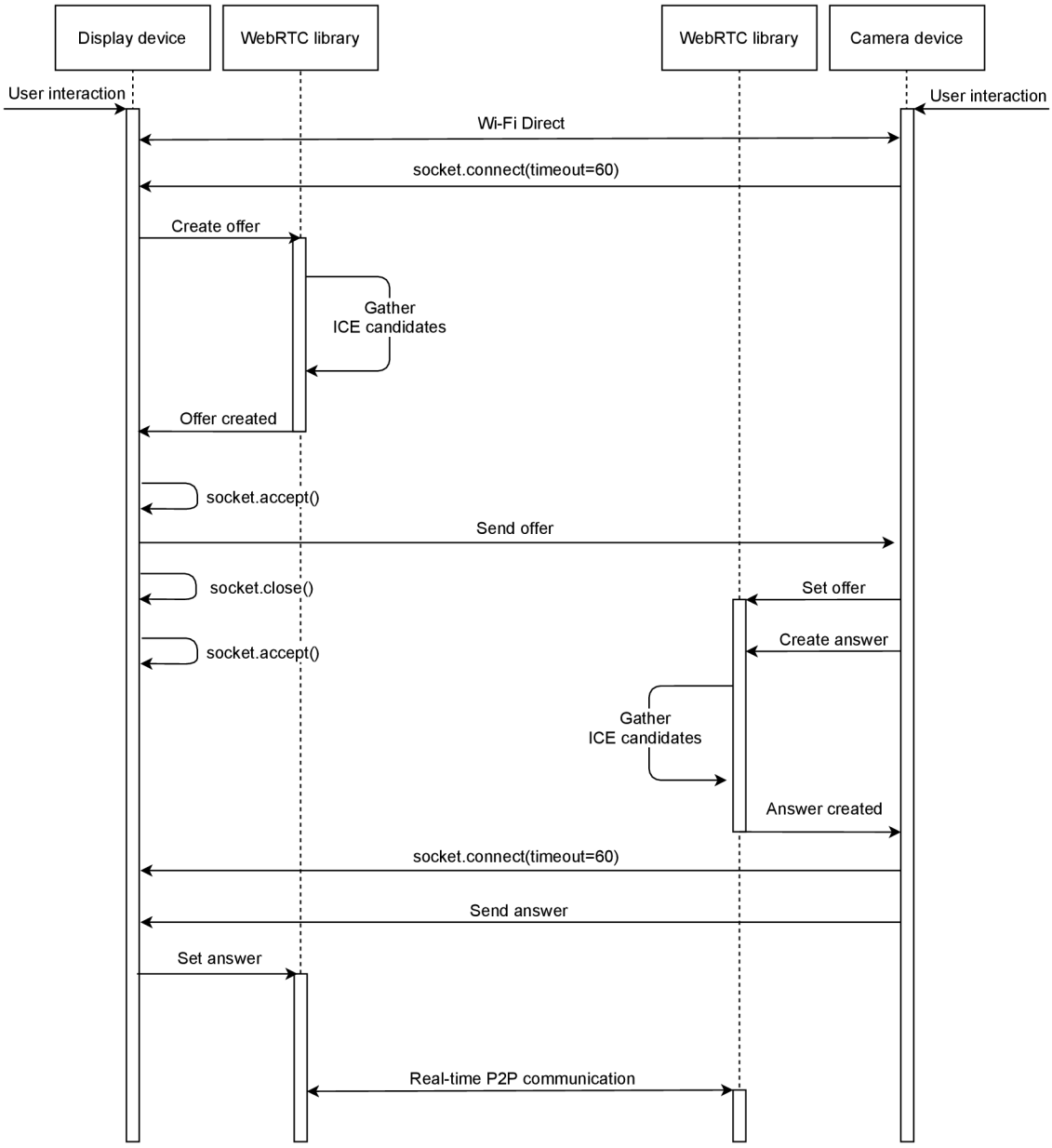
Obrázek 7.2: Stavové diagramy, které ukazují implementaci algoritmu vytvoření Wi-Fi Direct skupiny s displej zařízením v roli *group owner*. Diagram **vlevo** ukazuje události a přidružené akce v kamerovém zařízení, předpokladem je, že služba vzdáleného zařízení již byla objevena. Po zvolení peera uživatelem je zahájen pokus o připojení. Je-li po navázání spojení toto zařízení *group owner*, případně není-li skupina vůbec vytvořena, pokus o připojení je opakován. **Vpravo** vidíte popis chování displejového zařízení, které po navázání spojení rovněž kontroluje, zda je *group owner* zařízením. Pokud ano, je možné pokračovat v signalizaci. V opačném případě je vytvořena skupina – displejové zařízení se automaticky stává *group owner* zařízením a očekává opětovné připojení kamerového zařízení.

rozlišení svázaná s konkrétním kamerovým zařízením jsou získávána prostřednictvím balíčku `android.hardware.camera2`. Konkrétně pro každé kamerové zařízení – přední i zadní kameru – získám objekt typu `CameraCharacteristics`, který, mimo jiné, obsahuje výčet všech možných rozlišení kamery, se kterými lze operovat. Tato rozlišení jsou dále seřazena vzestupně dle počtu pixelů na snímek a uložena v paměti kamerového zařízení.

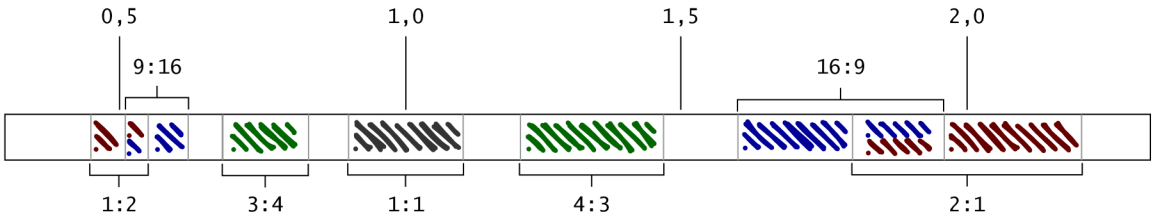
Během uživatelského testování došlo k rozšíření funkcionality o volbu *poměru stran* video stopy – viz sekce 6.3.2. Tento fakt znamenal nutnost všechna možná rozlišení kamery filtrovat dle aktuálního poměru stran. Objekt typu `MyCameraManager`, určený pro přístup ke kamerovému rozhraní zařízení, tedy obsahuje následující struktury:

- Seznam `Supported[Front|Back]CameraFormats` – ten obsahuje všechny dostupná rozlišení získána prostřednictvím objektu `CameraCharacteristics` při inicializaci kamerového rozhraní.
- Seznam `_currentPossibleFormats` – dynamický seznam, který obsahuje pouze ta rozlišení, která odpovídají aktuálnímu poměru stran.

Jako výchozí byl zvolen poměr stran 16:9, jelikož bývá nejčastěji výchozím poměrem stran u mobilních aplikací na bázi kamery či fotoaparátu. Protože kamerová zařízení nabízejí také spoustu méně typických poměrů stran video výstupu, rozhodl jsem se toto uživatelům abstrahovat a nabízet pouze několik základních – řekněme populárních – poměrů,



Obrázek 7.3: Sekvenční diagram signalizace aplikace Mirrorer v režimu Wi-Fi Direct. Ustavení samotné Wi-Fi Direct komunikace je záměrně abstrahováno.



Obrázek 7.4: Ukázka pokrytí možných rozlišení. K danému poměru stran řadíme všechna rozlišení, která jsou v 10% toleranci. Barvy jsou použité pouze pro odlišení jednotlivých poměrů. Pokud rozlišení nespadá do žádné předvolené kategorie, je přiřazeno do speciální kategorie `Other`.

a to 16:9, 9:16, 4:3, 3:4, 2:1, 1:2 a 1:1. Abych zajistil co největší variabilitu při nastavování kvality (rozlišení) pro daný poměr stran, rozhodl jsem se, že k danému poměru stran přiřadím všechna rozlišení, jejichž poměr se odlišuje o méně než 10 %. Je tedy možné volit z poměrně rozsáhlé škály rozlišení video stopy, a zároveň výstup s drobnou či žádnou odchylkou odpovídá zvolenému poměru stran. Pokrytí všemožných rozlišení tímto řešením ukazuje obrázek 7.4.

7.1.4 Protokol datových kanálů

V průběhu vývoje aplikace se ukázalo, že bude vhodné zajistit mechanismus, kterým by si zařízení mohla zasílat informace o svém aktuálním stavu, případně o událostech, které mohou v průběhu přenosu nastat. Aktuální verze aplikace Mirrorer využívá datové kanály trojím způsobem:

1. Pro zaslání názvu nového dlouhodobého spojení do kamerového zařízení.
2. Pro zaslání událostí z displej zařízení do kamerového zařízení, kterými uživatel vzdáleně nastavuje kamerové zařízení.
3. Pro zaslání aktuálního stavu kamerového zařízení do displej zařízení, například počet nastavitelných profilů kamery.

Právě k tomuto účelu slouží *datové kanály*. Velkou výhodou datových kanálů v konceptu WebRTC je velká míra abstrakce – pracujeme pouze s objekty `DataChannel`, které obsahují přímočaré rozhraní pro zasílání a příjem dat. Data jsou zasílána v binární formě. Pro zvýšení přehlednosti komunikace byl vytvořen textový protokol na bázi formátu JSON. Každá zpráva je sama o sobě JSON objektem, který obsahuje jednu nebo více dvojic *klíč-hodnota*. Každá z těchto dvojic reprezentuje jednu zprávu, klíčem je řetězec, který udává typ zprávy. V následující části textu vysvětlím jednotlivé typy zpráv, k čemu slouží a jak jsou zasazeny do kontextu užívání aplikace.

Nové dlouhodobé spojení

Jak již bylo řečeno, aplikace umožňuje vytvoření tzv. *Long Term* spojení mezi kamerovým a displej zařízením, což v praxi znamená, že po prvním navázání tohoto spojení je v obou zúčastněných zařízeních uložen *párovací kód*, *režim zařízení* a *název* dlouhodobého spojení. Párovací kód a režim je oběma zařízením znám již před samotným začátkem přenosu videa.

Název spojení je zadáván v displej zařízení po úspěšném navázání spojení. Z důvodu zlepšení uživatelské zkušenosti bylo vyžadováno, aby bylo spojení uloženo také v kamerovém zařízení, pokud možno zcela transparentně. Toho bylo docíleno zasláním názvu připojení do kamerového zařízení bezprostředně po jeho zadání. Takto mohou obě zařízení sdílet stejný název dlouhodobého spojení, což je z pohledu případů užití žádoucí. Ukázku zprávy `M_NEW_SESSION` můžete vidět ve výpisu 7.3.

```
{
    "M_NEW_SESSION": {
        "name": "LENOVO + HUAWEI",
        "hash": "dc0827"
    }
}
```

Výpis 7.3: Ukázka zprávy typu nové dlouhodobé spojení. Zpráva je zasílána po vytvoření názvu spojení z displej zařízení do kamerového zařízení. Po přijetí zprávy uloží kamerové zařízení toto dlouhodobé spojení do vlastní paměti, čímž je v budoucnu umožněno snadné navázání tohoto spojení.

Přepnutí kamery

Vzdálené přepínání přední a zadní kamery prostřednictvím displejového zařízení bylo jednou z prvních implementovaných zpráv. Zpráva s klíčem `M_SWITCH_CAMERA` má prázdné tělo – jedná se pouze o impuls k přepnutí kamery na opačnou. Samotná implementace přepínání kamery spočívá v zavolání funkce `switchCamera()` objektu třídy `VideoCapturer` z WebRTC knihovny. Toto se ukázalo jako výrazné zjednodušení a úspora v psaní kódu. Pokud by knihovna neobsahovala podporu přepínání kamery, bylo by nutné odebrat původní video stopu, vytvořit novou, přidat ji k objektu `PeerConnection` a v displejovém zařízení na přidání stopy adekvátně zareagovat.

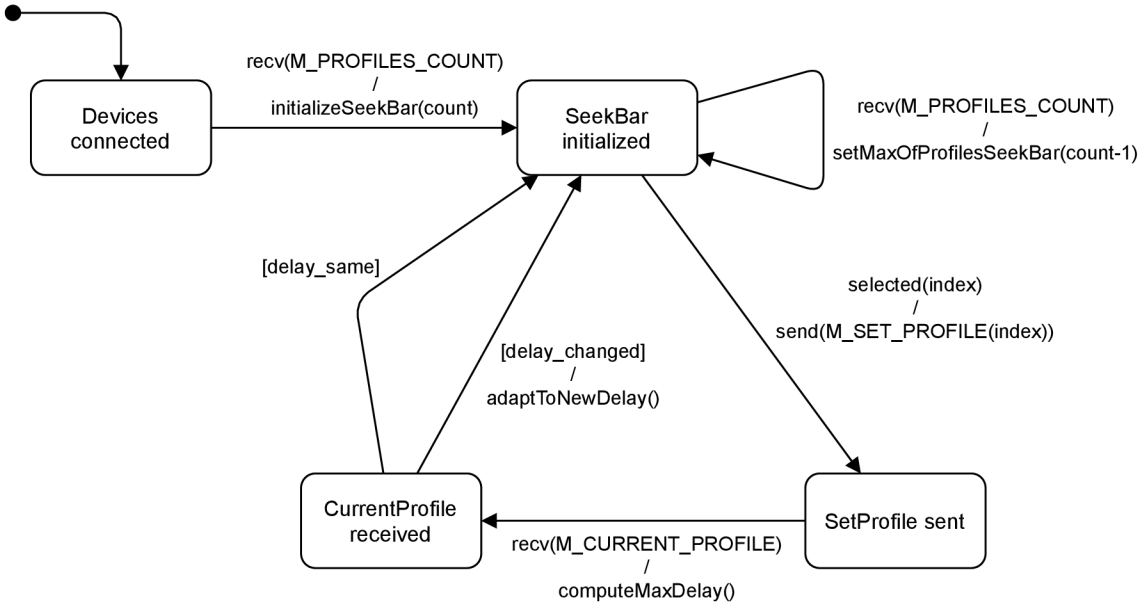
Nastavení profilu kamery

V rámci vzdáleného ovládání kamerového zařízení lze z displeje nastavit kvalitu výstupu prostřednictvím *posuvníku*. Proto je nutné informovat displej zařízení o počtu nastavitelných profilů kamery. V tomto kontextu rozumíme profilem **rozlišení** snímků videa. V úvahu jsou brány pouze profily odpovídající aktuálnímu nastavení *poměru stran*, a tyto jsou pro účely výběru řazeny vzestupně. Proto stačí, aby displej zařízení znalo pouze aktuální počet nastavitelných profilů. Právě ten je jedinou vlastností, která je ve zprávě `M_PROFILES_COUNT` obsažena, viz výpis 7.4. Tato zpráva je zasílána vždy bezprostředně po začátku komunikace, a také kdykoli dojde ke změně konfigurace kamerového zařízení – například k přepnutí kamery.

```
{
    "M_PROFILES_COUNT":{
        "currentProfilesCount": 8
    }
}
```

Výpis 7.4: Zpráva typu počet profilů kamery. Tuto zprávu posílá kamerové zařízení do displejového, aby bylo možné vzdáleně nastavit kvalitu video stopy kamerového zařízení.

V opačném směru lze zasílat zprávu `M_SET_PROFILE`, která specifikuje požadovaný profil. K výběru profilu slouží uživateli posuvník, jehož maximální hodnota odpovídá nejvyššímu



Obrázek 7.5: Stavový diagram popisující chování displejového zařízení při vzdáleném nastavování profilů kamery. Zápis `recv(x)` značí událost příchodu zprávy `x`, podobně `send(y)` znamená odeslání zprávy `y`. Diagram ukazuje, že první příchod zprávy `M_PROFILES_COUNT` způsobí inicializaci posuvníku, kterým se profily nastavují. Příchod zprávy `M_CURRENT_PROFILE` může a nemusí způsobit změnu maximálního možného zpoždění, což může vést například ke zkrácení aktuálního zpoždění, v některých případech i k nutnosti alokovat více paměti pro kruhový buffer.

indexu v poli aktuálních profilů, které zná pouze kamerové zařízení. Index tedy může nabývat hodnot z množiny $\{0, 1, ..., \text{currentProfilesCount} - 1\}$. Vyšší index znamená kvalitnější výstup, což jde ruku v ruce s tím, že maximum hodnoty posuvníku odpovídá nejlepšímu profilu. V reakci na obdržení zprávy `M_SET_PROFILE` vybírá kamerové zařízení dle indexu ve zprávě daný profil z pole aktuálních možných profilů. Bezprostředně po nastavení profilu odesílá kamerové zařízení do displejového zprávu `M_CURRENT_PROFILE`. Tato zpráva je potvrzením změny profilu, ale zároveň hraje důležitou roli při vyhodnocování maximální možné délky zpoždění. Ta totiž závisí právě na aktuálním profilu, pokud je tedy vybrán lepší profil vůči předchozímu, pravděpodobně dojde ke snížení maximální možné délky zpoždění. Pokud by tento mechanismus neexistoval, mohl by kruhový buffer začít přepisovat ještě nevykreslená data snímků, což by vedlo k vykreslování poškozených dat. Stavový diagram na obrázku 7.5 zachycuje události a akce, které zahrnují nastavování profilů kamery z pohledu displejového zařízení.

Nastavení poměru stran

V průběhu procesu uživatelského testování došlo k přidání možnosti nastavení poměru stran libovolné video stopy. Lokální video stopa je nastavována přímo v zařízení, pro vzdálené nastavení bylo třeba implementovat dvě zprávy s podobnou sémantikou, jako v případě nastavení kamerového profilu. Jedná se o zprávy:

- `M_POSSIBLE_AR` – jedná se o zprávu ve směru z kamerového zařízení do displejového. Zpráva obsahuje seznam aktuálně nastavitelných poměrů stran.

```
{
    "M_PROFILES_COUNT": {
        "currentFormatsCount": 6
    },
    "M_POSSIBLE_AR": {
        "currentPossibleAspectRatios": [
            "16:9", "4:3", "3:4", "1:1", "Other"
        ]
    }
}
```

Výpis 7.5: Ukázka spojení dvou zpráv – počet profilů kamery a nastavitelné poměry stran. Tato kombinace zpráv je zasílána z kamerového zařízení bezprostředně po navázání spojení, a také při přepnutí kamery.

- `M_SET_AR` – zpráva určená kamerovému zařízení, která říká, který poměr stran má být nastaven.

Zpráva `M_POSSIBLE_AR` je zasílána bezprostředně po navázání spojení, a také při přepnutí kamery. Přední a zadní kamera totiž často podporují odlišná rozlišení snímků, a tedy obecně i jiné poměry stran. U této zprávy bylo využito toho, že navržený protokol podporuje zasílání více zpráv současně, jak ukazuje výpis 7.5 – při přepnutí kamery je totiž nutné odeslat také zprávu `M_PROFILES_COUNT` popsanou výše. Displejové zařízení v reakci na obdržení zprávy `M_POSSIBLE_AR` inicializuje grafickou komponentu `Spinner` pro výběr poměru stran.

Je-li uživatelem vybrán poměr, odesílá se zpráva `M_SET_AR`, která obsahuje identifikátor poměru stran. Při změně poměru stran je nutné přehodnotit v kamerovém zařízení seznam `_currentPossibleFormats` – více v sekci 7.1.3, načež je zpět do displeje odeslána samostatně zpráva `M_PROFILES_COUNT`.

7.1.5 Bufferování videa

Jednou z největších výzev v průběhu vývoje aplikace Mirrorer byla implementace zpoždění video stopy a s tím spojené bufferování dat videa a jejich zpětné přehrávání. Prvním nepříjemným zjištěním bylo, že WebRTC knihovna pro Android sama o sobě vrací již dekódované snímky videa typu `VideoFrame` a neobsahuje mechanismus, kterým by se dalo ke kódovanému videu přistoupit.

Mé první úvahy vedly k tomu, že nutně potřebuji snímky zpětně zakódovat, jinak bude zpoždění v praxi nepoužitelné – dekódované snímky jsou velmi náročné na paměťový prostor. Po pokusech, které zahrnovaly také studium používání třídy `MediaMuxer` se mi sice podařilo vytvořit kódovanou stopu videa, ukázalo se však, že toto řešení nebude schůdné. Důvodem jsou vysoké nároky na výkon zařízení – v jeden moment by totiž musely probíhat až 3 operace (de)kódování videa **pro každou stopu**[4]. Ne vždy se mi podařilo spustit kódovací mechanismus pro druhou video stopu, což je také popsáno v Android dokumentaci[5].

[4] Jedna operace dekódování uvnitř WebRTC knihovny, jedno kódování pro dočasné uložení stopy a jedno dekódování při vykreslování.

[5] https://developer.android.com/reference/android/media/MediaCodecInfo.CodecCapabilities#getMaxSupportedInstances()

S výše uvedeným souvisí také vyšší spotřeba baterie, delší odezva uživatelského rozhraní a sekání přehrávaných video stop. Bylo tedy nutné najít vhodnější řešení.

Objekty třídy `VideoFrame` jsou předávány jako parametr handleru `onFrame()` třídy `SurfaceViewRenderer`. Ten se již postará o vykreslení snímku na `Surface`. Dalším pokusem tedy bylo ukládat objekty `VideoFrame` do fronty a ve vhodný okamžik je zpětně vykreslovat. Pokud se snímkem chceme nakládat po svém, je potřeba inkrementovat počet referencí na snímek voláním metody `retain()`, což nám zajistí, že data snímku budou danému vláknu aplikace dostupná, dokud není voláno `release()`. Zjednodušený kód první implementace bufferování je vidět ve výpisu 7.6. Zpočátku se zdálo, že tento mechanismus zafungoval a bude stačit experimentálně zjistit, kolik snímků mi knihovna dovolí v paměti zařízení bufferovat.

Problém u vzdáleného videa

Nápad s frontováním snímků byl přímočarý, problém však nastal u frontování snímků *vzdálené* video stopy. Ukázalo se, že v tomto případě má volání funkce `retain()` vedlejší efekt – dokud nebylo voláno `release()`, další snímek do handleru nedorazil. Pokusil jsem se nahlásit tento stav jako chybu do *issue trackeru* WebRTC knihovny[6], z diskuze jsem později usoudil, že funkce `release()` by měla být pro aktuálně zpracovávaný snímek volána vždy nejpozději na konci funkce `onFrame()`. Nápad s pozdržením snímků ve frontě po dobu zpoždění měl tedy poměrně zásadní trhlinu.

Každý objekt `VideoFrame` je pouhou *obálkou* nad objektem implementujícím rozhraní `Buffer`, který obsahuje data daného snímku. `Buffer` je nejčastěji implementován typem `JavaI420Buffer` – a pokud není, obsahuje funkci pro převod do tohoto formátu. Objekty `JavaI420Buffer` ukládají snímky do tří `ByteBuffer` objektů, které obsahují zvlášť složky Y, U a V formátu YUV420 – popsáno v sekci 3.5. Zvolil jsem tedy nový přístup k problému ukládání snímků, potřeboval jsem získat kontrolu nad surovými daty, mít data uložená ve vlastních strukturách a sám si rozhodovat, kdy budou uvolněna z paměti. Každý z Y, U, V bufferů je s příchodem snímku vykopírován do vlastní paměťové struktury a je pro ně vytvořena obálka třídy `VideoFrame`. Původní objekt `Buffer` je uvolněn, protože data z něj již jsou ve vlastních strukturách. Takto vytvořený snímek je umístěn do výše popsané fronty snímků, původní algoritmus vybírání z fronty zůstal zachován.

Kruhový buffer

Při implementaci mechanismu kopírování dat z existujících bufferů bylo třeba ověřit, zda tento krok nebude zkázou pro méně výkonná zařízení. Praxe však ukázala, že samotné kopírování nemá pozorovatelný dopad na výkon aplikace. Co jsem však doposud nezmínil je struktura, do které jsou data snímků kopírována. Jedná se o variantu *kruhového bufferu*[7] s následujícími vlastnostmi:

1. Do bufferu jsou sekvenčně vkládány trojice objektů `ByteBuffer`, kdy každá trojice reprezentuje data jednoho snímku.

2. Je-li kapacita bufferu vyčerpána, ukazatel zápisu se vrací zpět na začátek a přepisuje stará data novými.

[6] https://bugs.chromium.org/p/webrtc/issues/detail?id=12272
[7] https://en.wikipedia.org/wiki/Circular_buffer

```
@Override
public synchronized void onFrame(VideoFrame frame) {
    if (_delayMillis == 0) {
        super.onFrame(videoFrame);
        return;
    }
    frame.retain();
    while(true) {
        // Get next frame from queue.
        VideoFrame nextFrame = _framesQueue.peek();

        // If there are no more frames in queue, there's nothing to paint.
        if (nextFrame == null)
            break;

        // Get time difference between incoming frame and next frame in queue.
        long diffMillis =
            (frame.getTimestampNs() - nextFrame.getTimestampNs()) / 1000000;

        if (diffMillis >= _delayMillis) {
            // If it is time to paint next frame from queue,
            // withdraw it, paint it and release it.
            // Don't break loop because following frame has to be checked.
            nextFrame = _framesQueue.poll();
            super.onFrame(nextFrame);
            nextFrame.release();
        } else {
            // Otherwise break loop.
            break;
        }
    }

    _framesQueue.add(frame);
}
```

Výpis 7.6: První implementace bufferování snímků, která fungovala pouze pro lokální video stopu, princip vybírání z fronty však zůstal totožný také v aktuální implementaci. V případě nenulového zpoždění byly snímky z fronty vykreslovány, pokud rozdíl časových razítek aktuálního snímku a snímku na začátku fronty byl menší než nastavené zpoždění.

Rozlišení	1 snímek (MB)	Zpoždění (ms)	Velikost kruh. bufferu (MB)
1920×1080	3,1	$1000 + 200$	112
1280×720	1,4	$5000 + 200$	215,7
640×360	0,3	$15000 + 200$	157,6

Tabulka 7.1: Tabulka s příklady velikostí kruhového bufferu v závislosti na rozlišení a zpoždění stopy. V kontextu aplikace Mirrorer uvažujeme pouze režim 30 snímků za sekundu. Velikost snímku odpovídá počtu pixelů snímku vynásobených konstantou 1,5 (12 bitů na pixel). Ke skutečnému zpoždění je v praxi přičítána malá hodnota – v tomto případě 200 ms – aby vznikla dostatečná rezerva mezi právě zapisovanými a právě zobrazovanými daty a nedošlo tak ke kolizi.

3. Velikost bufferu je ovlivněna **délkou zpoždění** video stopy a **rozlišením** snímků. Tabulka 7.1 ukazuje příklady velikostí bufferu v závislosti na těchto parametrech. Buffer musí být dostatečně velký, aby nedocházelo k přepisování ještě nevykreslených snímků novými daty.

4. Paměť pro buffer je alokována pouze tehdy, pokud je třeba ji zvětšit.

Kruhový buffer je vhodná struktura pro ukládání dat snímků především z toho důvodu, že při běžných operacích zápisu a čtení není potřeba alokovat a uvolňovat operační paměť. Operace alokování paměti je spojena pouze se změnou zpoždění nebo rozlišení, což jsou relativně nahodilé události. Pokud by byla paměť alokována příliš často, téměř jistě by se to na výkonu aplikace projevilo, jelikož se jedná o netriviální operaci[8]. Další výhodou tohoto přístupu je existence jednotlivých `VideoFrame` objektů, které obsahují odkazy do kruhového bufferu v podobě tří `ByteBuffer` objektů. Není tedy třeba vytvářet další mechanismus pro čtení z kruhového bufferu – toto nám automaticky zajistí operace vykreslení objektu `VideoFrame`. Navíc odpadá starost s uvolňováním paměti, toto je zcela v kompetenci kruhového bufferu.

Zbývalo vyřešit poslední otázku – kolik paměti může kruhový buffer využít aniž by operační systém zařízení začal protestovat a vyvolal chybový stav `OutOfMemoryError`. Tvrdý limit je dán samozřejmě *hardwarovými* komponentami daného zařízení. Objekt třídy `ActivityManager` poskytuje řadu metod, kterými lze získat informace o velikosti dostupné operační paměti. První verze aplikace operovaly s hodnotou, kterou získáme voláním funkce `getMemoryClass()`. Tato vrací přibližnou hodnotu v jednotkách MB, která říká, jaký by měl být paměťový limit pro aplikaci tak, aby operační systém fungoval co nejlépe. Hodnota se u dnešních zařízení pohybuje většinou v nižších stovkách MB. Mým záměrem však bylo dát uživatelům více volnosti při nastavování zpoždění video stop, s tímto řešením jsem se tedy nespokojil a pátral jsem dále. Zjistil jsem, že pokud alokujeme paměť mimo *Java heap* (náš případ), limit v podstatě neexistuje a je na zodpovědnosti vývojáře, kolik paměti dovolí alokovat.[9]. Samozřejmě bylo nutné mít představu o tom, kolik paměti je v zařízení aktuálně volné. K tomuto účelu slouží funkce `getMemoryInfo()` třídy `ActivityManager`. Tato plní objekt třídy `MemoryInfo` z něhož byly pro můj účel zajímavé vlastnosti

- `availMem` – množství aktuálně dostupné paměti na systému v bajtech a

[8] https://en.wikibooks.org/wiki/Optimizing_C%2B%2B/Writing_efficient_code/Allocations_and_deallocations

[9] https://groups.google.com/g/android-ndk/c/lcnwzszrESo/m/wYpPk5BNC-QJ

Zařízení	$\mathcal{M}_C$ (MB)	$\mathcal{M}_{SL}$ (MB)	$\mathcal{M}_{SH}$ (MB)
Huawei P9 Lite	192	525	686
Huawei P20 Lite	384	1087	1364
Lenovo Yoga Smart Tab 4	192	1304	1633

Tabulka 7.2: Tabulka srovnávající různá zařízení z pohledu přístupů ke zjišťování dostupné paměti. Sloupec $\mathcal{M}_C$ ukazuje pro dané zařízení návratovou hodnotu z volání funkce `getMemoryClass()`. Hodnoty ve sloupci $\mathcal{M}_{SL}$ udávají bezpečně dostupnou paměť – rozdíl vlastností `availMem` a `threshold` – pokud má zařízení spuštěno několik (5-6) jiných aplikací. Hodnoty ve sloupci $\mathcal{M}_{SH}$ značí aktuální bezpečně dostupnou paměť, pokud nejsou v systému spuštěny žádné jiné aplikace. Data ukazují, že při využití aktuální dostupné paměti zařízení dojde k několikanásobnému zvětšení paměťového prostoru aplikace, což umožní úměrně zvýšit délku zpoždění video stop.

Zařízení	**Rozlišení**	$\mathcal{D}_C$ (s)	$\mathcal{D}_I$ (s)
Huawei P9 Lite	1920 × 1280	1,7	4,7
	640 × 360	15,5	42,3
Huawei P20 Lite	1920 × 1280	3,4	9,7
	640 × 360	31	87,7
Lenovo Yoga Smart Tab 4	1920 × 1280	1,7	11,6
	640 × 360	15,5	105,2

Tabulka 7.3: Praktické znázornění rozdílů v maximální možné délce zpoždění **jedné** video stopy. Data ukazují významné zvětšení maximálního zpoždění při užití varianty s voláním `getMemoryInfo()` – $\mathcal{D}_I$ – oproti původní variantě, kdy byl ve výpočtu používán výstup z funkce `getMemoryClass()` – $\mathcal{D}_C$. Pro výpočet $\mathcal{D}_I$ byly užity hodnoty $\mathcal{M}_{SL}$ z tabulky 7.2.

- `threshold` – prahová hodnota v bajtech. Udává při jaké hodnotě `availMem` začne operační systém ukončovat zbytné procesy za účelem uvolnění paměti.

Je tedy zřejmé, že bychom se, co se paměti týče, neměli dostat přes hodnotu rozdílu těchto dvou vlastností. Tabulka 7.2 ukazuje rozdíly dvou zmíněných přístupů zjišťování velikosti dostupné paměti.

V praxi byl dále stanoven koeficient využití paměti, který říká, že umožníme alokovat maximálně 80 % hodnoty rozdílu vlastností `availMem` a `threshold`. Tato hodnota má přímý vliv na maximální možnou délku zpoždění video stop. Reálné hodnoty maximální délky zpoždění ukazuje tabulka 7.3.

$$\mathcal{M} = \lambda(\mathcal{A} - \tau) \tag{7.1}$$

Rovnice (7.1) shrnuje předešlé odstavce do výpočtu, jehož výsledkem je maximální velikost kruhového bufferu pro ukládání snímků videa. Symbolem $\mathcal{A}$ rozumíme vlastnost `availMem`, symbol τ pak značí vlastnost `threshold` objektu `MemoryInfo`. Parametrem λ lze nastavit procentuální využití bezpečně dostupné paměti, v případě mého řešení je nastaven na hodnotu 80 %.

$$\mathcal{D}_M = \frac{\mathcal{M}}{\text{BPP} \cdot \text{FPS} \cdot \sum_{t \in \mathbf{T}} w_t \cdot h_t} - \Delta, \tag{7.2}$$

Rovnice (7.2) ukazuje, jakým způsobem se počítá maximální povolená délka zpoždění v závislosti na následujících parametrech:

- $\mathcal{M}$ – hodnota získaná výpočtem v rovnici (7.1).
- w_t, h_t – aktuální šířka a výška snímků video stopy t v pixelech. $\mathbf{T}$ je množina momentálně aktivních stop videa.
- BPP – hodnota *bytes-per-pixel* daná formátem snímků, v případě formátu **YUV420** má hodnotu 1,5 (12 bitů na pixel).
- FPS – počet snímků za sekundu, v kontextu mého řešení se jedná o konstantu s hodnotou 30.
- Δ – malá časová hodnota, která zajistí, že nedojde ke kolizi při zápisu do a čtení z kruhového bufferu. Aktuální verze pracuje s hodnotou 0,2 s.

Výsledkem výpočtu je maximální zpoždění $\mathcal{D}_M$ v sekundách. Jedná se o nezápornou reálnou hodnotu, která je zaokrouhlena směrem k nejbližší nižší celočíselné hodnotě z *kosmetických* důvodů.

7.2 Uživatelské rozhraní

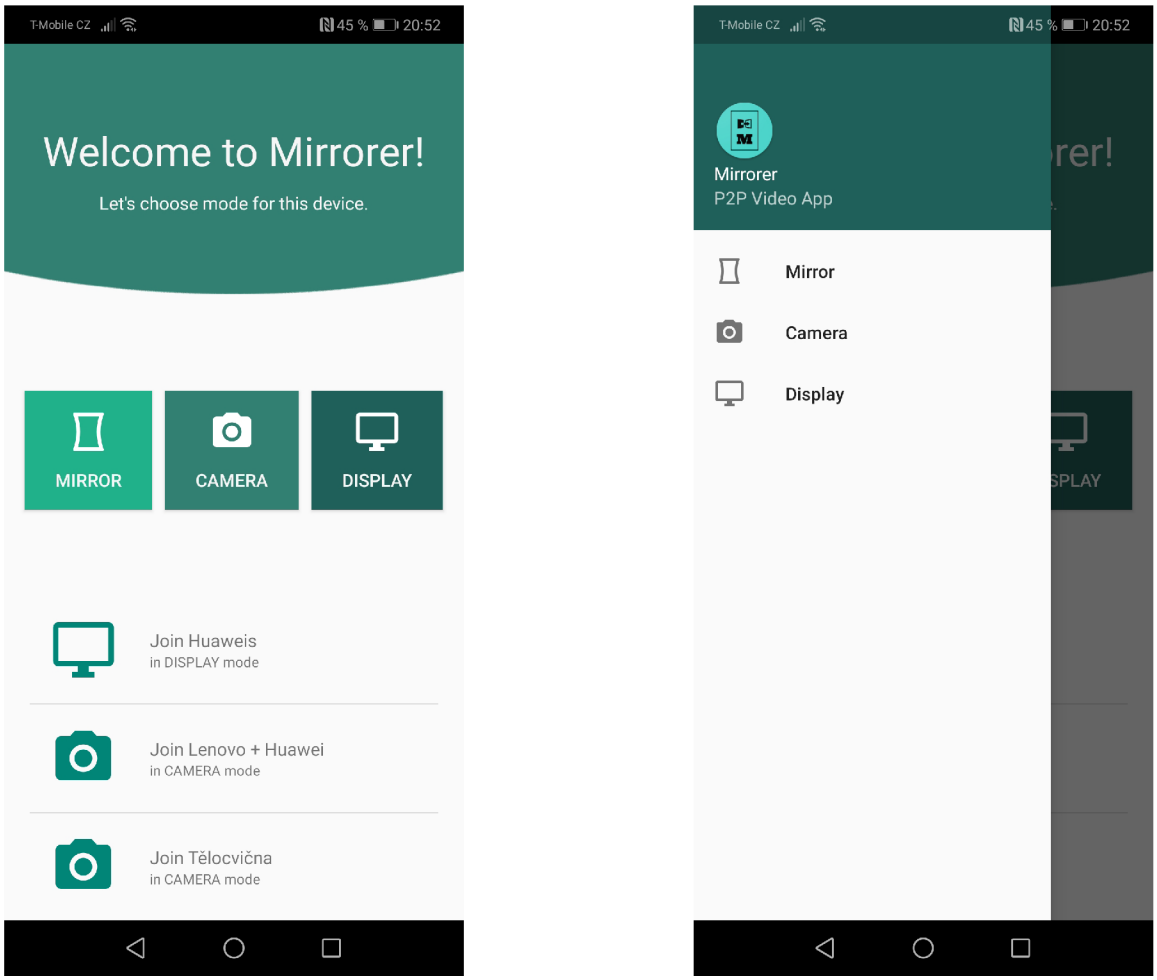
Tato sekce ukáže aktuální verzi uživatelského rozhraní nativní mobilní aplikace **Mirrorer** a webové varianty displej zařízení. Uživatelské rozhraní bude popsáno s ohledem na různé případy užití finálního řešení.

7.2.1 Mirrorer – Android

Při popisu nativní mobilní aplikace **Mirrorer** ukáži jednotlivé obrazovky, vysvětlím k čemu slouží, co se na obrazovce odehrává a do jakého stavu se aplikace z dané obrazovky může dostat. Obrazovky, které jsou spjaty s režimem **kamera** budou zobrazeny na menším displeji, typickým pro mobilní telefony. Obrazovky pro režim **displej** budou zobrazeny na větší obrazovce, která reprezentuje tablet.

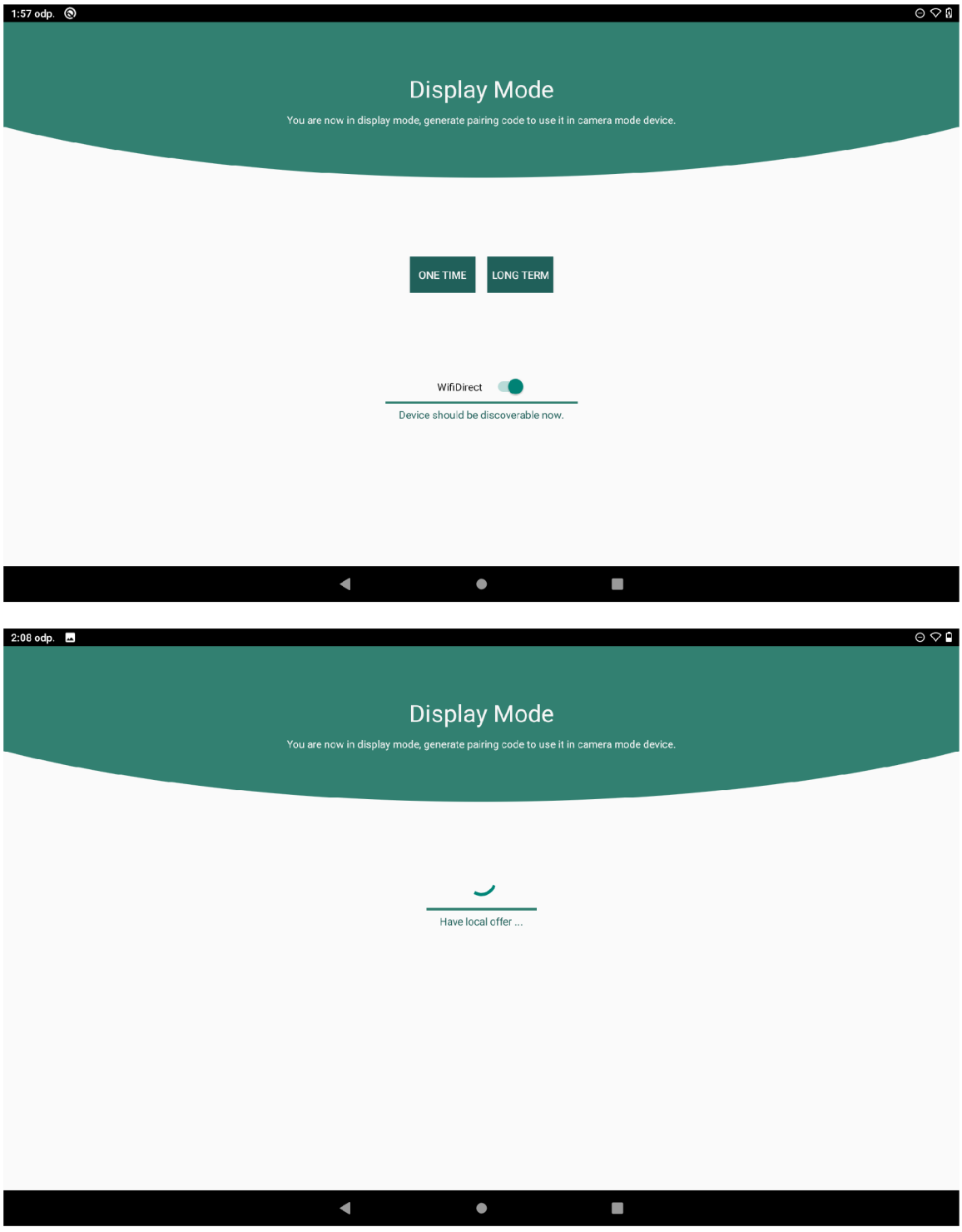
7.2.2 Webový displej

Webová varianta displej zařízení je optimalizována pro použití na větším displeji, například PC monitoru, případně na televizním displeji. Neobsahuje veškerou funkcionalitu režimu displej v nativní Android aplikaci. Jedná se spíše o demonstraci toho, že WebRTC standard lze používat napříč různými platformami. Přesto aplikace obsahuje dostatek přidané hodnoty k tomu, aby již nyní byla použitelná v praxi, což demonstrují následující náhledy obrazovek.

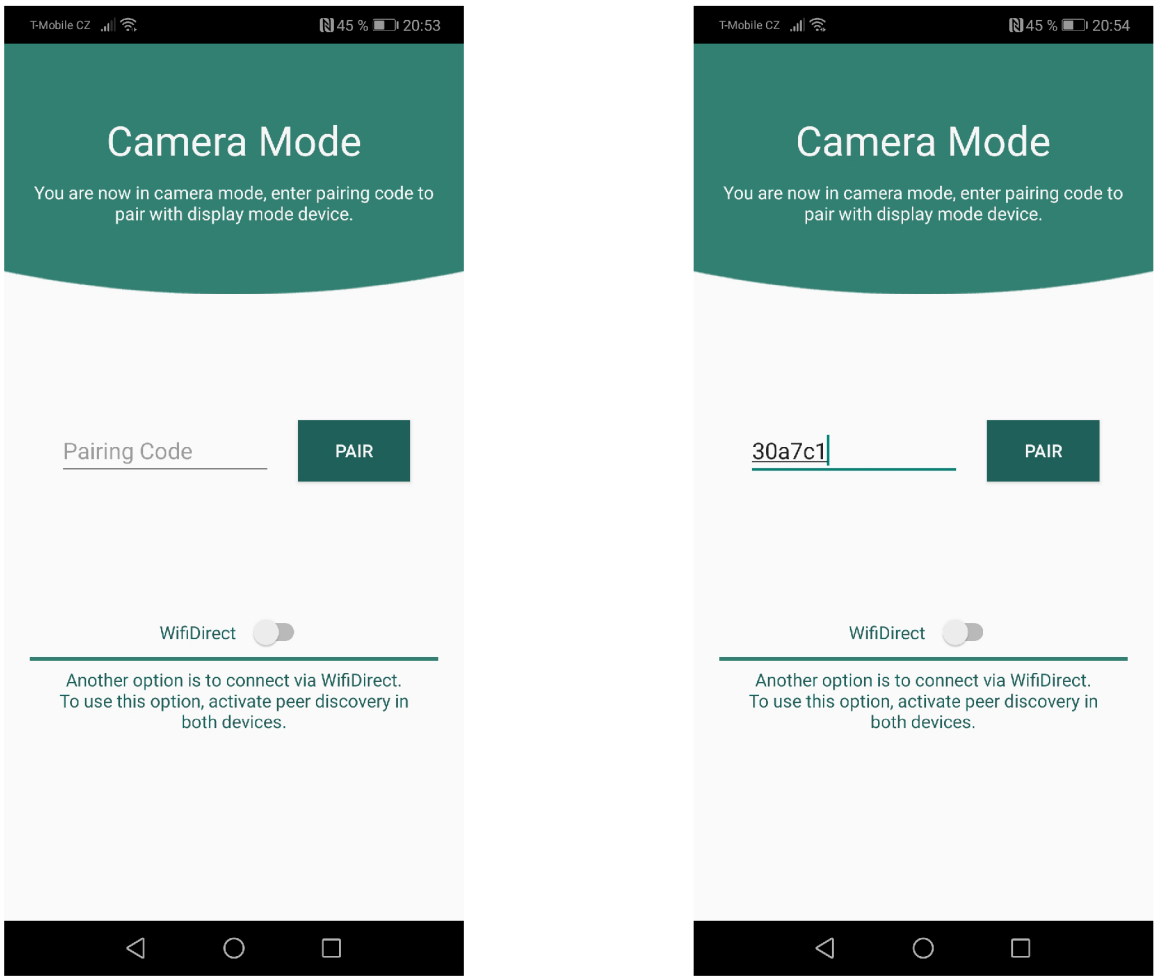


Obrázek 7.6: **Vlevo** je úvodní obrazovka, která je zobrazena bezprostředně po spuštění aplikace. Lze si všimnout úvodního textu, který uživatele vybízí k volbě režimu. Volbu lze provést poklepáním na jedno ze tří tlačítek uprostřed obrazovky. Ve spodní části obrazovky se mohou vyskytovat uložená dlouhodobá spojení, ikonou u každého spojení je indikováno, v jaké roli toto zařízení v daném spojení vystupuje. Poklepáním na jednu z položek se zařízení spustí v daném režimu a pokusí se navázat spojení se vzdáleným zařízením. Na obrazovce **vpravo** vidíte otevřený tzv. *drawer*, který lze vyvolat přejetím od levého okraje obrazovky. Drawer slouží k rychlému přepnutí režimů z jakéhokoli stavu aplikace.

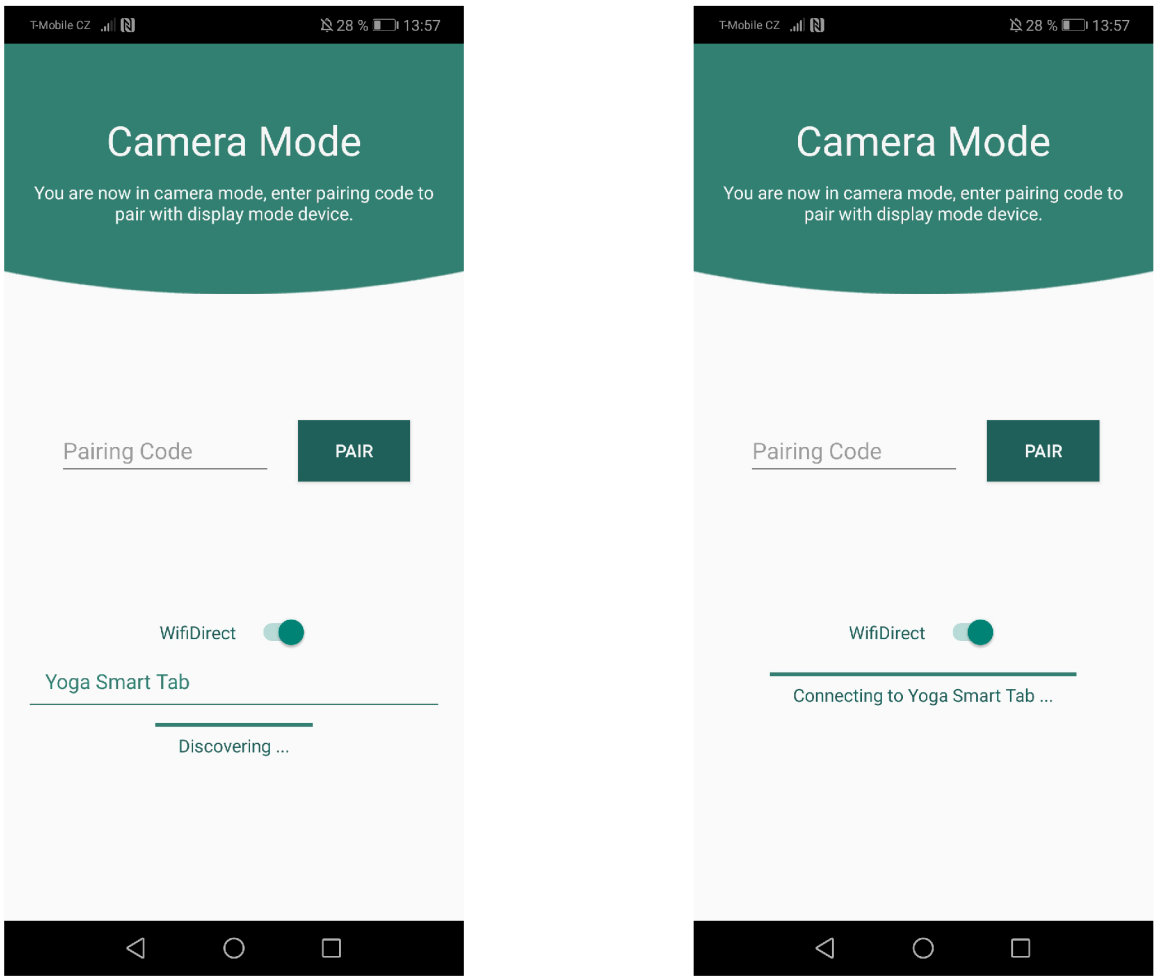
Obrázek 7.7: Obrazovka **nahoře** je hlavní obrazovkou režimu **displej**. Uživatel je vyzván k vygenerování párovacího kódu – jednorázové nebo dlouhodobé varianty. Níže je umístěn přepínač, kterým lze aktivovat Wi-Fi Direct mechanismus – v případě aktivování jsou vysílány okolním zařízením informace o Wi-Fi Direct službě aplikace Mirrorer. Navazování spojení pomocí Wi-Fi Direct ukazuje obrázek 7.8. **Spodní** obrazovka ukazuje vygenerovaný párovací kód, který je určený k zadání v kamerovém zařízení. Jedná se o kód pro dlouhodobou variantu spojení.



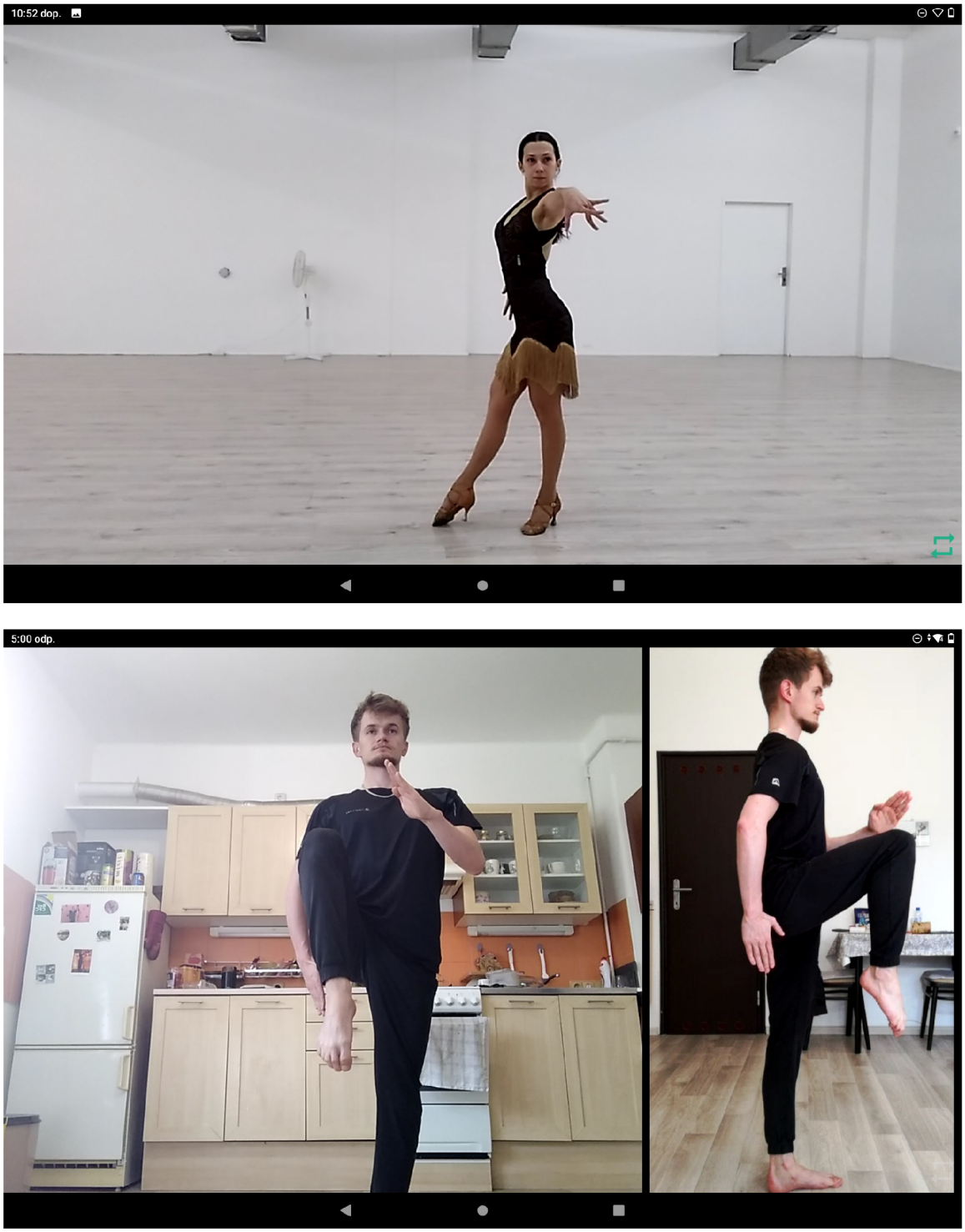
Obrázek 7.8: V **horní** části vidíte obrazovku režimu displej s povoleným Wi-Fi Direct přepínačem. V tomto stavu je zařízení objevitelné kamerovým zařízením, které musí mít rovněž povolený Wi-Fi Direct přepínač. Obrazovka **dole** indikuje, že spojení mezi zařízeními bylo úspěšně navázáno – včetně toho, že displejové zařízení figuruje jako *group owner*. Text na obrazovce říká, že byla vytvořena lokální zpráva typu *offer* protokolu SDP a čeká se na výměnu SDP zpráv s kamerovým zařízením, načež je sestaveno P2P spojení a umožněna komunikace v reálném čase.



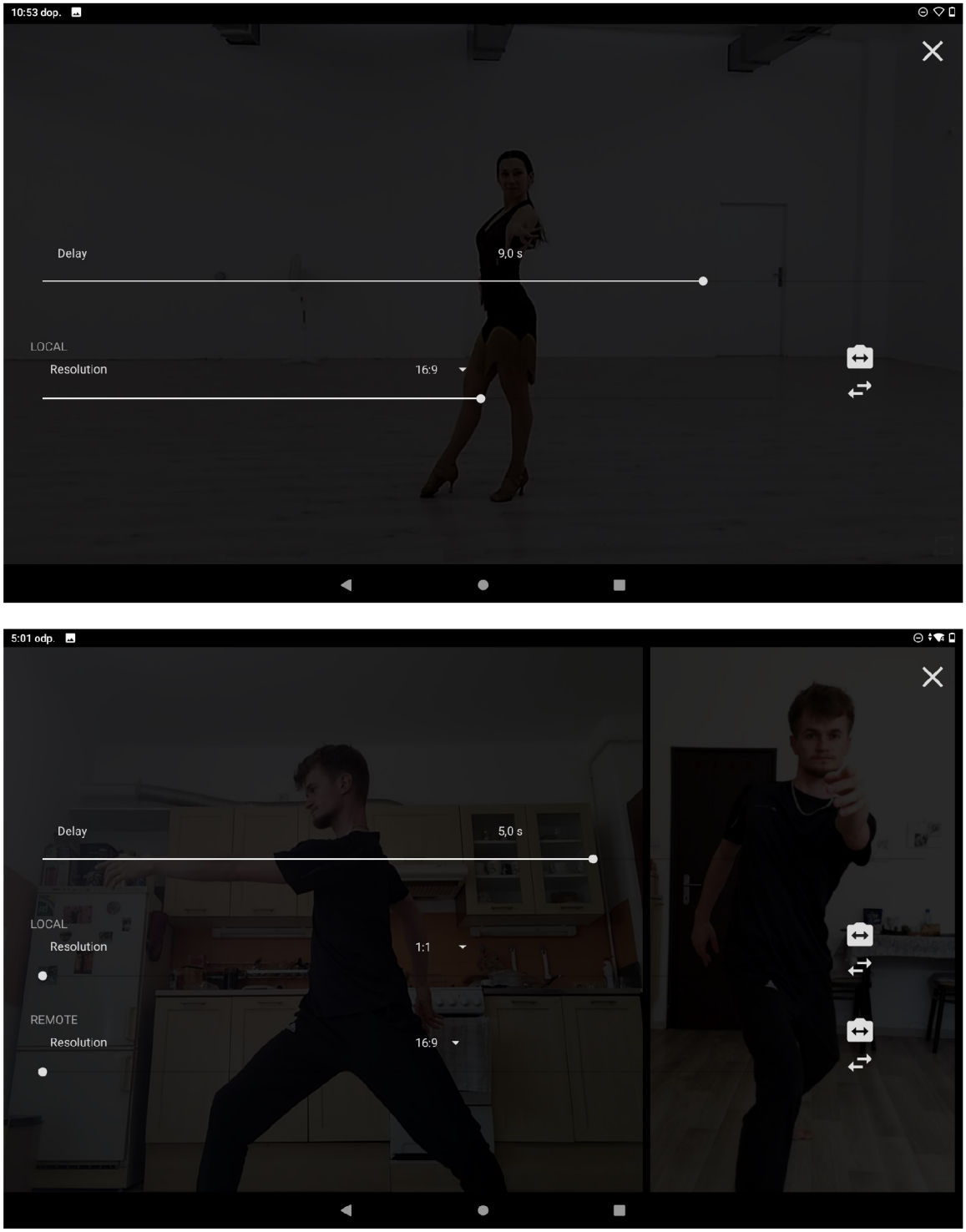
Obrázek 7.9: Na tomto obrázku je vidět hlavní obrazovka režimu **kamera**. Horní část obrazovky má popisnou funkci, vybízí uživatele k zadání *párovacího kódu*, který je generován v displejovém zařízení. Kód může být čtyřmístný pro jednorázové spojení nebo šestimístný pro dlouhodobé spojení. Ve spodní části obrazovky se nachází přepínač pro povolení vyhledávání zařízení prostřednictvím technologie Wi-Fi Direct – ukázka této varianty je na obrázku 7.10.



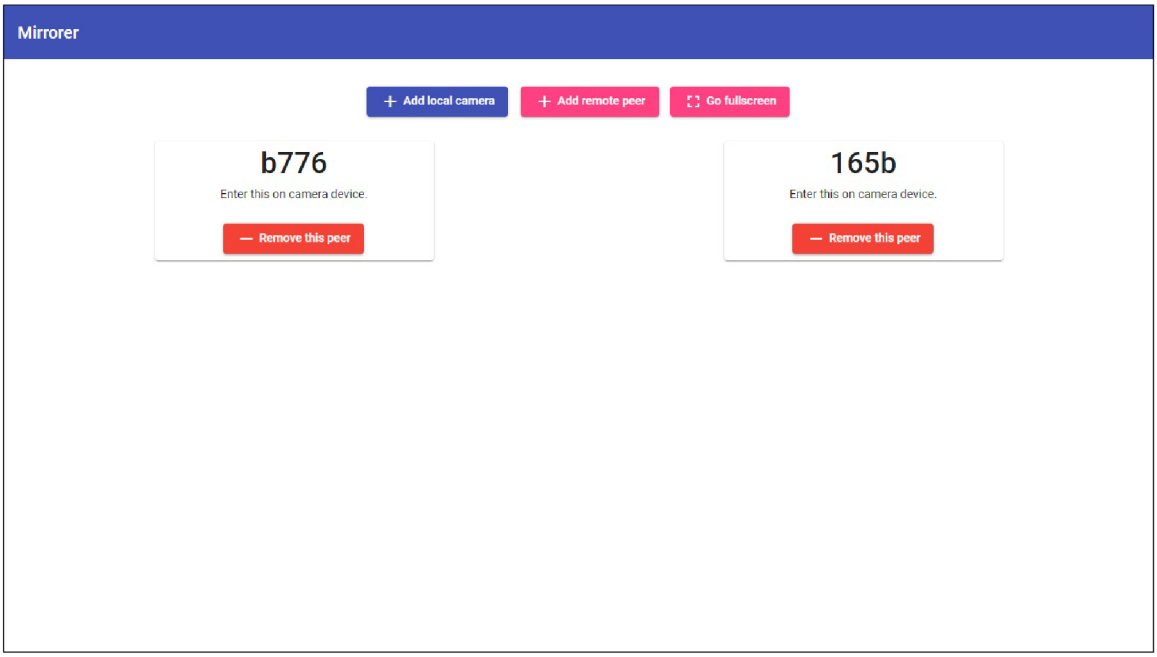
Obrázek 7.10: Tento obrázek ukazuje obrazovky kamerového režimu, na kterých je navazováno Wi-Fi Direct spojení s displejovým zařízením. Obrazovka **vlevo** ukazuje režim, ve kterém je skenováno okolí s cílem objevit Wi-Fi Direct službu, kterou registruje displejové zařízení. Jak je z obrázku zřejmé, vyhovující zařízení jsou přidána do seznamu. Poklepáním na položku seznamu se spustí mechanismus navázání spojení se vzdáleným zařízením, jak je ukázáno na obrazovce **vpravo**.



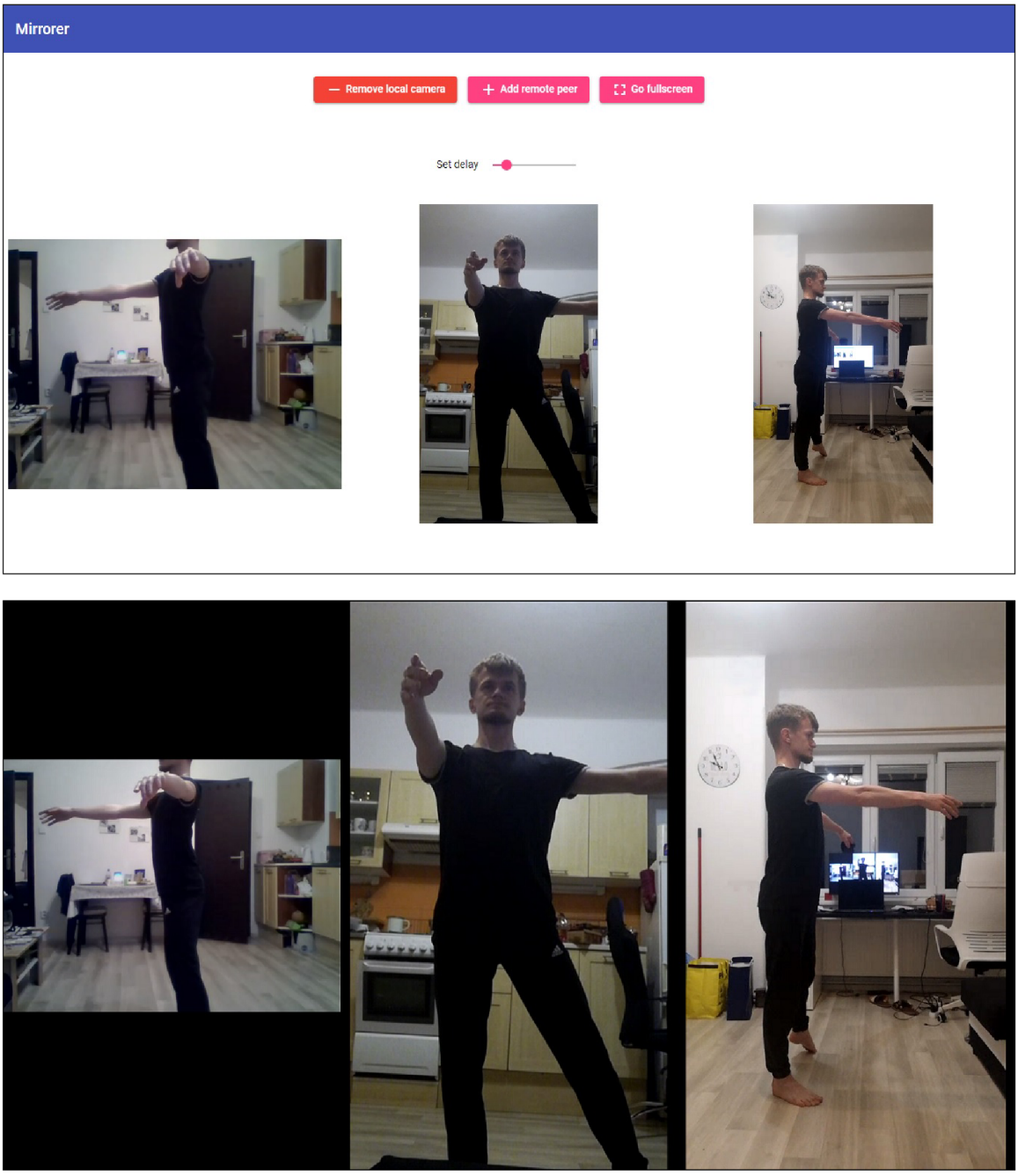
Obrázek 7.11: Ukázka obrazovek, na kterých je promítán video výstup z kamery v reálném čase. **Horní** obrazovka ukazuje zařízení v režimu zrcadlo – tedy pouze s lokální video stopou. Na obrazovce **dole** je zaznamenán stav po úspěšné výměně signalizačních informací mezi kamerovým a displejovým zařízením a zahájení přenosu vzdálené video stopy. Obrazovka ukazuje, že rozložení video výstupů na displeji lze přizpůsobit daným okolnostem.



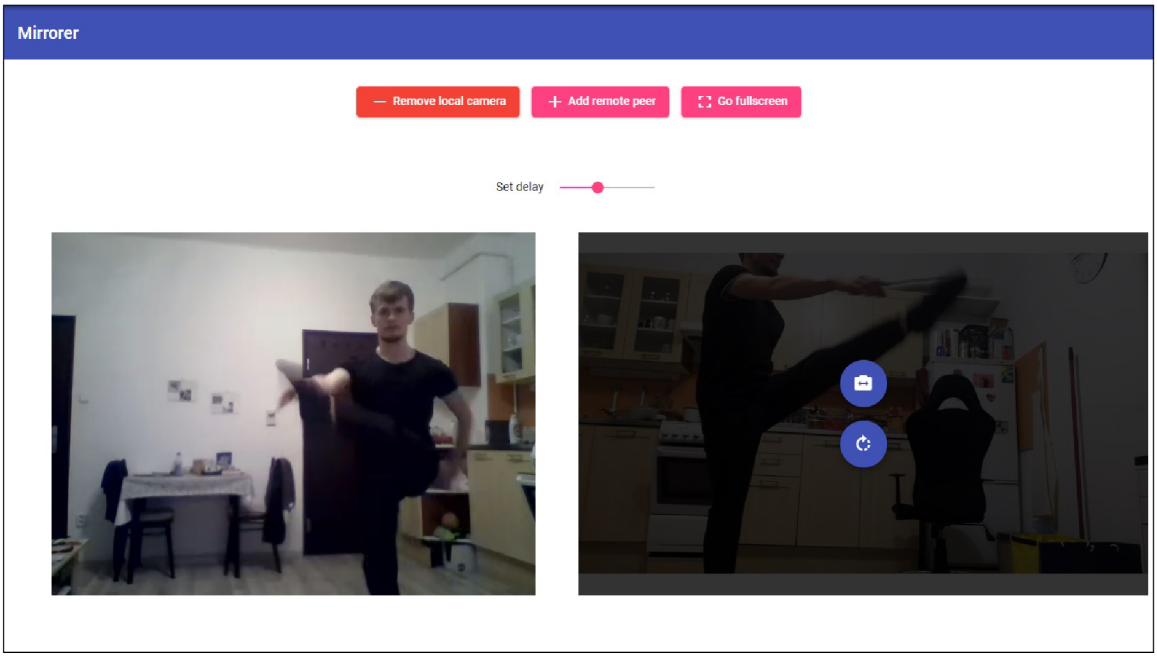
Obrázek 7.12: Poklepáním na displej daného zařízení lze vyvolat nastavení video výstupů. Na obrazovce **nahoře** je přehled nastavení v režimu zrcadlo. V tomto režimu je možné u lokální video stopy nastavit zpoždění, poměr stran, kvalitu (rozlišení), zrcadlení obrazu, a také lze přepínat mezi přední a zadní kamerou. Na **spodní** obrazovce vidíme nastavení v režimu displej. Oproti režimu zrcadlo zde navíc figuruje nastavení vzdálené video stopy. Zpoždění je nastavováno jednotně pro lokální i vzdálenou stopu.



Obrázek 7.13: Zde je vyobrazena webová varianta displej zařízení. Na této obrazovce vidíme ovládací tlačítka a vygenerované dvě instance objektů, které reprezentují vzdálené peery. Tuto instanci lze vygenerovat kliknutím na tlačítko `Add remote peer` a může jich být teoreticky neomezené množství. Po kliknutí na tlačítko `Add local camera` se prohlížeč pokusí přistoupit ke kamerovému rozhraní daného zařízení a v případě úspěchu zobrazí lokální video výstup. Tlačítko `Go fullscreen` je užitečné zejména po sestavení spojení se vzdálenými kamerovými zařízeními a zahájení přenosu video stop v reálném čase. U vygenerovaných peerů vidíme kódy, které lze použít pro spárování v kamerovém zařízení.



Obrázek 7.14: Tyto obrazovky demonstrují výhodu webového řešení, která spočívá v teoreticky neomezeném počtu vzdálených kamerových zařízení. Oba obrázky obsahují video výstup z lokální kamery (vlevo) a dvou vzdálených zařízení, která jsou připojena prostřednictvím nativní mobilní aplikace Mirrorer v režimu kamera. Obrazovka **nahoře** ukazuje nový ovládací prvek, kterým je posuvník umožňující nastavit délku zpoždění video stop. **Spodní** obrazovka ukazuje stav aplikace po aktivování režimu *fullscreen*.



Obrázek 7.15: Obrazovka ukazuje aktivní lokální a jednu vzdálenou video stopu. U vzdálené video stopy si lze všimnout překrytí videa ovládacími prvky, prostřednictvím kterých lze otáčet videem, je-li to nutné a přepnout vzdálenou kameru. Ovládací prvky jsou vyvolány při přejetí kurzoru nad oblastí, ve které je daná video stopa vykreslována.

Kapitola 8

Závěr

Cílem této diplomové práce bylo vytvořit *chytré digitální zrcadlo*, které slouží sportovcům z oblasti individuálních sportů při nácviku techniky a koordinace libovolných prvků. Důraz byl kladen na potlačení nevýhod běžného zrcadla, jako je nutnost soustředit se na sledování prvku již během exekuce, nepřirozená pozice hlavy a nemožnost provedení prvku zpětně analyzovat.

V úvodní části práce jsem se seznámil s technologiemi, které souvisí se zpracováním obrazu z kamery a přenosem videa v reálném čase. Byly identifikovány možné případy užití a definovány hlavní entity systému, což podněcovalo k vytvoření konceptu, ve kterém vystupují zařízení v režimech zrcadlo, kamera a displej. V další fázi bylo vytvořeno minimální možné řešení v podobě první verze nativní mobilní aplikace Mirrorer.

V jisté fázi vývoje bylo potřeba zařadit do návrhu systému také webový signalizační server, který umožňuje párování dvou a více zařízení fungujících v režimu displej či kamera. Kompletní systém zahrnující mobilní aplikaci Mirrorer a webový signalizační server byl implementován. V rámci iterativního uživatelského testování byl systém pravidelně vylepšován – zejména se jednalo o přidávání nových funkcí do aplikace Mirrorer. Na základě posledních fází uživatelských testů byla vytvořena také webová varianta aplikace figurující v režimu displej.

Výsledky uživatelského testování ukázaly, že systém digitálního zrcadla lze využívat napříč sportovními odvětvími, kde již dnes pomáhá sportovcům zlepšovat své individuální technické dovednosti na základě zpětné vazby, které toto řešení poskytuje. Navíc, díky webové variantě displej zařízení, se celé řešení stalo částečně multiplatformním a má tedy potenciál oslovit větší množství uživatelů.

8.1 Budoucí vývoj

Další vývoj vytvořeného produktu by se měl věnovat zejména následujícím bodům, které jsou seřazeny sestupně dle priority:

1. Přidat funkce webového displeje takové, aby se stal plnohodnotnou alternativou displejového režimu nativní mobilní aplikace Mirrorer.
2. Nabídnout systém k testování širšímu okruhu uživatelů.
3. Implementovat funkce, které byly identifikovány jako vhodné případy užití aplikace – například možnost uložit si v aplikaci Mirrorer fragment videa získaný aktivací *playback loop* módu.

4. V aplikaci Mirrorer vylepšit práci s pamětí, zvážit vlastní mechanismus jednoduchého kódování snímků.
5. Zvážit možnost zaznamenávat také zvukovou stopu – bylo by využito v tréninku tanečních sportů.
6. V případě úspěchu současného řešení zajistit vývoj pro platformu iOS.

Literatura

[1] Camps Mur, D., Garcia Saavedra, A. a Serrano, P. Device-to-device communications with Wi-Fi Direct: overview and experimentation. *IEEE Wireless Communications Magazine*. Červen 2013, sv. 20. DOI: 10.1109/MWC.2013.6549288.

[2] Cheng, F. *Build Mobile Apps with Ionic 4 and Firebase*. Sandringha, Auckland, New Zealand: Apress, 2018. ISBN 978-1-4842-3775-5.

[3] Davenport, T. H. a Prusak, L. *Working Knowledge*. Brighton, MA 02135 USA: Harvard Business School Press, leden 1998. ISBN 9780875846552.

[4] Doroftei, D., De Cubber, G., Wagemans, R., Matos, A., Silva, E. et al. User-Centered Design. In:. Srpen 2017. DOI: 10.5772/intechopen.69483. ISBN 978-953-51-3375-9.

[5] Gamma, E., Helm, R., Johnson, R. a Vlissides, J. M. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, říjen 1994. ISBN 978-0-201-63361-0.

[6] Gary Sullivan, S. E. Recommended 8-Bit YUV Formats for Video Rendering. Listopad 2008, [cit. 2021-04-30]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/medfound/recommended-8-bit-yuv-formats-for-video-rendering?redirectedfrom=MSDN.

[7] Griffith, C. *Mobile App Development with Ionic, Revised Edition*. United States of America: O'Reilly Media, Inc., 2017. ISBN 978-1-491-99812-0.

[8] Guilizzoni, P. What Are Wireframes? Srpen 2020, [cit. 2021-04-15]. Dostupné z: https://balsamiq.com/learn/articles/what-are-wireframes/.

[9] *Ergonomics of human–system interaction – Part 210: Human-centred design for interactive systems*. Standard. Geneva, Switzerland: International Organization for Standardization, 2010.

[10] Ivov, E., Rescorla, E., Uberti, J. a Saint Andre, P. *Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol* [Work in progress]. Internet-Draft 21. Duben 2018. Dostupné z: https://tools.ietf.org/html/draft-ietf-ice-trickle-21.

[11] Keranen, A., Holmberg, C. a Rosenberg, J. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal* [Internet Requests for Comments]. RFC 8445. červenec 2018. Dostupné z: https://www.rfc-editor.org/info/rfc8445.

[12] Kocsmárszky, Z. RITE Method: Comprehensive guide for Rapid Iterative Testing and Evaluation. Červen 2020, [cit. 2020-12-27]. Dostupné z: https://www.hellopingpong.com/blog/rite.

[13] Levent Levi, T. 10 Massive Applications Using WebRTC. Prosinec 2017, [cit. 2020-12-29]. Dostupné z: https://bloggeek.me/massive-applications-using-webrtc/.

[14] Lupton, D. The Sociology of Mobile Apps. Květen 2020, s. 2.

[15] Manson, R. *Getting Started with WebRTC*. Birmingham, UK: Packt Publishing Ltd., září 2013. ISBN 978-1-78216-630-6.

[16] Marsch, J. *UX pro začátečníky (rychlokurz - 100 lekcí)*. Brno: Zoner Press, 2019. ISBN 978-80-7413-397-8.

[17] MDN Contributors. *Introduction to the Real-time Transport Protocol (RTP)* [online]. Mozilla Developer Network [cit. 2020-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Intro_to_RTP.

[18] Medlock, M. C., Wixon, D., Terrano, M. a Romero, R. L. Using the RITE method to improve products; a definition and a case study. In:. 2007.

[19] Morris, J. W. a Elkins, E. There's a History for That: Apps and Mundane Software as Commodity. *The Fibreculture Journal*. Listopad 2015, s. 70. ISSN 1449-1443.

[20] Perkins, C., Handley, M. a Jacobson, V. *SDP: Session Description Protocol* [Internet Requests for Comments]. RFC 4566. červenec 2006. Dostupné z: https://www.rfc-editor.org/info/rfc4566.

[21] PlaybookUX Authors. 10 Popular Usability Testing Methods. [cit. 2021-01-02]. Dostupné z: https://www.playbookux.com/10-popular-usability-testing-methods/.

[22] Rosenberg, J., Weinberger, J., Huitema, C. a Mahy, R. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)* [Internet Requests for Comments]. RFC 3489. Březen 2003. Dostupné z: https://www.rfc-editor.org/info/rfc3489.

[23] Schoonwinkel, D. *Practical Measurements of Wi-Fi Direct in Content Sharing, Social and Gaming Android Applications*. Matieland 7602, South Africa, 2016. Diplomová práce. Stellenbosch University.

[24] Schulzrinne, H., Casner, S., Frederick, R. a Jacobson, V. *RTP: A Transport Protocol for Real-Time Applications* [Internet Requests for Comments]. Standard 64. červenec 2003. Dostupné z: https://www.rfc-editor.org/info/std64.

[25] Wodtke, C. The Intuitive and the Unlearnable. Květen 2018, [cit. 2020-12-28]. Dostupné z: https://cwodtke.medium.com/the-intuitive-and-the-unlearnable-cccffd9a762.

Příloha A

Obsah přiloženého CD

Přiložené CD obsahuje následující položky:

- Adresář `mirrorer`, který obsahuje Android projekt včetně zdrojových a spustitelných souborů nativní mobilní aplikace Mirrorer.
- Adresář `mirrorer-web`, který obsahuje Angular projekt včetně zdrojových a kompilovaných souborů webové varianty displej zařízení.
- Adresář `web-signaling`, který obsahuje PHP skripty pro provoz REST API webového signalizačního serveru.
- Adresář `source-tex`, který obsahuje zdrojový tvar této písemné zprávy.
- Soubor `mirrorer-plakat.png` – propagační plakát aplikace Mirrorer.
- Soubor `mirrorer-video.mp4` – prezentační video aplikace Mirrorer. Video je rovněž dostupné online – https://www.youtube.com/watch?v=w3u78CqfcGk.
- Soubor `README.md` – návod k obsahu CD včetně popisu kompilace jednotlivých projektů.

Příloha B

Plakát

Součástí práce je propagační plakát, který je umístěný na přiloženém CD pod názvem `mirrorer-plakat.png`. Zde je zobrazena jeho zmenšená podoba.

Obrázek B.1: Propagační plakát aplikace Mirrorer.