VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Kevin Singh

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

MULTIMEDIÁLNÍ PŘEHRÁVAČ PRO IOS
MULTIMEDIA PLAYER FOR IOS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Kevin Singh**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Štěpán Grabovský**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Kevin Singh ***ID:*** 162255
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Multimediální přehrávač pro iOS

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je realizace knihovny přehrávače multimediálního obsahu s podporou pokročilých služeb jako reklamních formátů, odesílání statistik, zabezpečení obsahu atd. V práci nejdříve definujte vybrané pokročilé služby přehrávače a proveďte analýzu již dostupných řešení. Na základě zjištěných informací poté realizujte funkční knihovnu přehrávače, která bude implementována v jednoduché testovací aplikaci. Celý projekt musí být vytvořen v jazyce Swift.

DOPORUČENÁ LITERATURA:

[1] NEUBURG, Matt. IOS 11 programming fundamentals with Swift: Swift, Xcode and Cocoa basic. Fourth edition. Sebastopolo, CA: O´Reilly, 2018. ISBN 978-1-491-99931-8.

[2] HOFFMAN, Jon. Mastering Swift. 1. Packt Publishing, 2015. ISBN 9781784392154.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: Ing. Štěpán Grabovský
Konzultant:

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

ABSTRAKT

Diplomová práce „Multimediální přehrávač pro iOS“ se zabývá vývojem pokročilého přehrávače v programovacím jazyce Swift. Vyvinutý přehrávač umožňuje kromě interaktivního přehrávání videa také zobrazení reklam, titulek, změnu kvality přehrávání, AirPlay a stahování obsahu pro offline přehrávání. Přehravač byl vyvinut jako knihovna a umožňuje také zasílat statistická data do analytického nástroje *Google Analytics*. V rámci této práce vznikla i testovací aplikace, jenž tuto vyvinutou knihovnu v sobě implementuje a dokazuje tak funčknost jednotlivých funkcionalit.

KLÍČOVÁ SLOVA

iOS, Swift, přehrávač, multimediální, HLS, MP4, DRM, VAST, video, framework, stahování, titulky, AirPlay, kvalita videa, aplikace, iPhone

ABSTRACT

Diploma work „Multimedia player for iOS“ deals with the development of advanced player in Swift programming language. Developed player besides playing interactive videos is able to show ads, subtitles, can change the video quality, AirPlay and download the content for offline playing. The player was developed as a library and is able to send statistic data to Google Analytics as well. A testing app is also a part of this work that implements this framework and proves the functionality of the individual features.

KEYWORDS

iOS, Swift, player, multimedia, HLS, MP4, DRM, VAST, video, framework, downloading, subtitles, AirPlay, video quality, app, iPhone

SINGH, Kevin. *Multimediální přehrávač pro iOS*. Brno, Rok, 100 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Štěpán Grabovský

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Multimediální přehrávač pro iOS“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Štěpánovi Grabovskému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

V moderní době je předání informací pomocí textu na ústupu a rozvijí se sdílení pomocí audia a videa. K tomu, aby bylo možné tuto informaci přehrát na mobilním zařízení se systémem iOS, je potřebné tento obsah otevřít v přehrávači, který se stává nedílnou součástí mobilních aplikací. Moderní přehrávače musí zvládnout přehrávání *VoD* z internetu, v různých podobách i formátech. Musí také umožnit uživateli zasáhnout do funkcionalit přehrávače (posun v čase, znovuzpuštění v daném čase, zobrazit titulky, změna kvality, uložení do paměti, atd.). U těchto přehrávačů chtějí mít poskytovatelé přehled o chování uživatele a také zobrazovat reklamy, které jsou jejich nejčastějším zdrojem příjmů.

Diplomová práce se zabývá vývojem pokročilého přehrávače, který nabízí širokou škálu funkcí. Nejprve se věnuje teoretickému popisu technologií použitých ve vyvinutém přehrávači. Popsány jsou formáty MP4, HLS, MPEG-Transport Stream a MPEG DASH. Dále navazuje popis technologie DRM (Digital Rights Management), která je využita například systémy *FairPlay*, *PlayReady* a *Widevine*. Poté je věnována pozornost typu zabezpečení HLS-AES, analytickému nástroji *Google Analytics* a reklamním formátům VAST (Video Ad Serving Template), VPAID (Video Player-AD API Definition) a VMAP (Video Multiple Ad Playlist).

V diplomové práci také nechybí popis již dostupných řešení, jež se nazývají *Kaltura*, *BitMovin*, *Brightcove* a *AVFoundation*. Výsledkem provedené analýzy je výběr knihovny *AVFoundation* jako základu pro realizovaný přehrávač.

Dále se již práce zaměřuje na samotný vývoj přehrávače pro iOS v nativním programovacím jazyce *Swift*, který by se dal jako knihovna snadno integrovat do jakékoliv aplikace. Kapitola je doplněna ukázkami kódů jednotlivých funkcionalit. Konkrétněji se jedná o samotné přehrávání videa a jeho ovládání, posílání statistik pomocí analytického nástroje, zobrazování titulků, reklam, přepínání kvality rozlišení obrazu, přehrávání zabezpečeného obsahu, podpora *AirPlay* pro přehrávání obsahu na televizi, stahování obsahu pro offline přehrávání a přepínání přehrávaného videa.

Pro ověření funkčnosti vyvinuté knihovny je navazující část práce věnována vývoji testovací aplikace, která zahrnuje seznam videí. Na nich si lze ověřit správnou funkčnost přehrávače.

Poslední část je věnována popisu, jak publikovat aplikaci do virtuálního obchodu *AppStore*.

1 Pokročilé technologie multimediálního přehrávače

V této kapitole jsou popsány formáty, které budou podporovány multimediálním přehrávačem. Přesněji se jedná o MP4, MPEG-TS, HLS a MPEG DASH. Dále jsou zde popsány typy přehrávání zabezpečeného obsahu DRM, jenž se nazývají FairPlay, PlayReady a Widevine. Poté je zde k nalezení popis analytického nástroje Google Analytics, po němž následuje seznámení s reklamními standardy VAST, VPAID a VMAP.

1.1 MP4

Formát kontejneru MP4, též známý jako MPEG-4, jenž umožňuje kombinaci digitálního zvuku a videa. Byl vyvinut společností MPEG a patří mezi standardy ISO/IEC. Je navržen tak, aby obsahoval informace o synchronizovaných médiích ve flexibilním formátu za účelem výměny, správy a úpravy multimediálních souborů [1].

MPEG-4 umožňuje streamování přes internet, multiplexování více video a audio streamů v jednom souboru, variabilní rámcové a bitové rychlosti, titulky a statické obrázky [2].

MP4 je hlavním formátem pro ukládání digitálních audio a video streamů na mnoha video webových stránkách. Technické video parametry jsou znázorněny v tabulce 1.1 níže [1].

Tab. 1.1: Technické parametry formátu MP4 [1].

Typ	Hodnota
`Bitrate videa`	od 5 kbit/s až k více než 1 Gbit/s
`Skenování`	Progresivní i prokládané
`Podvzorkování barvonosných složek`	4:2:0, 4:2:2, 4:4:4
`Rozlišení`	od sub-QCIF až po subjektivní "Studio"

Algoritmus kódování videa použitý pro video založené na rámcích je založeno na osvědčené technologii ze standardů, jako je např. kompenzace pohybu na základě bloků DCT (Diskértní cosínova transformace), škálovatelnosti či odolnosti proti chybám [1].

Kompresní nástroje jsou definovány tak, aby zlepšily účinnost komprese oproti předchozím standardům MPEG-1 a MPEG-2. Zároveň se též klade důraz na podporu vysokého výkonu komprese pro všechny adresované bitové rychlosti, což za-

hrnuje kódování textur od přijatelné kvality pro vysoké kompresní poměry až po téměř bezeztrátovou kvalitu. Základní kompresní algoritmus je hybridní kódování, což si lze představit jako kombinaci pohybově kompenzované predikce a skalárně kvantovaného kódování DCT koeficientů [1].

Zvukové soubory, jenž jsou komprimovány algoritmem komprese AAC (Advanced Audio Coding), poskytují vynikající kompresi zvukových signálů při současném dosažení kvality. Ta je srovnatelná s kvalitou kompaktního disku (CD) při kódování 128 kb/s [3].

Formát souborů MP4 patří k možnostem formátu videa pro audio a video obsah. To je z velké části důsledkem kombinace vyjímečné kvality, malého rozměru, podpory pro více platforem a streamovací kompatibility [4].

1.2 HLS

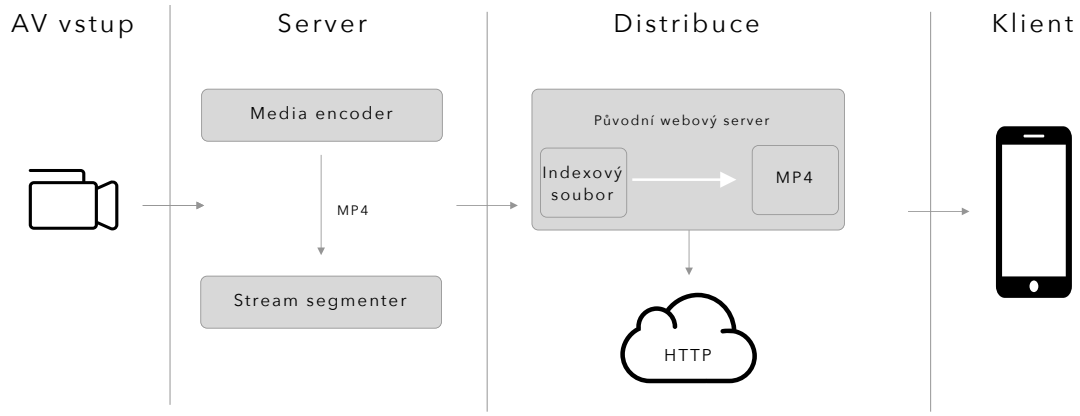
HLS je protokol, jehož zkratka znamená HTTP Live Streaming, se používá pro přenos zvuku a videa přes HTTP z běžného webového serveru pro přehrávání na zařízeních. Je navržen pro spolehlivost a dynamické přizpůsobování podmínkám sítě optimalizací přehrávání pro dostupnou rychlost kabelových a bezdrátových připojení [5].

1.2.1 Služby podporované protokolem HLS

- Živé vysílání,
- předem nahraný obsah (video na vyžádání (VOD)),
- vícenásobné střídavé streamy při různých bitových rychlostech,
- iteligentní přepínání streamů v závislosti na změnách šířky pásma sítě,
- šifrování médií,
- ověřování uživatelů [5].

1.2.2 Princip fungování HTTP Live Streaming

Protokol se skládá ze tří hlavních částí, jenž se nazývají: komponenty serveru, distribuční komponenty a klientský server. V klasické konfiguraci přebírá hardwarový snímač audio-video vstup, které kóduje video jako HEVC a audio jako AC-3. Následně vysílá fragmentovaný soubor MPEG-4 či přenosový tok MPEG-2. Segmenter poté stream přeruší do krátkých mediálních souborů, jenž jsou umístěné na webovém serveru. Mezi jeho další funkce patří vytváření a udržování indexového souboru obsahující seznam mediálních souborů [6].



Obr. 1.1: Princip fungování HLS [6].

Serverová komponenta

Je zodpovědná za přijímání a kódování vstupních dat médií a také připravuje zapouzdřené médium k distribuci. Server vyžaduje u živých událostí média kodér, který může být dostupný i hardwarově. Tento server lze také využít jako způsob, jak rozdělit zakódované médium do segmentů a uložit je jako soubory, které mohou být softwarem či součástí integrovaného řešení třetí strany [6].

Distribuční komponenta

Jedná se o webový server nebo systém pro ukládání do mezipaměti, který poskytuje klientům indexované soubory pomocí HTTP. Pro dodávání obsahu nejsou vyžadovány žádné vlastní moduly serveru a navíc je na webovém serveru obvykle potřebná lehká konfigurace. Pro nasazení protokolu HTTP Live Streaming je nutné vytvořit pro prohlížeče či aplikaci klienta HTML stránku, jenž bude sloužit jako přijímač. Dále je potřebné mít způsob, jak zakódovat živé streamy jako fragmentovaná média souboru typu MPEG-4. Ty obsahují videa HEVC či H.264 a audio AAC nebo AC-3 [6].

Klientský server

Je zodpovědný za určení vhodných médií, které si vyžádá. Tyto zdroje poté stáhne a následně sestaví tak, aby média mohla být uživateli prezentována v souvislém toku. Celkový proces začíná načtením indexového souboru pomocí adresy URL, jenž identifikuje stream. Indexový soubor naopak určuje umístění dostupných média souborů, dešifrovacích klíčů a dostupných alternativních toků. U vybraného streamu klient stahuje každý dostupný média soubor postupně, kde každý soubor obsahuje po

sobě následující segment streamu. Jakmile má dostatečné množství dat, klient začne prezentovat znovu sestavený stream uživateli. Klient je zodpovědný za získání jakýchkoli dešifrovacích klíčů a ověřování uživatelského rozhraní, které umožní autentizaci a dešifrování média souborů podle potřeby. Tento proces pokračuje, dokud klient nenajde značku *EXT-X-ENDLIST* v indexovém souboru. Pokud žádná taková značka není k dispozici, indexový soubor je součástí probíhajícího vysílání. Během probíhajících vysílání klient načte periodicky novou verzi indexového souboru. Klient vyhledá nové soubory médií a šifrovací klíče v aktualizovaném indexu a přidá tyto adresy URL do fronty [6].

1.2.3 Základy pro nasazení HTTP Live Streamu

- Stránka HTML pro prohlížeče nebo klientskou aplikaci sloužící jako přijímač.
- Webový server nebo CDN sloužící jako hostitel.
- Způsob kódování zdrojového materiálu nebo živých přenosů jako fragmentovaných mediálních souborů MPEG-4 obsahujících videoslužby HEVC či H.264, audio AAC nebo AC-3 (přestože pro video ve formátu H.264 lze používat zvukové soubory MP3 nebo přenosové toky MPEG-2, tyto formáty se nedoporučují) [8].

1.2.4 Klady a zápory protokolu HLS

+ Podpora na všechna zařízení.

  Původně byl tento protokol zaměřen pouze pro Safari a iOS zařízení. Nyní je však podporován všemi moderními webovými prohlížeči.

+ Výborná kvalita.

  Protokol využívá metodu pro měření rychlosti internetu, která se nazývá adaptivní přenos dat. Využití této metody je právě při zobrazení každého videa uživatelem, kdy se na základě rychlosti dynamicky upravuje kvalita videa.

+ Nízká cena.

  Díky podpoře na téměř všech zařízeních není potřeba používat konkrétní zařízení.

+ Bezpečnost.

  Ve srovnání s aplikací Flash poskytuje HLS divákům bezpečnější prohlížeč a to ne jen při sledování videa, ale i poté.

- HLS latence.

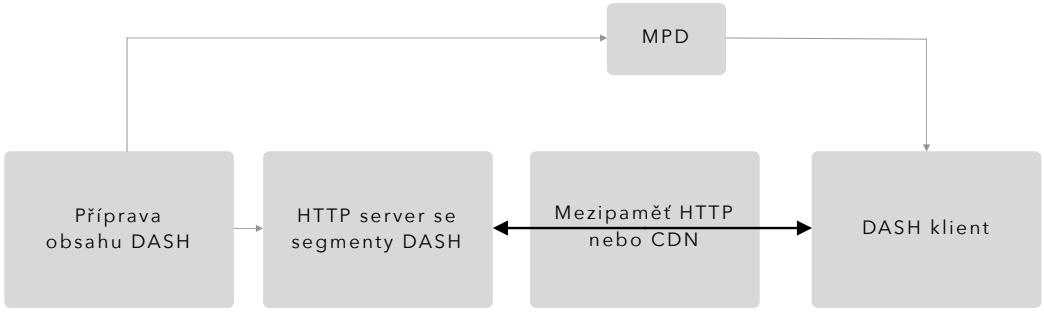
  Protokol byl navržen tak, aby maximalizoval kvalitu, ne minimalizoval absolutní latenci. Interval snímků, velikost paketu a požadavky na přehrávací

paměť nejsou vhodné pro rychlé přenosy živého vysílání. Proto zpravidla přidává do streamu zpoždění 20-60 sekund [7].

1.3 MPEG-DASH

MPEG Dynamic Adaptive over HTTP (MPEG-DASH) poskytuje standardní řešení pro efektivní a snadné streamování multimédií pomocí stávající infrastruktury HTTP. Umožňuje nasazení streamingových služeb pomocí nízkonákladové a rozšířené infrastruktury bez zvláštních ustanovení. Podporuje jak on-demand, tak živé vysílání. Zatímco lze DASH použít s libovolným mediálním formátem, má specifická ustanovení pro formát souborů MPEG-4 a MPEG-2 Transport Streams. Vzhledem k tomu, že relace streamování je řízena klientem DASH, server HTTP se nemusí potýkat s dodatečným zatížením řízení přizpůsobení streamu ve velkém měřítku [10].

Tento formát není systémy iOS podporován, proto mu byla věnována pouze okrajová pozornost [11].



Obr. 1.2: Princip fungování DASH [12].

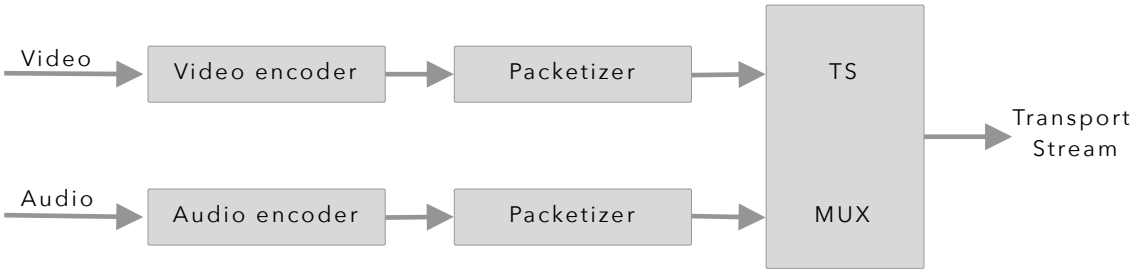
Obvykle klient DASH adaptivně požaduje malé kousky médií na základě dostupné šířky pásma a dalších zdrojů. Tato client-pull technologie pro zákazníky se ukázala být flexibilnější, firewall-přátelská a CDN-škálovatelná než technologie push-push [10].

1.4 MPEG Transport Stream (MPEG-TS)

Multimediální formát MPEG Transport Stream, který byl vytvořen tak, aby uspokojil potřeby spojené s vysíláním přes kabelové, pozemní a satelitní sítě. V praxi se tento standard používá především tam, kde není zaručena bezchybost přenosu dat.

Existují dvě hlavní varianty MPEG-2 TS. Pokud se do TS vkládá více programů, což jsou sady videí a dalších elementárních streamů, které jsou určeny ke společnému

přehrávání, nazývá se to MPTS nebo-li více programový přenosový proud. Toto je obecně používáno pro tradiční vysílání, jako je DVB (Digital Video Broadcasting) nebo HDTV. Pokud je do TS přidán pouze jeden program, považuje se to za SPTS nebo-li za jednotný programový přenosový proud. V případě SPTS dekodér obdrží pouze jeden stream (např. audio), což se například využívá v případě, když je program, který má být dekódován, předem znám (pro ušetření přenosové šířky pásma a zachování časových informací) [13].



Obr. 1.3: Princip standardu MPEG-TS [14].

1.5 DRM - Digital Rights Management

Každým dnem vzniká obrovské množství elektronických dokumentů, mezi které například patří filmy, firemní dokumenty, hry apod. Tato data je potřeba chránit, aby nedošlo k jejich zneužití vůči autorským právům. Cílem DRM je tedy šíření digitálního obsahu pod odchranou. Poskytuje funkce jako jsou autentizace, oprávnění, licencování, platby, řízení, ochrana soukromí a sledování porušování práva [16].

1.5.1 Některé funkce poskytované DRM:

- Autentizace,
- licencování,
- platby,
- ochrana soukromí,
- sledování porušování práva [16].

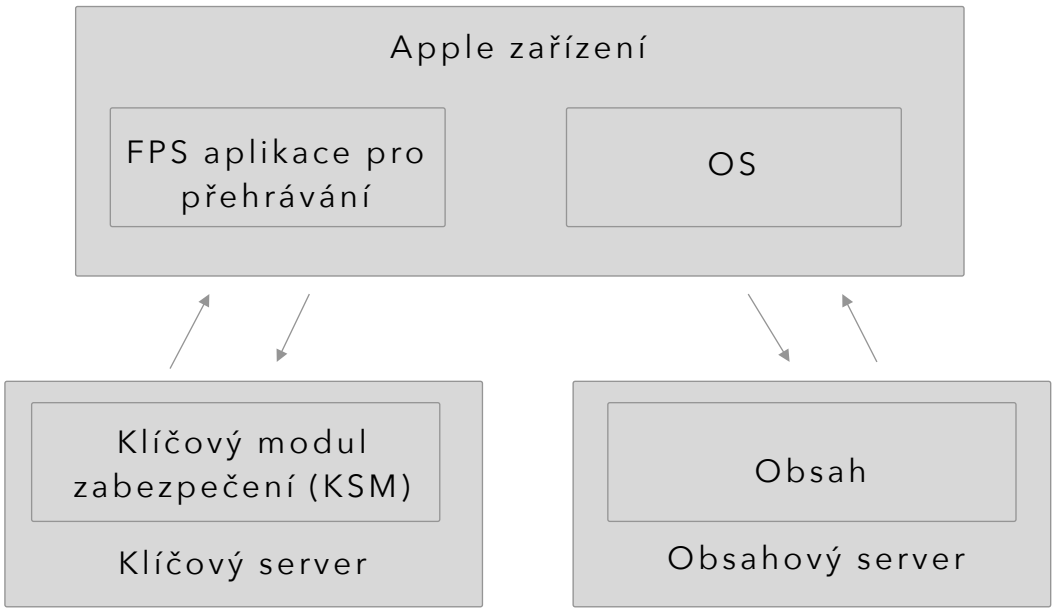
1.5.2 FairPlay

FairPlay je integrován do operačních systémů zařízení s nativní podporou *iOS* a *Apple TV*. Bezpečně dodává klíče, což umožňuje přehrávání šifrovaného video obsahu do mobilních zařízení *Apple*, *Apple TV* a na *OS X*. Obsah je zprostředkován přes

web pomocí protokolu HLS. Na server je odeslán konstantní identifikátor zařízení, jenž umožňuje serveru anonymní formou identifikovat zařízení [17].

Proces doručení klíče

Framework FPS inicializuje proces doručování klíčů a vytvoří relaci s klíčovým serverem poskytovatele obsahu. Následně klíčový server poskytovatele obalí 128-bitový klíč obsahu klíčem AES relace a mechanismem proti přehrávání. Proces doručení klíče implementuje řešení s trojitou ochranou, jenž se skládá z funkce AES, RSA a derivace. Poskytovatel obsahu šifruje videoobsah H.264 na základě jednotlivých snímků pomocí režimu AES-CBC, k čemuž využívá inicializační vektor a klíč obsahu. Na základě tohoto režimu poskytovatel také plně šifruje zvukový obsah. Mezi funkce FPS také patří podpora vytrvalosti bezpečnostního materiálu pro offline přehrávání, anonymní identifikaci zařízení vyžadující klíč obsahu či mechanismus zabezpečeného pronájmu (umožňující poskytovatelům spravovat obsah). Na obrázku 1.4 níže je znázorněn celý proces [17].



Obr. 1.4: Proces doručení klíče FPS [17].

1.5.3 PlayReady

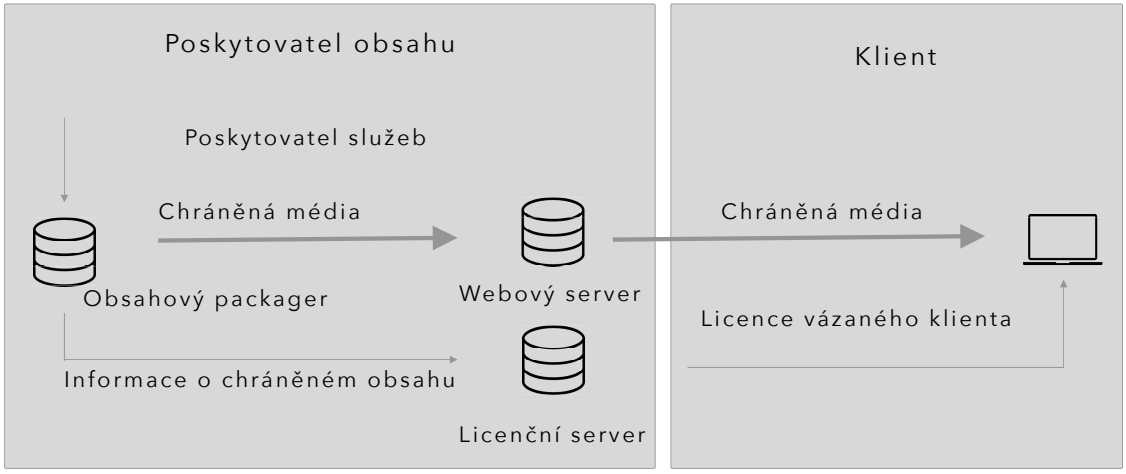
Další technologií umožňující zabezpečený přenos je PlayReady společnosti *Microsoft*, která slouží k ochraně obsahu a přístupu. Obecně to lze považovat za sadu technologií, která slouží k předcházení neoprávněného použití určitého obsahu. Dále se také používá k definování, zpracování a prosazování práv pro digitální média. V

rámci PlayReady může poskytovatel obsahu či služeb určit datum vypršení platnosti nebo dobu, po kterou může uživatel soubor přehrát. Využití této technologie je velice rozsáhlé - lze ji nalézt v mediálních aplikacích v televizorech, set-top boxech, mobilních telefonech či tabletech, osobních počítačích atd. [18].

Tato technologie není omezena pouze na zařízení se systémem *Windows*, implementovat ji je možné například i na operační systém *Android* [18].

Princip fungování

Mezi hlavní komponenty technologie PlayReady patří klient a server, které spolu komunikují pomocí protokolů specifikovaných společností Microsoft. Obsah je chráněn službou pro balení obsahu a poté je přenášen klientům, kteří dešifrují obsah pomocí informací uložených v licenci. Jako klienta lze považovat zařízení, které je schopné přehrávat chráněný obsah. Server může například být přizpůsobená aplikace, jenž umožňuje spolupráci s klienty. Servery zahrnují servery licenční, řadiče domén, servery se zabezpečeným zastavením či mazáním apod. Systém s technologií PlayReady šifruje obsah a ukládá jej na webový server, který klienti získávají prostřednictvím streamování či stahování. Klienti také získají licenci z licenčního serveru, který obsahuje informace potřebné k dešifování obsahu pro vykreslování. Celý proces je znázorněn na obrázku 1.5 níže [18].



Obr. 1.5: Proces fungování PlayReady [18].

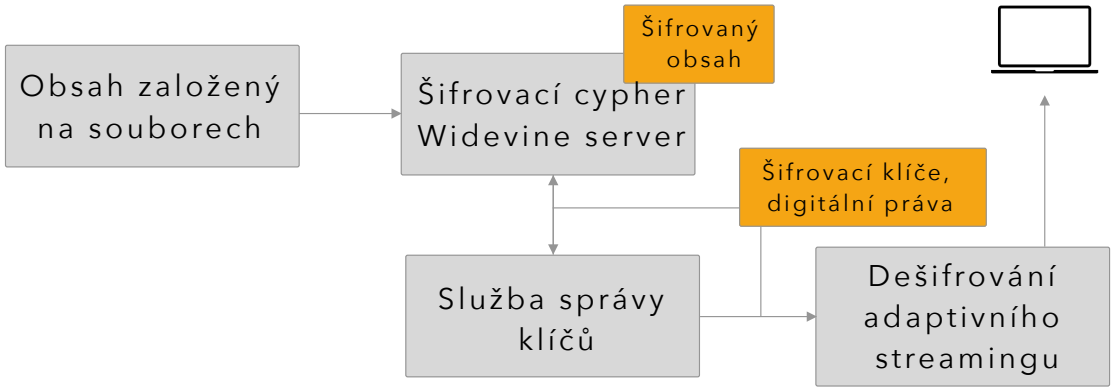
1.5.4 Widevine

Technologie Widevine, jenž patří společnosti *Google*, poskytuje bezpečné a chráněné přehrávání obsahu v podobě vybaveného *HTML5* přehrávače, mezi jehož funkce

například patří adaptivní streaming či *QoS* (Quality of Service). Důraz je kladen na zabezpečení zařízení, jenž je tvořeno pomocí továrních zabezpečených klíčů pro vytvoření kořenového adresáře pro bezpečné dešifrování a vykreslování obsahu [19].

Princip fungování

Widevine DRM nabízí vydavatelům kontrolu nad jejich šifrováním a správou klíčů. Na všechny příchozí prostředky registrované u Widevine uplatňují tvůrci obsahu (kteří také distribuují obsah) digitální práva a šifrování. Obsah je poté nahrán do cílové partnerské sítě, z kama poté Widevine přenáší soubory na všechna zařízení a pomocí formátu DASH zajišťuje jejich přehrávání. Na obrázku 1.6 níže je celý proces znázorněn [19].



Obr. 1.6: Proces fungování Widevine [19].

1.5.5 Výhody využití Widevine DRM:

- Podpora šifrování v každém zařízení,
- podpora kontejnerů (MP4, WebM),
- šifrování řešené hardwarem,
- framework, jenž podporuje současný i starší standard DRM,
- podpora HTML5 [19].

1.6 AES

Jedná se o standard (Advanced Encryption Standard), pro šifrování dat, jenž nahradil algoritmus DES (Data Encryption Standard), který se používal dříve. V principu se jedná o algoritmus používající sdílený klíč, který lze využít jak pro šifrování, tak i dešifrování dat. Velikost bloku je 128 bitů. Délka klíče však není pevně daná - může

nabývat hodnot 128, 192 či 256 bitů. Počet kol algoritmu AES na šifrování/dešifrování je závislý na délce klíče. Vztah mezi počtem těchto kol a délkou klíče je uveden v tabulce 1.2 [20].

Typ AES	Délka klíče	Velikost bloku	Počet kol
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

Tab. 1.2: Struktura algoritmu AES [20].

1.6.1 Popis šifrování

SubBytes - Každý bajt je nahrazen odpovídajícími bajty podle klíče. Tato záměna je způsobena díky nelineární funkci, jenž zde také funguje jako zábrana proti útokům.

ShiftRows - Přesunutí jednotlivých řádků podle určitých pravidel.

MixColumns - Operace pro smísení náhodných řádků v matici. Využivá se zde lineární transformace pro smísení každého ze čtyř bytů.

AddRoundKey - Každý byte v matici provádí operace pomocí rotačního klíče (subklíče), který je generován algoritmem. Následně je každý byte nahrazen odpovídajícími byty, čímž vznikne finální šifra [20].

1.6.2 Šifrování HLS AES

Pro tento standard jsou k dispozici dva typy šifrování - AES-128 a Sample-AES [21].

AES-128

Segment šifrován pomocí 128 bitového klíče [20].

Sample-AES

Jedná se o zašifrování jednotlivého vzorku média vlastním AES, kde šifrování závisí na formátu média. Umožňuje různé režimy šifrování, mezi které například patří šifrování jednoho z deseti vzorků [21].

1.7 Google Analytics

Jedná se o jednoduchý a snadno použitelný statistický nástroj, které pomáhá majitelům webových stránek či mobilních aplikací měřit celkovou aktivitu uživatele ve formě statistických dat. Mezi tato data například patří návštěvnost, čas strávený na jednotlivých prvcích aplikace či případy, kdy aplikace spadla. Princip měření takové aktivity funguje pomocí souborů HTTP cookies, jenž vývojáři poskytují informace o interakci uživatele s obsahem [22].

1.7.1 Hlavní komponenty

Statistický nástroj *Google Analytics* se skládá ze čtyř hlavních komponent: kolekce, konfigurace, zpracovávání a hlášení [22].

Obr. 1.7: Proces fungování Google Analytics [22].

Kolekce - shromažďování informací o interakcích uživatelů.
Konfigurace - umožňuje spravovat zpracování dat.
Zpracovávání - zpracování dat o interakci uživatele s konfiguračními údaji.
Hlášení - poskytuje přístup ke všem zpracovaným datům.

1.7.2 Google Analytics v mobilních aplikacích - iOS

Data o aplikacích jsou nativně integrovány do platformy společnosti Firebase (placené řešení), jenž poskytuje neomezené přehledy až pro 500 různých událostí. K měření aktivity uživatele v aplikaci tedy funguje ve spojení se sadou Firebase SDK, jenž automaticky zachycuje řadu událostí a uživatelských vlastností. Dále umožňuje definovat vlastní události a měřit věci, které mohou být pro učitého vlastníka aplikace

důležité. Může se například jednat o aktivní uživatele, demografické údaje, zakoupené položky apod. Jakmile jsou data zachycena, jsou ihned dostupná jak v rozhraní *Google Analytics*, tak i v konzoli *Firebase* [22].

Implementace

Implementace probíhá ve čtyřech základních krocích. Nejprve je potřeba propojit aplikaci s *Firebase* tak, že do nové či vytvořené aplikace se přidá *Firebase SDK* pro automatický sběr dat. Dále je možné připojit *Firebase* do *Google Analytics*, aby byla data viditelná v obou konzolích. V dalším kroku je možnost definovat publikum, které je pro vlastníka aplikace důležité. To lze například využít k cílení na zprávy, propagaci, reklamní produkty, notifikaci apod. [22].

1.8 VAST

V dnešní době je publikování obsahu online videí stále větší trend. Je tedy pochopitelné, že vydavatelé těchto videí se snaží svůj obsah zpeněžit pomocí in-stream reklam. Z tohoto důvodu vzniklo univerzální schéma XML, Video Ad Serving Template (VAST), jenž slouží k zobrazování reklamy pro přehrávače digitalních videí. Dále také popisuje očekáváné chování přehrávače při odpovědi na tyto reklamy. Před tímto schématem nebyl pro videopřehrávače žádný běžný reklamní protokol, což tedy umožnilo reklamním serverům distribuci reklam [23].

K tomu, aby bylo možné zobrazovat reklamy více vydavatelům používající nezávislé přehrávače, musely organizace poskytující reklamy vyvíjet odlišné odpovědi na reklamu u každého cíleného majitele stránek či videa. Tento přístup byl však drahý a ne snadno měřitelný. Jak již bylo řečeno, VAST, jenž byl představen na trhu společností IAB v roce 2008, poskytuje univerzální schéma, tedy protokol, který umožňuje reklamním serverům používat jeden formát odpovědi reklam ve více přehrávačích videí [23].

Dnes se však trh s digitální videoreklamou stává mnohem více sofistikovanějším. Pro zobrazení a zlepšení podpory reklam jsou vyžádovány stále nové funkce.

Technologie VAST není specifiká pro zařízení či platformu, a proto funguje v mnoha typech přehrávačů videa. Mezi tyto typy například patří:

- přehrávače na webových stránkách (optimalizované i pro mobilní zařízení),
- přehrávače v mobilních aplikacích,
- přehrávače v televizorech připojených k internetu,
- přehrávání videa přes IPTV či set-top-box [23].

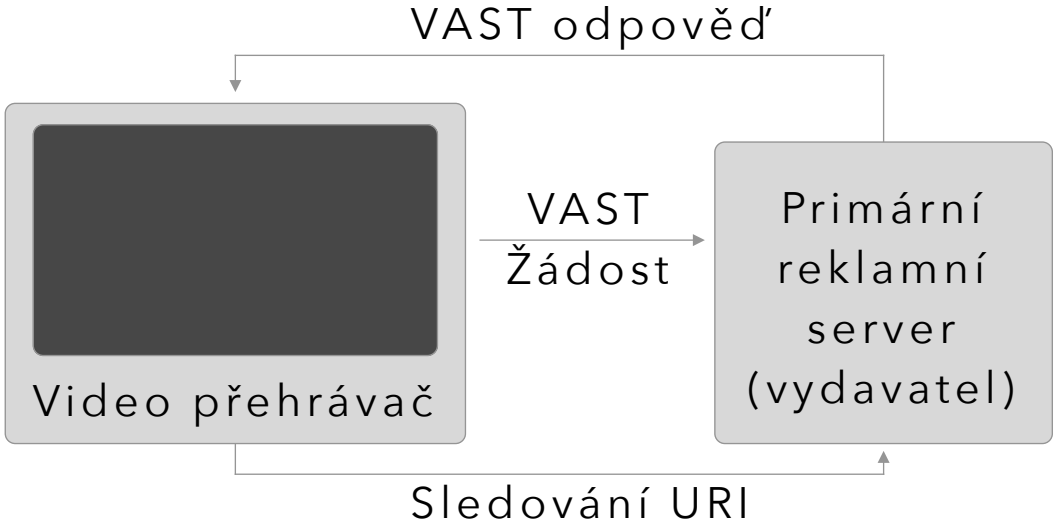
1.8.1 Princip fungování VAST

Proces zobrazování reklam, který je podporován technologií VAST, zahrnuje přehrávač videa požadující reklamu, odpověď VAST a zasílání informací o sledování zobrazené reklamy a dalších událostí zpět na server. Tato komunikace může být přímo provedena mezi videopřehrávačem a jedním či několika reklamními servery. Pro tento případ je proces zobrazování reklamy znázorněn níže na obrázku 1.8 [23].

VAST Request - Videopřehrávač se dotazuje serveru o VAST odpověď (VAST RESPONSE).

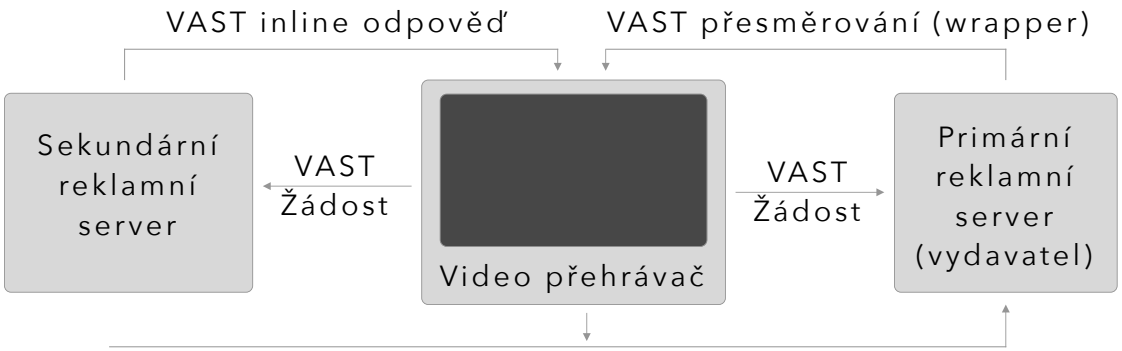
VAST Inline Response - Server posílá VAST Inline odpověď, jenž obsahuje veškeré multimediální soubory a sledovací URI potřebné k zobrazení reklamy.

Tracking URIs Pinged - Videopřehrávač požaduje ze sledovacích indetifkátorů URI sledování zdrojů v případě, kdy se v reklamě vyskytnou určité přidružené události [23].



Obr. 1.8: Princip fungování VAST [23].

Pokud se několik reklamních serverů stává součástí procesu zobrazování videoreklam, jsou výhody služby VAST ještě více přínosné. Tento proces je zobrazen na obrázku 1.9, kdy je zaimplementován i sekundární reklaní server.



Obr. 1.9: Princip fungování VAST se sekundárním reklamním serverem [23].

1.8.2 VAST 3.0

Poskytuje další funkce a lepší přehledy při zachování zpětné kompatibility s VAST 2.0, aby byl zajištěn bezproblémový přechod pro průmysl. S touto aktualizací přišla pro přehrávače možnost deklarovat, které formáty reklam budou podporovat [23].

Reklamní formáty:

- Lineární reklamy,
- nelineární reklamy,
- lineární reklamy, které jde přeskočit,
- lineární reklamy s doprovody,
- reklamní panely (sekvenční skupina reklam).

Určité přehrávače podporují pouze vybrané formáty reklamy VAST, jenž jsou v souladu s jejich obchodním modelem pro publikování. Vlastníci videopřehrávačů nemusí podporovat všech pět modelů, které jsou kompatibilní s technologií VAST 3.0. Libovolně mohou deklarovat podporu pouze pro jeden či více formátů [23].

Další novinky ve verzi VAST 3.0:

Podpora pro reklamní pody - Použitím atributu sekvence v prvku <Ad> lze naformátovat VAST odpověď, jenž seskupuje více reklam do jedné reklamní sekvence.

Podpora pro oznámení o ochraně osobních údajů - Videoklamy se může zůčastnit více reklamních serverů. V tomto případě je obtížné zobrazit oznámení o ochraně osobních údajů v reklamách. Novinkou ve VAST 3.0 je sdílení pokynů pro osvědčené postupy ke zpracování těchto údajů v reklamách.

Lepší hlášení chyb - S novou aktualizací také přichází zdokonalený seznam chybových kódů umožňující přehřavačům hlásit více konkrétních detailů [23].

1.8.3 VAST 4.0

Nová verze 4.0 služby VAST byla uvedena na trh 21. ledna v roce 2016. Toto vylepšení řeší plno nedostatků předchozích verzí a zároveň přináší další nové funkce [24].

Některé z nových funkcí:

Samostatný video a interaktivní soubor - V minulosti multimediální soubor VAST přijal celou řadu multimediálních souborů. Interaktivní rozhraní API však nelze vždy vykonat. Nově lze tedy oddělit videosoubor od rozhraní, což se může projevit v dobrém prospěchu mezi platformami a zařízeními.

Podpora ze strany serveru - Sledování a zobrazení reklam na straně klienta bylo doporučeným způsobem. In-stream reklamy se však často zobrazují na zařízeních, jenž nemohou sledovat reklamy pomocí tradičních způsobů zařízení, tedy klientů. VAST 4.0 podporuje stále častější metodu "adhesion", která umožňuje spojení lineárních videoreklam do streamu (videoobsah) a jeho odeslání klientům s omezenými možnostmi.

Soubor Mezzanine - Podpora reklamy na platformách, které zahrnují obsah s dlouhým obsahem či obrazovky s vysokým rozlišením. Tento soubor je však příliš velký, a proto jej nelze použít pro zobrazení reklam. Své uplatnění má spíše u služeb pro vytváření reklam a další prodejce, kde je využíván pro vytvoření souborů s odpovídající úrovní kvality pro prostředí, ve kterém hrají.

Univerzální ID reklamy - Nová funkce, jenž se používá pro zahrnutí identifikátoru, který je udržován v různých systémech.

Standardizovaný timestamp - V nové verzi byl standardizovaný nový formát tak, aby umožňoval konzistentnější měření citlivé na čas [24].

1.9 VPAID

V současné době mohou být videoreklamy vytvořené pomocí konkrétních reklamních platforem, jenž jsou spouštěny pouze u video přehravačů, které jsou předem integrované do stejné platformy. Tento problém se může komplikovat například v případě, kdy je reklama vytvořena v inovativních formátech, které vyžadují vysokou úroveň komunikace mezi reklamou a videopřehrávačem. Mezi tyto formáty kupříkladu patří nelinearní a interaktivní reklamy [25].

K řešení takových problémů s interoperabilitou se používá norma VPAID (Video Player-AD API definition), která standardizuje komunikaci mezi přehrávači videí a in-stream reklamou. Protože tento standard nevyřeší celou komunikaci sám, musí majitelé reklamních stránek, chtějící takové reklamy, přijmout i následující normy:

Normy

- Digital Video Measurement Guidelines,
- Digital Video Ad Format Guidelines and Best Practices,
- Digital Video In-Stream Ad Metrics Definitions,
- Digital Video Ad Serving Template (VAST).

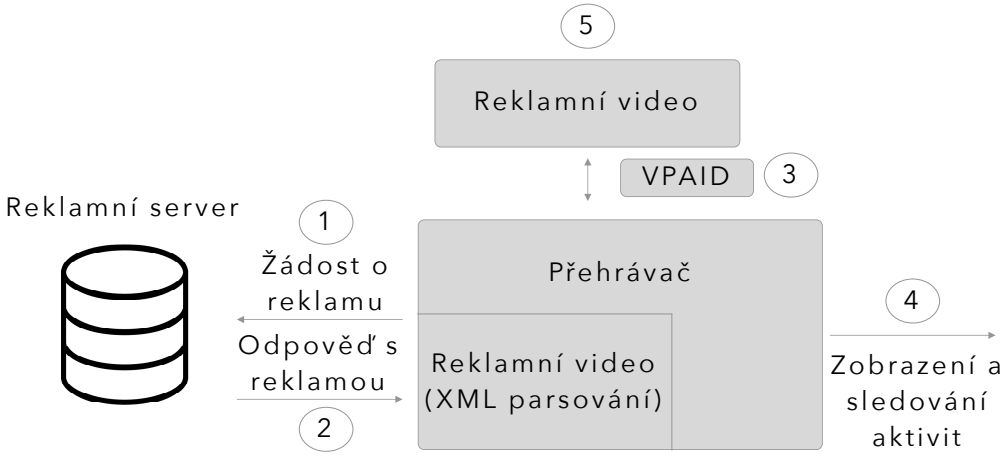
Kdokoliv, kdo přijme výše uvedené standardy (včetně VPAID), bude schopen vysílat jakýkoliv druh videoreklamy z jakéhokoli serveru s videoreklamami, jenž bude dodržovat stejné formáty [25].

1.9.1 Princip fungování

Na rozdíl od bannerových reklam se reklamy in-stream provádějí přímo v prostředí přehrávače. Ten je dále zodpovědný za interpretaci odpovědi reklamy na reklamním serveru a také za provedení reklamního hovoru. Samotná reklama může spouštět skripty a variabilitu má založenou na interakci uživatele. Pro umožnění interoperability pokročilých videoreklam v různých prostředích videopřehrávače je vyžadována minimální standardizace v úrovních zmíněných níže:

- Standard VAST - odpověď reklamního serveru by měla být pro přehrávač, které je v souldu s touto normou, srozumitelná .
- Standard VPAID - pokročilá reklama vyžaduje standardní dobu přehrávače, která je vyjádřena jako standardní rozhraní API.

Princip fungování standardu VPAID, které znárorňuje obecný tok videoreklamy a bod rozhraní, je zobrazeno na obrázku 1.10 níže [25].



Obr. 1.10: Princip fungování VPAID [25].

1 - Přehrávač provede volání na reklamní server.
2 - Reklamní server zareaguje v podobě XML reklam.

3 - Přehrávač načte reklamní média, jenž byla uvedena v XML reklamě, a vykreslí je (interakce přehrávače a reklamy).

4 - Zobrazení a sledování aktivit.

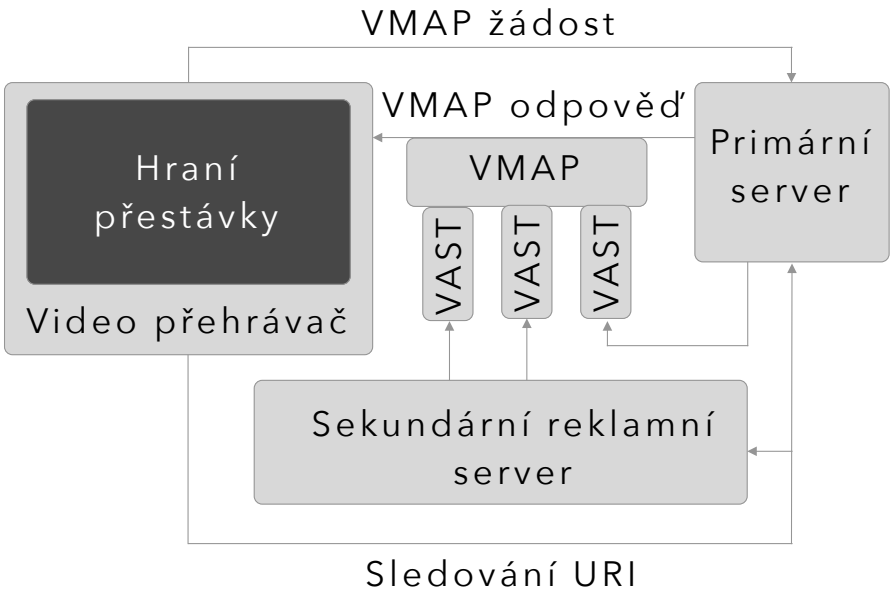
5 - Pravidla pro vykreslování reklamy [25].

VPAID je nutné a zároveň nejužitečnější pro ty reklamy, které se zobrazují současně s videoobsahem. To například mohou být překryvné reklamy či reklamy, jenž musí v důsledku interakcí uživatele s reklamou přerušit přehrávání videoobsahu [25].

1.10 Video Multiple Ad Playlist (VMAP)

Pro efektivní zpeněžení video obsahu s vkládáním reklamy do streamu musí majitelé video obsluhy pečlivě spravovat strukturu. Když vlastník obsahu ovládá distribuci obsahu, může tento vlastník lehce spravovat umístění reklamy v hrajícím obsahu. V případě, kdy ale video obsah vysílaný v přehrávači neovládá, správa pro umístění reklamy do videa se stává komplikovaná. V takových případech je možné využít standard VMAP, jehož pomocí mohou majitelé videoobsluhy ovládat reklamní inventář zobrazený v obsahu videa. Služba dále umožňuje vlastníkovi obsahu definovat přestávky pro reklamy, včetně časového rozvrhu každé přestávky, kolik reklam je k dispozici, typ reklamy apod. Jednodušeji řečeno lze tento standard popsat jako šablonu v jazyce XML, kterou majitelé video obsahu mohou použít k popisu struktury pro vkládání reklamního inventáře, kdy nekontrolují přehrávač videí [26].

Na obrázku 1.11 níže je znázorněn zjednodušený proces VMAP.



Obr. 1.11: Proces VMAP [26].

VMAP požadavek - Video přehrávač požaduje VMAP odpověď z primárního serveru.

VMAP odpověď - Primární server vrátí VMAP odpověď, jenž obsahuje seznam reklamních přestávek. Každá tato přestávka odkazuje na odpověď reklamy VAST, která přehrávači videí poskytuje reklamy k vyplnění konkrétních reklamních přestávek.

Přehrávání přestávky - Videopřehrávač spouští reklamy v souladu se standardem VAST.

Sledování URI indentifikátorů - V příslušných časech posílá přehrávač požadavek na sledovací identifikátory URI, jenž jsou uvedené v normě VMAP. Pod tímto identifikátorem si lze například představit začátek a konec reklamy [26].

Co je důležité mít na paměti:

VMAP neslouží jako nahráda standardu VAST - Je pouze doplňkem, jenž pomáhá vyřešit případ užití. VAST poskytuje reklamy a VMAP představu seznamu těchto reklam.

VMAP nepatří mezi nutnou implementaci - Pro splnění standardu VAST není nutné také implementovat normu VMAP (funguje i normy).

VMAP není pro každého - Technologie není určena pro použití všemi prvky v rámci reklamy. Určení je spíše pro reklamní server, který je řízen stranou s primárním právem spravovat inventář videoreklamy. Obvykle je na mysli strana, jenž vytváří samotné video.

VMAP není přehrávačem video odpovědí - Obecně lze říci, že se jedná o způsob přenosu informací z reklamního serveru na videopřehrávač. Zachování správné funkcionality je tedy na straně přehrávače [26].

2 Dostupná řešení

Pro lepší chápání při implementování jednotlivých funkcí přehrávače byla provedena analýza dostupných řešení, které jsou k dispozici. Existuje již velké množství firem a projektů, jenž se snaží vyvinout videopřehrávač pro danou platformu co nejlépe. Níže jsou popsány tři taková řešení (Kaltura, BitMovin a Brightcove), která na trhu patří mezi nejpoužívanější. Závěr této kapitoly je ukončen srovnáním těchto přehravačů a popisu systémové knihovny AVFoundation.

2.1 Kaltura

Jedná se o multi-platformní HTML5/Flash videopřehrávač, jenž umožňuje jednoduché přizpůsobování uživatelského rozhraní. Kaltura podporuje přehrávání zabezpečeného obsahu pomocí Widevine DRM, ale již je plánována podpora i pro PlayReady DRM. Dále také podporuje různé druhy reklam, jako jsou například VAST 3.0, Google DoubleClick DFP, FreeWheel, apod. Pro analýzu přehrávače je možné využít analytických nástrojů, jako jsou například Google Analytics, Nielsen Video Census a Omniture SiteCatalyst 15. Důležité je také poznamenat, že se jedná o placený přehrávač, mezi jehož klienty patří Philips, HBO, Stanford či Warner Bros [27].

2.2 BitMovin

BitMovin je též multi-platformí HTML5 přehrávač, jenž mimo *iOS* a *Android* podporuje také širokou škálu televizních systémů (*Android TV*, *Apple TV*, *Samsung TV*, *LG TV*,...) a Roku. Mezi podporu přehrávání zabezpečeného obsahu DRM patří například *Irdeto*, *ExpressPlay*, *BuyDRM*, *EZDRM*, *Conax*, *FairPlay*, *PrimeTime*, *Widevine* a *PlayReady*. Na platformě *iOS* je podpora pouze pro *FairPlay* a *HLS-AES*. Výhodou tohoto přehrávače je, že dokáže přehrát obsah DRM i offline či LiveStream. Pro zobrazení reklam by měla být umožněna spolupráce s jakýmkoliv reklamním serverem (VAST, VPAID, IMA a VMAP). Společnost má také vlastní řešení pro rychlé dešifrování videa pomocí cloudu, jenž se vztahuje na formáty MPEG-DASH a HLS. U platformy *iOS* není MPEG-DASH podporován, neboť tento formát není u systémů firmy Apple celkově podporován. BitMovin přehrávač je též, jako Kaltura, placený. Mezi partnery této firmy například patří *Ooyala*, *Microsoft*, *Amazon Web Services* a *Google Cloud Platform* [28].

2.3 Brightcove

Stejně jako předchozí zmíněná řešení je i tento přehrávač firmy *Brightcove* vyvíjen pomocí HTML5. Výrobce navíc uvádí, že jejich přehrávač, který navíc podporuje širokou škálu modulů, je až o 70 procent rychejší než jakýkoliv jiný přehrávač. S podporou přehrávání zabezpečeného obsahu DRM lze například využít *FairPlay*, *PlayReady* a *Widevine*. *Brightcove* přehrávač také podporuje zobrazení VAST reklam ve videích, přičemž nabízí i vlastní služby. Využití tohoto přehrávače je také zpoplatněno, jehož cena závisí na rozsáhlosti využití. Většina funkcí pro systém *iOS* není psáno v jazyce *Swift*, ale *Objective-C* [29].

2.4 Srovnání dostupných řešení

V tabulce 2.1 níže je znázorněno srovnání zmíněných řešení. Přehrávače mají tatřka identické funkce, přičemž všechna realizují přehrávání pomocí *HTML5*. Jejich hlavní rozdíl spočívá v podpoře přehrávání zabezpečeného obsahu pomocí DRM a také ceně odvíjející se dle licence. Příklady podporovaných DRM jsou již zmíněny v popisu jednotlivých řešení, přičemž konkrétní podpora u přehrávače *Brightcove* nebyla možná zcela dohledat.

Přehrávač	**Podpora DRM**	**VAST**	**Analýza**	**Řešení přehrávače**	**Zdarma**
Kaltura	Widevine	Ano	Ano	HTML5	Ne
BitMovin	Irdeto, ExpressPlay, BuyDRM, EZDRM, Conax, FairPlay, PrimeTime, Widevine, PlayReady	Ano	Ano (vlastní řešení)	HTML5	Ne
Brightcove	Ano (druhy neznámé)	Ano	Ano (vlastní řešení)	HTML5	Ne

Tab. 2.1: Srovnání řešení.

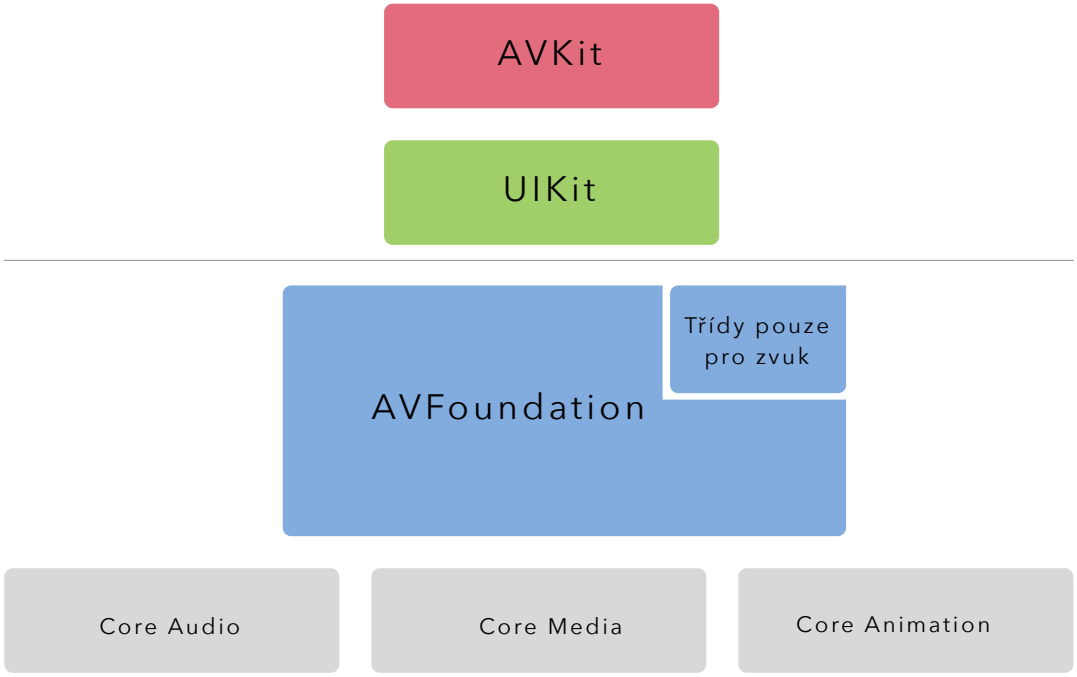
2.5 AVFoundation

Jedná se systémový framework firmy Apple, jenž slouží pro práci s audiovizuálními médii v *Apple* systémech (*iOS*, *macOS*, *watchOS* a *tvOS*). Na rozdíl od výše zmíněních řešení se jedná o alternativu, která není postavena na webové technologii. Pomocí této knihovny je možné snadné přehrávání a úprava souborů MPEG4 a HLS. Práce s multimediálním obsahem však není jediná funkce, neboť se tato knihovna také používá k pořizování fotografií, videa či zvukové stopy. Primární třídu, kterou *AVFoundation* používá k reprezentaci médií, je *AVAsset*. Tuto instanci lze popsat jako reprezentaci sbírky jedné nebo více zvukových či obrazových stop, jenž poskytuje informace o sbírce jako celku (název, doba trvání, velikost atd.).

AVFoundation umožňuje vytvářet nové reprezentace instance *AVAsset* několika způsoby. Jednoduše lze znovu zakódovat existující instanci (podpora od *iOS* 4.1) či provést určité operace s obsahem daného materiálu a uložit výsledek jako novou instanci.

Knihovna dále umožňuje sofistikované přehrávání materiálu, což například umožňuje přehrávání dvou různých segmentů stejné instance, které jsou současně vykresleny v různých rozlišeních [30].

2.5.1 Architektura knihovny AVFoundation



Obr. 2.1: Architektura *AVFoundation* [30].

AVKit - Framework sloužící pro výchozí přehrávání videa.

UIKit - Pomocí knihovny *AVFoundation* je také možné zaznamenávat video. Framework UIKit lze použít například v případě, kdy nad takovým záznamem stačí minimální kontrola.

Core knihovny - Deklarace datových struktur, které jsou použity v knihovně AVFoundation. Zahrnuty jsou také datové struktury související s časem a neprůhlednými objekty, jenž popisují mediální data.

2.5.2 AVAsset

AVAsset definuje kolektivní vlastnosti skladeb, jenž obsahují daný materiál a lze jej vytvořit inicializací pomocí místní nebo vzdálené adresy URL. Vytvoření takového objektu je znázorněno ve výpise 2.1 níže [31].

Výpis 2.1: Inicializace objektu AVURLAsset.

```
let asset = AVAsset(url: url)
```

Ve skutečnosti je však *AVAsset* abstraktní třídou. To znamená, že pokud se asset vytvoří znázorněným způsobem, vznikne instance jedné ze svých konkrétních podtříd nazývaných *AVURLAsset*.

Kromě tohoto způsobu je také možné přímé vytvoření instance *AVURLAsset*. Využití přímého vytvoření je například v případě, kdy je potřebné mít větší kontrolu nad inicializací aktiva. Takový inicializátor poté přijímá slovník možností, pomocí něhož lze přizpůsobit inicializaci daného případu. Příklad této inicializace je znázorněn ve výpisu 2.2 níže [31].

Výpis 2.2: Inicializace objektu AVURLAsset.

```
let asset = AVURLAsset(url: url, options: options)
```

2.5.3 Reprezentace médií

Data videa a související metadata jsou zastoupeny v *AVFoundation* neprůhlednými objekty z knihovny *Core Media*. Tato data reprezentuje pomocí *CMSampleBuffer*, což si lze představit jako neprůhlednou instanci typu *Core Foundation*. Zmíněná instance obsahuje vyrovnávací paměť pro rámec dat videa jako *CVPixelBuffer* (vyrovnávací paměť pixelů) [32].

3 Návrh přehrávače

3.1 Navrhované řešení

Protože nebylo nalezeno žádné vhodné open-source řešení, bude vývoj přehrávače probíhat zcela od začátku. Oproti zmíněným dostupným řešením je tedy v plánu vyvinout přehrávač, jenž bude napsán pouze v programovacím jazyce Swift. Přehrávání videa tedy nebude řešeno pomocí technologie HTML5/Flash, ale bude použit systémový framework *AVFoundation*, jenž slouží k přehrávání multimediálního obsahu. Přehrávač s implementací této knihovny poskytuje pouze základní funkce (play, pause, next, previous, slidebar, titulky) k ovládání přehrávače s neměnitelným uživatelským rozhraním.

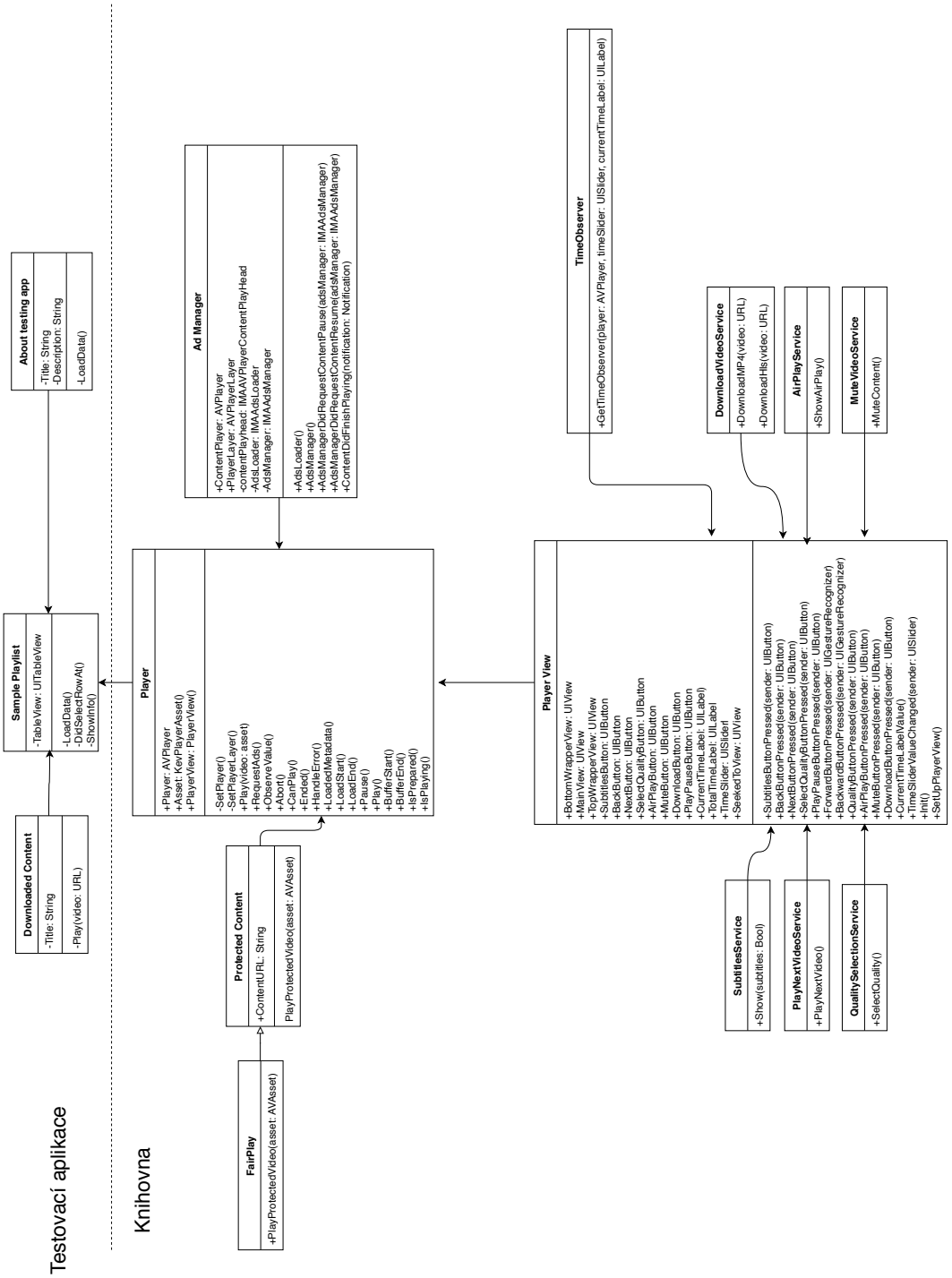
Navrhnuté řešení tedy bude ze zmíněné knihovny používat pouze vrstvu pro přehrávání videa, přičemž rozhraní ovládacího panelu a další funkce rozšiřující tento přehrávač budou tvořeny samostatně. Podporované formáty budou HLS a MP4. Předpokládala se i podpora pro MPEG-DASH, ale ten systémem iOS není podporován.

Mezi plánované funkce také bude patřit implementace přehrávání zabezpečeného obsahu pomocí DRM (FairPlay) a zobrazování reklam (VAST, VMAP, VPAID).

Pro sbírání dat o chování uživatele při přehrávání knihovny bude implementován analytický nástroj Google Analytics.

3.2 UML diagram knihovny

Na následující stránce je znázorněn UML diagram připravované knihovny pro zařízení se systémem iOS, ke kterému je popsána i funkcionalita jednotlivých tříd.



Obr. 3.1: Diagram přehrávače.

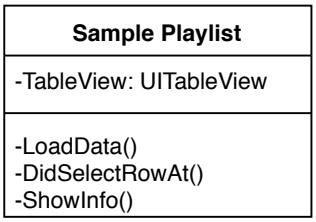
3.2.1 Popis jednotlivých komponent diagramu

3.2.2 SamplePlaylist

SamplePlaylist představuje jednoduchou aplikaci, ve které bude implementován vyvíjený framework. Aplikace bude mít čtyři obrazovky, přičemž jedna bude obsahovat seznam videí, druhá představovat seznam stažených videí, třetí informace o vyvíjené aplikaci a čtvrtá samotné zobrazení přehrávače.

Popis metod

- *LoadData()* - Slouží k načtení seznamu videí do tabulky.
- *DidSelectRowAt()* - Při stisknutí určitého řádku v tabulce dojde k volání na přehrávač a následně k zobrazení požadovaného obsahu.
- *ShowInfo()* - Tato metoda bude použita pro zobrazení informací aplikace.



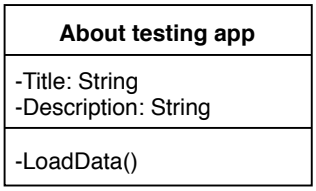
Obr. 3.2: Třída charakterizující testovací aplikaci.

3.2.3 About app

Tato třída slouží pouze k zobrazení informací. Volána bude přes třídu *SamplePlaylist*.

Popis metod

- *LoadData()* - Slouží pouze k inicializaci dat.



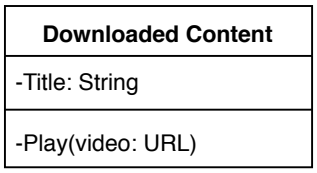
Obr. 3.3: Třída About app.

3.2.4 Downloaded content

Třída Downloaded content zobrazuje seznam stažených videí.

Popis metod

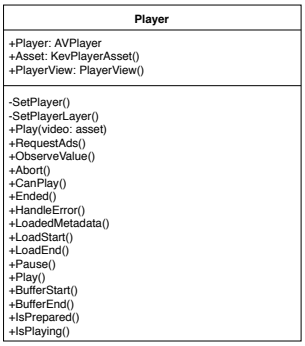
- *Play(video: URL)* - Metoda, jenž se zavolá po stisku na určité stažené video, zobrazí přehrávač s požadovaným obsahem.



Obr. 3.4: Třída Downloaded content.

3.2.5 Player

Player je hlavní komponenta, jenž do sebe implementuje všechny ostatní potřebné funkce. Nedochází zde tedy jen k inicializaci obsahu, ale také reklam. Dále třída obsahuje funkce pro analýzu knihovny. Důležitý je zde objekt *asset* typu *KevPlayerAsset*, který přehrávači poskytuje všechny potřebné parametry pro inicializaci. Zmíněný typ bude více popsán v sekci Přehrávač. Třída je znázorněna na obrázku 3.5.



Obr. 3.5: Třída Player.

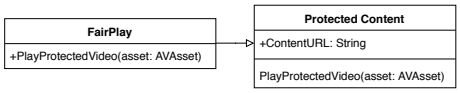
Popis metod

- *Init()* - Metoda sloužící k inicializaci přehrávače.

- *SetPlayerLayer()* - Jedná se o funkci, jenž do již přehrávaného videa přidává další vrstvu. Ta je například potřebná pro zobrazení titulek, reklamy, apod.
- *Play()* - Funkce starající se o spuštění videa.
- Ostatní metody jsou tzv. observery. Jejich využití je v konkrétním případě vhodné pro kontrolování stavů videa. Dle názvů jednotlivých observerů by mělo být jisté, jaká je jejich funkce. Díky těmto observerům je např. možné zachytit, kdy bylo přehrávání stopnuto, došlo ke změně kvality obsahu apod.

3.2.6 Protected Content (FairPlay)

Pokud bude do přehrávače poslán zabezpečený obsah, bude zavolána třída Protected Content, která se již postará o správné přehrání videa, výměnu klíčů apod. Z této třídy dále dědí třída FairPlay.



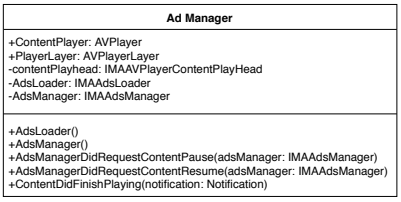
Obr. 3.6: Třída ProtectedContent.

3.2.7 AdManager

Třída AdManager slouží k zobrazení reklam. Zde bude implementována knihovna *IMA* od firmy *Google*, která se stará o zpracování reklam.

Popis metod

- *AdsLoader()* - Stará se vykreslení reklamy.
- *AdsManager()* - Metoda sloužící ke komunikaci mezi reklamním serverem a klientem.
- *RequestAds()* - Funkce, jenž provádí volání na server a žádá o reklamy.
- Ostatní metody jsou opět observery sledující stav reklamy.



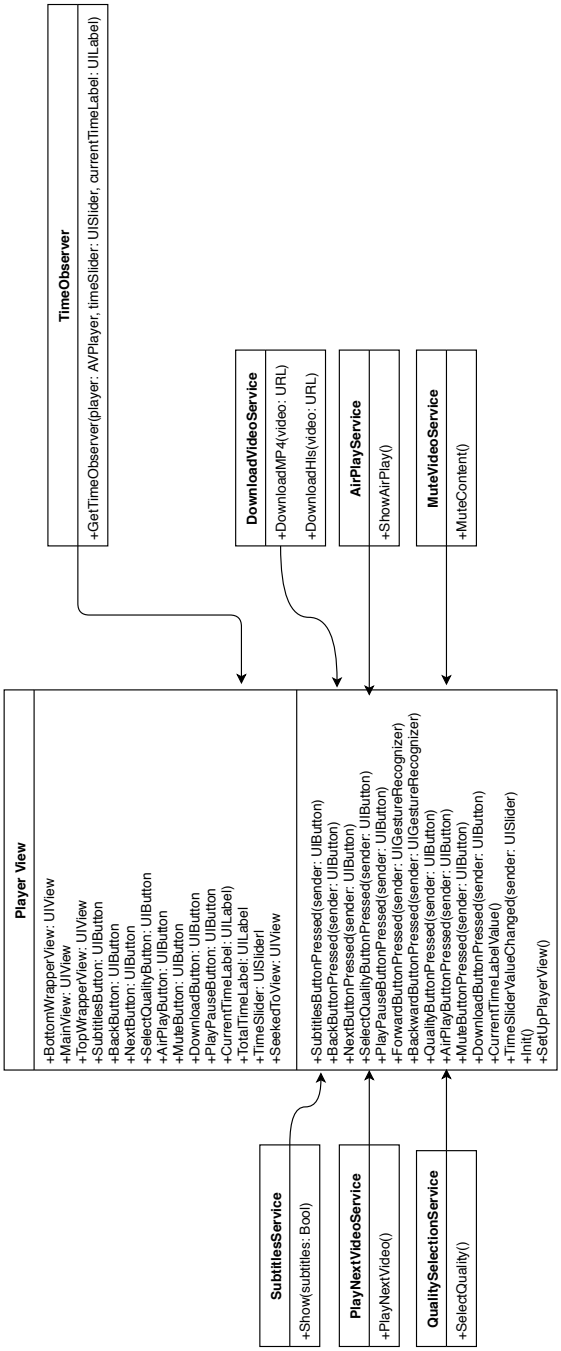
Obr. 3.7: Třída AdManager.

3.2.8 PlayerView

Při správném nastavení přehrávače dojde k zobrazení rozhraní přehrávače. Rozhraní PlayerView obsahuje potřebná tlačítka a funkce pro ovládání a pozorování obsahu.

Popis metod

- *SubtitlesButtonPressed()* - Metoda provádějící volání na metodu *Show(subtitles: Bool)* z pomocné třídy *SubtitlesService*. Na základě parametru *subtitles* skryje či naopak zobrazí titulky.
- *BackButtonPressed()* - Tlačítko, které po stisknutí přestane přehrávat aktuální obsah a nahradí jej jiným. Opět zde využívá pomocnou třídu, jenž se pro tuto funkci nazývá *PlayNextVideoService*. Zmíněnou pomocnou třídu využívá také metoda *NextButtonPressed*.
- *SelectQualityButtonPressed()* - Metoda zobrazující možnosti výběru pro změnu kvality přehráváného obsahu. Pomocná třída je zde *QualitySelectionService*.
- *PlayPauseButtonPressed()* - Spustí či naopak pozastaví přehráváný obsah.
- *ForwardButtonPressed()* - Funkce, která posune přehráváný obsah na požadovanou časovou hodnotu dopředu. Pro přetočení dozadu je zde metoda *BackwardButtonPressed()*.
- *AirPlayButtonPressed()* - Metoda zobrazující nabídku AirPlay pomocí pomocné třídy *AirPlayService*.
- *MuteButtonPressed()* - Slouží pro instantní ztišení přehrávaného obsahu.
- *DownloadButtonPressed()* - Využívá pomocné třídy *DownloadVideoService*, jenž na základě formátu přehrávaného videa začně požadovaný obsah stahovat pro offline přehrávání.
- *CurrentTimeLabelValue()* - Zobrazuje aktuální časovou hodnotu přehrávaného obsahu.
- *TimeSliderValueChanged()* - Indikuje posun na posuvníku a tím přetáčí obsah na požadovanou hodnotu.
- *Init()* - Slouží pro inicializaci ovládacího panelu přehrávače.
- *SetUpPlayerView()* - Nastavuje rozvržení všech objektů a také jejich vzhled.



Obr. 3.8: Třída PlayerView.

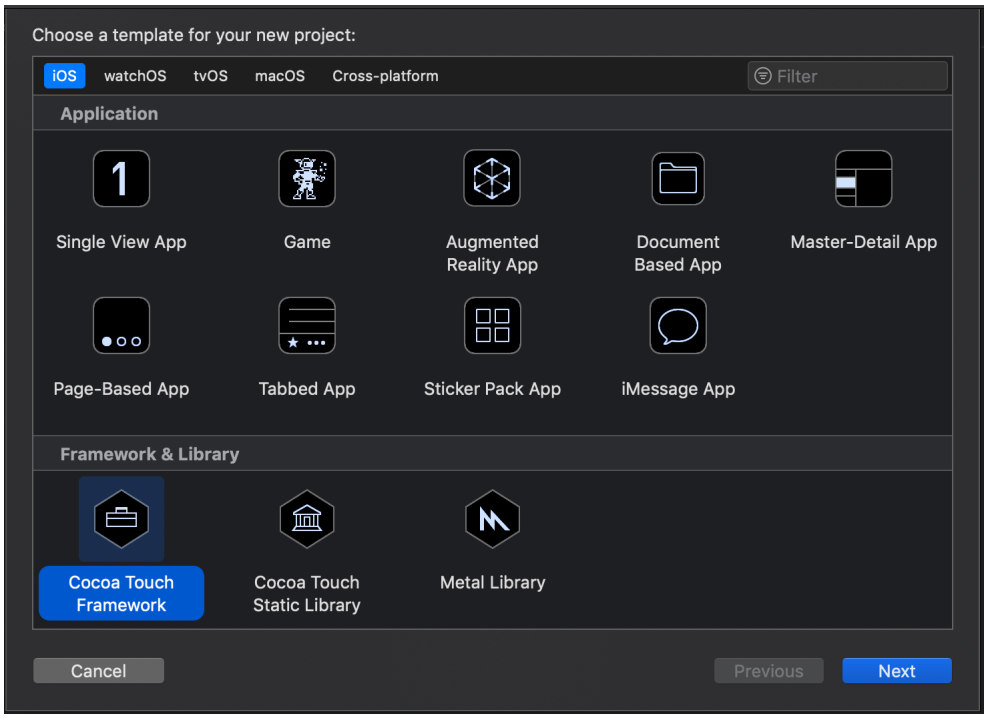
4 Vývoj knihovny

Následující kapitola je věnována vývoji knihovny. Formáty jednotlivých ukázek a výpisů kódu jsou zde upraveny, aby je bylo možné vložit do tohoto dokumentu. Reálné formátování a celý kód je možné zhlédnout na přiloženém disku. Jak již bylo zmíněno v sekci Návrh přehrávače, pro vývoj knihovny je využit framework *AVFoundation*. Z dalších systémových knihoven byla například použita i *AVKit*. Samotný přehrávač je založen na systémovém objektu, jenž se nazývá *AVPlayer*.

4.1 Vytvoření a nastavení projektu

Před začátkem vývoje je nejprve nutné pro tuto knihovnu založit projekt. K jeho vytvoření poslouží vývojářský program XCode, kde jej lze založit pomocí třech kroků.

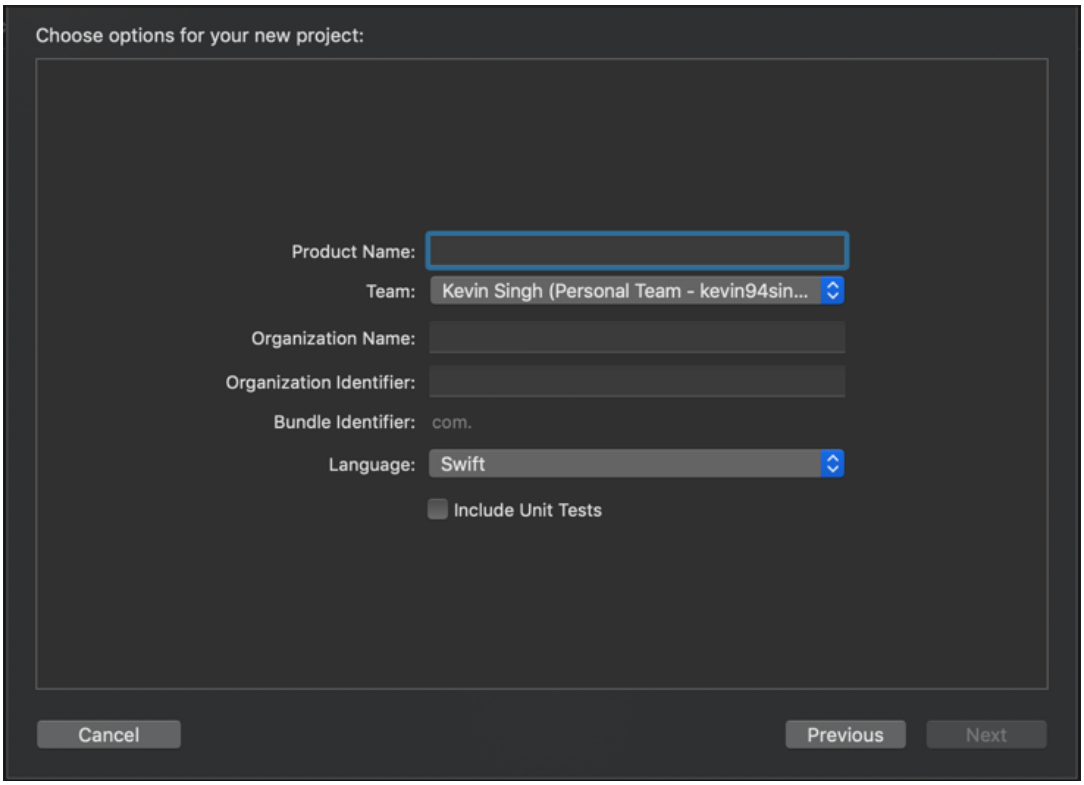
1 V programu Xcode je prvně nutné zvolit, jaký typ projektu je třeba vytvořit. V případě tvorby knihovny je potřebné zvolit *Cocoa Touch Framework*.



Obr. 4.1: Výběr typu projektu.

2 V druhém kroku je nutné vyplnit základní údaje o projektu. Nejprve je požadováno zvolit vhodné jméno, aby bylo pro daný projekt výstižné. Dále lze zvolit účet, pod kterým bude projekt vyvíjen. Organizační identifikátor je defaultně vytvořen, lze jej však změnit. Zde je důležité poznamenat, že jeho výběr ovlivní

balíčkový identifikátor (Bundle identifier), který slouží k identifikaci aplikace ve virtuálním obchodě *App Store*. Poslední krok slouží ke zvolení programovacího jazyka, který je pro projekt použit. Na výběr je zde Swift a Objective-C. V tomto kroku lze také zvolit, zda má projekt zahrnovat unit testy. V případě, že se vývojář rozhodně testy nezahrnovat, může je v budoucnu přidat.



Obr. 4.2: Doplnění informací o projektu.

3 Poté již stačí zvolit úložiště a adresář, kam se vytvořený projekt uloží. Tím je projekt vytvořen a lze přejít na samotný vývoj.

4.2 CocoaPods

Pro využívání funkcí potřebných knihoven třetích stran je nejprve tyto frameworky nějakým vhodným způsobem nutné implementovat do vyvíjené knihovny. Způsobů je několik, pro tento projekt je zvolena cesta s využitím *CocoaPods*. Ty si lze představit jako manažera závislostí, který spravuje jednotlivé knihovny jako pody. Jedním příkazem je například možné aktualizovat všechny obsažené pody či s nimi provádět další operace.

Implementace

Při zcela prvním použití CocoaPods je nejprve nutné nainstalovat takzvaný gem přes terminál, jako je znázorněno ve výpisu 4.1 níže.

Výpis 4.1: Instalace gemu.

```
$ sudo gem install cocoapods
```

V dalším kroku je potřebné se dostat v terminálu do složky, kde je umístěna vyvinutá knihovna. Zde poté následujícím příkazem, jenž je zobrazen ve výpisu 4.2, dojde k inicializaci souboru tzv. *Podfile*.

Výpis 4.2: Inicializace Podfile.

```
pod init
```

Následně lze tento vygenerovaný soubor otevřít. Právě do tohoto souboru je nutné přidat pody požadovaných knihoven a následně soubor uložit. *Podfile* pro vyvíjenou knihovnu je možné si prohlédnout ve výpisu 4.3.

Výpis 4.3: Podfile vyvíjené knihovny.

```
# platform :ios, '9.0'

target 'KevPlayer' do
  # Comment the next line if you're not using Swift
  and don't want to use dynamic frameworks
  use_frameworks!

  # Pods for KevPlayer
pod 'SnapKit', '~> 4.0.0'
pod 'GoogleAds-IMA-iOS-SDK', '~> 3.8'
pod 'Firebase/Core'
end
```

Po vložení všech potřebných podů a uložení souboru je možné tento dokument uzavřít a následně instalovat obsažené pody z terminálu, což je poslední krok pro práci s těmito pody. Příkaz pro instalaci je znázorněn ve výpisu 4.4 níže.

Výpis 4.4: Instalace podů.

```
pod install
```

Pro používání těchto podů je v posledním kroku nutností přestat používat původní soubor vývojového prostředí typu *.xcodeproj*. Inicializací vznikl ve složce nový spouštěcí soubor, jenž je typu *.xcworkspace*, který již obsahuje zmíněné frameworky.

4.3 Odesílání statistik

Aby bylo možné zjistit, zda knihovna funguje v pořádku a nezpůsobuje případný pád, je vhodné implementovat analytický nástroj. V kapitole 1.7 byl popsán jeden z těchto nástrojů, jenž se nazývá *Google Analytics*. Tato služba je zde implementována pomocí platofrmy *Firebase*, která slouží ke snadné implementaci služeb Googlu (včetně *Google Analytics*) pro vývojáře mobilních aplikací.

Implementace

Nejprve je nutné založit nový projekt na stránkách *Firebase* [1], kde je potřeba zadat požadované informace o projektu. Po úspěšném vytvoření je uživatel přesměrován již do konzole, kde může do projektu přidat různé funkcionality. V rámci této knihovny postačí přidat zmíněný analytický nástroj. Tuto funkci lze přidat stisknutím na tlačítko *Dashboard*, jenž se nachází v levé části konzole pod sekcí *Analytics*. Poté je potřebné zvolit platformu, pro kterou je projekt určen (zde je to *iOS*). Dále je nutné vložit konkrétní informace z vyvíjeného projektu, které jsou k dispozici například přes vývojové prostředí *Xcode*. Jedná se především o balíčkový identifikátor (*Bundle ID*) a případný identifikátor v obchodě *AppStore*. Poté se vygeneruje seznam vlastností (*GoogleService-Info.plist,*) který je nutný vložit do projektu ve vývojovém prostředí. V dalším kroce je potřeba přidat *Firebase SDK* do knihovny, které lze přidat přes *CocoaPods*.

V dalším kroce je nutné přidat kód do frameworku, aby se při inicializaci začali posílat data do konzole. Jedná se o pouhý řádek kódu, který je možný vidět ve výpisu 4.5.

Výpis 4.5: Konfigurace *Firebase*.

```
FirebaseApp.configure()
```

V posledním kroku se poté snaží *Firebase* spojit s knihovnou a pokud bylo vše nastavené správně, zobrazí se notifikace o úspěšném propojení.

4.3.1 Získávání informací o interakci uživatele s přehrávačem

Aby bylo možné zjistit, jaké schopnosti přehrávače jsou nejpoužívanější, je potřeba využít tzv. logy. Ty například dokážou indikovat, zda je jistá funkcionalita využívána a případně jak často. Jednotlivé logy jsou zde přiřazeny každé důležité funkčnosti a následně jsou zasílány do analytického nástroje. Každý log je nutný specifikovat klíčem, pod kterým v *Google Analytics* vystupuje. Lze také přiřadit parametry, které

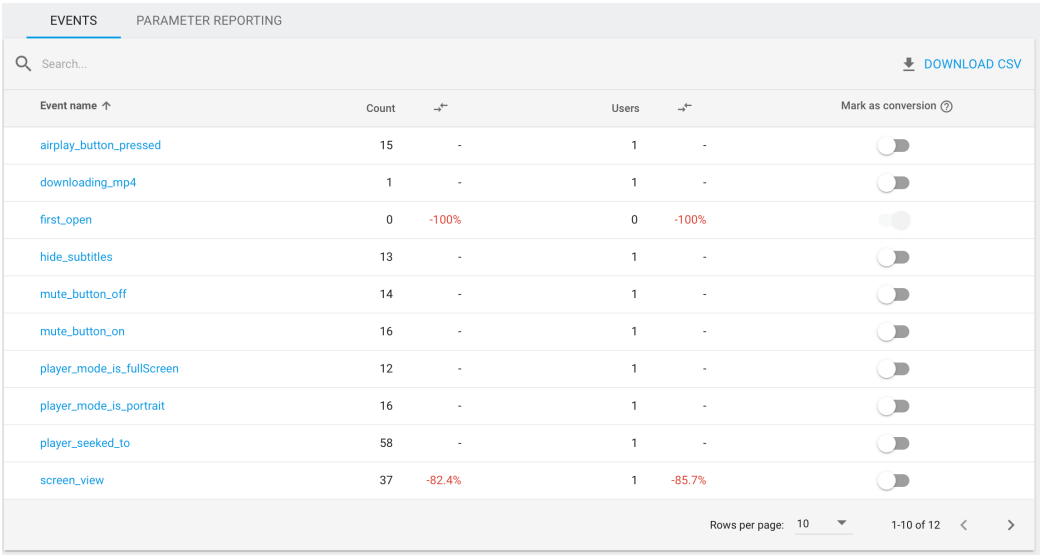
[1] Odkaz na stránky Firebase: <https://firebase.google.com>

například mohou zaznamenat, na jaký časový úsek videa se uživatel přemístil. Případ takového logu je znázorněn ve výpisu 4.6. Jak si lze všimnout, parametr je pole datového typu *Dictionary*, kde je opět získaná hodnota přiřazena k identifikovatelnému klíči.

Výpis 4.6: Log s parametrem.

```
Analytics.logEvent("progress_slider_value_changed",
parameters:
["progress_slider_value_changed_to_value":sender.value])
```

Obdobným způsobem je potřebné definovat veškeré ostatní aktivity. Zobrazení těchto tagů v analytickém nástroji je následně možné v sekci *Analytics* a záložka *Events*. Obsah této záložky je zobrazen na obrázku 4.3 níže.



Obr. 4.3: Zobrazení logů v Google Analytics.

Další zajímavé informace z analytického nástroje si lze prohlédnout v kapitole Seznam příloh.

4.4 Přehrávač

V této sekci je popsáno, jak probíhá inicializace videa pro přehrávače a jaké používá metody pro sledování stavů. Pro spuštění či zastavení videa používá přehrávač systémových funkcí knihovny *AVFoundation*, které je možné nad inicializovaným přehrávačem zavolat jednoduše pomocí metod *play()* či *pause()*.

4.4.1 Parametry pro inicializaci videa

Aby přehrávač věděl, co má vlastně přehrávat, je nutné mu určitým způsobem předat data, na jejichž základě sestaví přehrávač s požadovanými vlastnostmi. Pro inicializaci videa jsou nutné dva parametry. Prvním parametrem je předání abstrakní třídy *KevPlayerAsset* a druhým parametrem je začínající kvalita přehrávání, která je defaultně nastavena na první vložený. Metoda s parametry pro sestavení přehrávače je zobrazena ve výpisu 4.7.

Výpis 4.7: Sestavení přehrávače.

```
open func setVideoWith(asset: KevPlayerAsset,
                       selectedQuality: Int = 0)
```

První parametr, který je zde prezentován jako *asset*, je před samotným voláním metody pro inicializaci přehrávače také nutné naplnit požadovanými daty. Bez nich totiž není možné předat tento asset přehrávači. Tento asset má mnoho parametrů, přičemž některé z nich není nutné specifikovat. Jsou tedy tzv. typu *optional*. Přesně z tohoto *assetu* se získává jméno přehrávaného obsahu, odkazy na jednotlivé kvality, titulky, zda obsahuje reklamy a případně jaké či ilustraci obsahu. Poslední parametr očekává pole dat typu *Playlist* (objekt získaný při dekódování JSON řetězce), jenž obsahuje odkazy na další videa. Všechna tato data bude možné získat z playlistu, který bude knihovnou dekódován. Inicializace této abstraktní třídy, která se nazývá *KevPlayerAsset*, je možné vidět ve výpisu 4.8.

Výpis 4.8: Inicializace assetu.

```
public init(name: String = "",
            qualitySelection: [KevPlayerQualitySelection],
            coverImage: URL? = nil,
            subtitles: KevPlayerSubtitles? = nil,
            isAdsEnabled: Bool,
            adsTagUrl: String? = nil,
            nextAsset: String? = nil,
            obtainedPlaylists: [Playlist]? = nil)
```

Pokud by se rozepsaly jednotlivé parametry, tak první parametr *name* předává přehrávači jméno obsahu, jenž má být přehráván. Druhý parametr *qualitySelection* slouží pro předání možností změny kvality přehrávání. Tento parametr dále obsahuje další parametry (odkaz na obsah a jméno kvality). Tento parametr je více popsán v sekci 4.9. Další parametr (pro předání odkazu na ilustraci) je právě typu *optional*. To znamená, že se může, ale nemusí použít. Další parametr specifikující titulky, které

jsou typu *KevPlayerSubtitles* (popsáno v sekci Titulky), je též optional. Pátý parametr je typu *Bool*, očekává tedy hodnotu *true* či *false*. Na jejím základě se přehrávač dozví, zda přehrát reklamu. V případě, že je hodnota nastavena na *true*, musí se přehrávač ještě dozvědět tzv. *tag* reklamy, která má být přehrána. Tuto informaci právě předává parametr *adsTagUrl*, která je též *optional*. Aby bylo možné přehrát další obsah, kdy například skončí aktuální přehráváný, lze definovat předposlední parametr *nextAsset*. Ten je opět *optional* a typu *String*, kdy očekává odkaz na další obsah. Do posledního parametru *obtainedPlaylists* se předává celý získaný playlist z dekódování pro přehrávání dalšího obsahu. Parametr si lze tedy představit jako seznam obsahující videa k přehrávání.

Po nastavení *assetu* dochází k předání tohoto parametru přidané vrstvě přehrávače (*KevPlayerLayer*), která již spustí přehrávaný obsah. Tato vrstva ovládá další funkcionality a *observery* (naslouchače) přehrávače, mezi které například patří i zobrazování a správa reklam. Předání assetu této vrstvě je zobrazeno ve výpisu 4.9.

Výpis 4.9: Předání assetu vrstvě s přehrávačem.

```
playerLayer?.playAsset(asset: asset.avURLAsset)
```

4.4.2 Metody delegáta přehrávače

Pro přehrávač byl také vytvořen protokol, jenž obsahuje metody zaznamenávání různých stavů přehrávače. Celkově je metod pět a jejich název a funkčnost je následující:

- *playerStateDidChange* - Jedná se o metodu, která informuje o aktuálním stavu přehrávače. Celkově indikuje pět stavů, které mohou nastat. První stav (*notSetURL*) pojednává o tom, že nebyla nastavena žádná adresa pro přehrávané video. Druhý stav (*readyToPlay*) informuje testovanou aplikaci o tom, že jsou potřebná data nastavena a přehrávač je připraven začít přehrávat požadovaný obsah. Třetí stav (*buffering*) probíhá při načítání dat obsahu. Tohle je možné například indikovat u přehrávání videa ve formátu MP4, kdy se video musí před spuštěním načíst. Čtvrtý stav (*bufferFinished*) znamená, že načítání dat již bylo kompletně ukončeno. Pátý stav (*playedToTheEnd*) informuje o tom, že přehrávač přehrál přehrávaný obsah až do konce. Šestý a zároveň poslední stav (*error*) zachytává případné chyby.
- *playerLoadedTimeDidChange* - Metoda sloužící pro informaci o změně načteného času.
- *playerTimeDidChange* - Tato metoda předává informaci o aktuální časové hodnotě přehrávaného videa.

- *kevPlayerIsPlaying* - Metoda informující o stavu, zda probíhá přehrávání či nikoliv.
- *playerOrientationChanged* - Poslední metoda informuje o změně polohy přehrávače při přehrávání.

4.5 Ovládací panel přehrávače

Aby bylo možné přehrávaný obsah určitým způsobem ovládat, je nutné mít k němu ovládací panel. Ten je v této knihovně implementován jako komponenta *UIView()*, která se nastaví při inicializaci přehrávače. Ovládací panel obsahuje několik tlačítek, časovou osu videa a také informační pole, která obsahují informace ohledně přehrávaného obsahu. Mezi tyto data patří jméno videa, jeho aktuální a celkový čas a titulky. Tlačítek je celkem devět. Jedním tlačíkem lze spustit či pozastavit přehrávaný obsah, druhým skrýt či zobrazit titulky a třetím vybrat kvalitu přehrávaného videa. Čtvrté tlačíko slouží pro stáhnutí obsahu, páté pro spuštění obsahu na televizi pomocí *AirPlay* a šesté pro ztišení zvuku obsahu. Sedmé tlačítko slouží pro přepnutí polohy přehráváného obsahu (vertikální či horizontální) a poslední dvě tlačítka k přepínání obsahu přehrávače pro přehrávání.

Jak již bylo zmíněno, panel také obsahuje časovou osu videa. Posunutím kurzoru na libovolnou pozici osy dojde k přetočení videa na odkazující časovou stopu. Video lze také přetáčet na různé pozice bez použití této osy, neboť je implementována funkce přetáčení pomocí přejetí do levého či pravého směru v celé šířce zobrazení videa.

Komponent v ovládacím panelu je mnoho. Pro jejich rozložení na obrazovce je použita knihovna třetí strany, jenž se nazývá *Snapkit*. Ta umožňuje jednoduchým způsobem nastavit konstrainty pro jednotlivé objekty. Ve výpisu 4.10 lze vidět nastavení konstraintů pro objekt tlačítka, jenž umoňuje stahování.

Výpis 4.10: Použití knihovny Snapkit.

```
downloadVideoButton.snp.makeConstraints { (make) in
  make.right.equalTo(subtitlesButton.snp.left).offset(-20)
  make.top.equalTo(subtitlesButton.snp.top)
  make.width.equalTo(80)
}
```

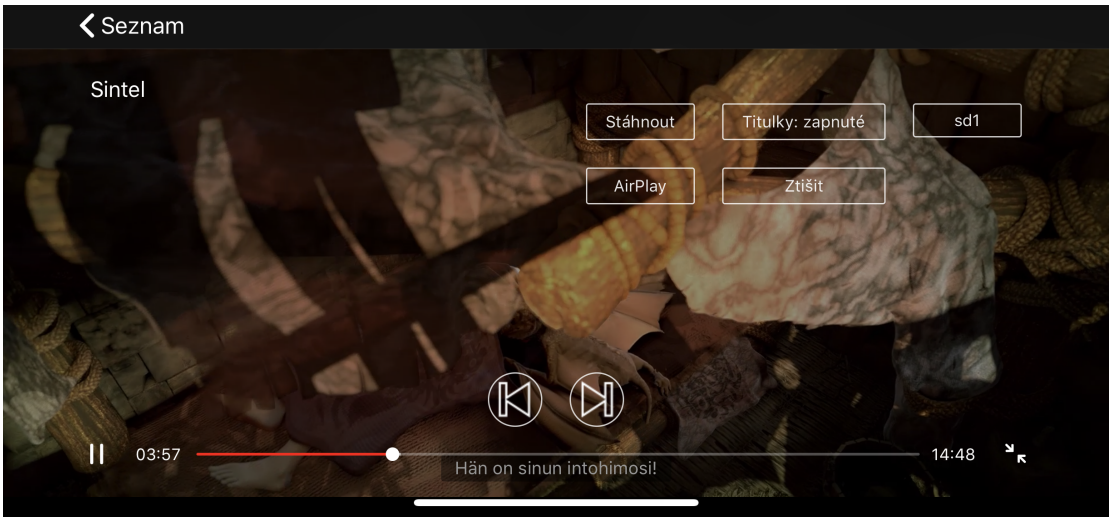
Při představě, že ve zmíněném výpisu je nastaven pouze jeden objekt a velmi podobným způsobem je nutné nastavit i ostatní komponenty je zřejmé, že *Snapkit* zanáší do kódu velký počet řádků kódu. To lze brát jako jednu nevýhodu použití této knihovny. Na druhou stranu je ale pomocí něj možné detailní nastavení jednotlivých komponent.

Přehrávač ale nutně nemusí používat panel s takovým rozložením. Jednoduše si lze vytvořit vlastní a ten použít namísto zmíněného. Ve výpisu 4.11 je možné zhlédnout inicializaci panelu ovládání pro přehrávač v již testovací aplikaci, kde kontrolér může být jiný vytvořený panel.

Výpis 4.11: Inicializace panelu ovládaní.

```
    player = KevPlayer(customControlView: controller)
}
```

Vzhled defaultního panelu ovládání s již přehrávaným obsahem je zobrazen na obrázku 4.4 níže. Celkové nastavení všech komponent umožňující interakci s přehrávaným obsahem, jsou k nalezení ve třídě *KevPlayerUIControlPanel*. Kromě samotného rozložení objektů zde také dochází k nastavení některých jejich funkcionalit a také vzhledu. Pro tlačítka a textová pole se zde dále nastavuje písmo, barva, velikost písma atd.



Obr. 4.4: Ovládací panel přehrávače.

4.6 Celoplošné přehrávání obsahu

Původní zobrazení přehrávače je nastavené pro zobrazení ve vertikální poloze (*portraid mode*). Pro celoplošné přehrávání stačí otočit telefon do horizontální polohy (*landscape mode*), což zahrnuje volání metody delegáta přehrávače. To způsobí navazující změnu konstraintů jednotlivých objektů a následné zobrazení přehrávače v této poloze. Pro změnu polohy přehrávání je však v knihovně vložena i funkce, která v důsledku stisknutí tlačítka pro změnu polohy tuto polohu změní. Metodu je

možné si prohlédnout níže ve výpisu 4.12. Funkce pracuje s parametrem *isForFullscreen*, který obdrží od nastavení přehrávače. Tento parametr dále podsouvá další metodě, která na jeho základě upraví zobrazení/skrytí dalších objektů.

Výpis 4.12: Změna polohy.

```
open func setUpdatedOrientation(_ isForFullScreen: Bool) {
     isFullScreen = isForFullScreen
     fullscreenButton.isSelected = isForFullScreen
     selectQualityView.isHidden =
     !KevPlayerConfiguration.isSelectQualityEnabled ||
     !isForFullScreen
     setTopBar(isFullScreen: isForFullScreen)
}
```

4.7 Přepínání obsahu v rámci přehrávače

Pokud by uživatel chtěl vidět jiný obsah, musí deinicializovat přehrávač, čímž se vrátí (například v testovací aplikaci) do seznamu videí. Následně by musel vybrat požadovaný obsah, kterým vyvolá inicializaci přehrávače s požadovanými parametry. Vyvíjený přehrávač však umožňuje měnit parametry i v rámci přehrávače bez nutnosti návratu do nabídky. Funkcionalita probíhá tak, že se knihovně při inicializaci přehrávače v parametru *obtainedVideos* předá celý dekódovaný JSON, jenž obsahuje jednotlivá videa. Při stisku tlačítka pro zobrazení dalšího videa dojde k vybrání náhodného videa ze získaného JSON řetězce a následně k jeho přehrání. Aby se také zamezilo duplicitě, je již přehraný obsah ze seznamu odstraněn. Pokud by již seznam byl prázdný, dojde opět k jeho naplnění. Tato funkce je možná díky přiřazení indexu každému videu z řetězce. Po získání odkazu na další video dojde tedy již ke klasické inicializaci přehrávače s parametry, které jsou získány ze zmíněného odkazu na video. Metoda pro přiřazení indexu k videím je znázorněna níže ve výpisu 4.13.

Výpis 4.13: Výběr náhodného obsahu.

```
func randomItems() -> Playlist {
    if indexes.count == 0 {
        indexes = Array(0 ..<
        (asset.obtainedPlaylist.count)
    }
    let randomIndex =
    Int(arc4random_uniform(UInt32(indexes.count)))
```

```
    let anIntex = indexes.remove(at: randomIndex)
    return (asset.obtainedPlaylist[anIndex]
}
```

4.8 Ztlumení přehrávače

Někdy se může stát, že uživatel potřebuje přehrávaný obsah instantně ztlumit, aby například mohl rychle slyšet něco jiného. Taková funkcionalita, jenž se nazývá ztlumení přehrávače, je v přehrávači též implementována. Jedná se o jednoduchou metodu, která po stisknutí tlačítka zjistí hodnotu hlasitosti přehrávače. Pokud hodnota není rovna nule, nastaví tuto hodnotu na nulu, čímž dojde ke ztlumení. V opačném případě, kdy je hlasitost nastavena na nulu, dojde k nastavení hodnoty hlasitosti na deset. Jedná se tak tedy o jednoduchou metodu s podmínkou typu *if else*, která pouze kontroluje úroveň hlasitosti. Popsanou funkci lze vidět níže ve výpisu 4.14.

Výpis 4.14: Ztlumení přehrávače.

```
            @objc open func muteVolume() {
        if player?.avPlayer?.volume == 0 {
            player?.avPlayer?.volume = 10.0
        } else {
            player?.avPlayer?.volume = 0
        }
    }
```

4.9 Výběr kvality přehrávání

Pokud je v přehrávači obsah ve formátu HLS, tak se kvalita obrazu přizpůsobuje dle velikosti toku dat. V případě formátu MP4 je však nutné pro zvýšení/snížení kvality načíst nový MP4 obsah s požadovanou kvalitou.

Požadované kvality je nutné získat při dekódování playlistu. Funkcionalita přitom funguje tak, že na základě počtu získaných odkazů MP4 videí vygeneruje stejný počet tlačítek. Každé tlačítko poté odkazuje na příslušnou kvalitu. V rámci implementace této vyvinuté knihovny do testovací aplikace je počet tlačítek omezen na tři.

Po stisknutí tlačítka pro sledování obsahu s požadovanou kvalitou je přehrávání aktuálního videa zastaveno a začne se načítat nový obsah, který se začne přehrávat od stejného časového úseku, kde skončilo původní video. Díky této funkcionalitě nemusí uživatel hledat, ve které části změnil kvalitu.

Pro zobrazení možnosti tlačítek potřebuje knihovna pro výběr kvality v inicializaci dva parametry. První je datového typu *URL*, což je odkaz na obsah v požadované kvalitě a druhý parametr je typu String, který charakterizuje pojmenování tlačítka. Pojmenování také charakterizuje volbu kvality. Popsanou inicializaci je možné vidět ve výpisu 4.15.

Výpis 4.15: Parametry pro inicializaci výběru kvality.

```
      public init(url: URL, qualityName: String) {
        self.url = url
        self.qualityName = qualityName
    }
```

Inicializace samotného výběru kvality v přehrávání bude zobrazena v sekci věnované inicializaci přehrávače s dalšími funkcemi.

Níže ve výpisu 4.16 je ukázka kódu, jenž na základě počtu získaných objektů výběru kvalit vytváří do ovladacího panelu přehrávače jednotlivá tlačítka. Jak již bylo řečeno, z důvodu testovací aplikace je zde omezen počet výběru kvalit na tři tlačítka, k čemuž pomáhá pomocná proměnna *limitedCountOfQualitySelections*.

Výpis 4.16: Výběr kvalit.

```
var limitedCountOfQualitySelections =
asset.qualitySelections
if limitedCountOfQualitySelections.count > 1 {
 let prefix = asset.qualitySelections.prefix(3)
 limitedCountOfQualitySelections = Array(prefix)
        }
for i in 0..<limitedCountOfQualitySelections.count {
    let QualityIdentifierButton = KevPlayerCustomButton()
    if i == 0 {
     QualityIdentifierButton.tag = selectedButtonQuality
        } else if i <= selectedButtonQuality {
            QualityIdentifierButton.tag = i - 1
        } else {
            QualityIdentifierButton.tag = i
        }
    QualityIdentifierButton.setTitle
("\(asset.qualitySelections[QualityIdentifierButton.tag]
.qualityName)", for: UIControl.State())
    selectQualityView.addSubview(QualityIdentifierButton)
    QualityIdentifierButton.addTarget(self,
```

```
    action: #selector(self.selectQualityTapped(_:)),
    for: .touchUpInside)
```

4.10 Titulky

Pokud je přehrávaný obsah například v cizím jazyce, je vhodné k takovému obsahu zobrazovat i titulky, které je samozřejmě také možné během přehrávání skrýt či naopak zobrazit. Ve vyvinuté knihovně je přidána podpora pro titulky formátu *.srt*.

Implementace titulek je vložena do třídy *KevPlayerSubtitles*, která pro inicializaci požaduje parametr datového typu *URL*. Druhý parametr je kódování, ten je však typu *optional*. Není tedy nutné jej specifikovat. Inicializace třídy je znázorněna ve výpisu 4.17 níže.

Výpis 4.17: Parametry pro inicializaci titulků.

```
public init(url: URL, encoding: String.Encoding? = nil)
```

Následně dojde k ověření parametru url a v případě, že je odkaz validní, dojde k volání funkce pro rozdělení titulek do jednotlivých pasáží. Ta poté vrací pole jednotlivých pasáží titulek, které jsou poté zobrazovány v požadovaný čas. Inicializace jednotlivých pasáží je znázorněna ve výpisu 4.18.

Výpis 4.18: Parametry pro inicializaci jednotlivých sekvencí.

```
init(_ numberOfSequence: Int, _ start: NSString,
     _ end: NSString, _ text: NSString)
```

Metoda pro rozdělení titulek do jednotlivých sekvencí a jejich následné přidání do pomocné proměnné typu pole (*array*) je znázorněno ve výpisu 4.19. Jak si lze všimnout, funkce obsahuje jediný parametr, kde již očekává dekódovaný text z titulek. Na základě značek a symbolů z celého textu je metoda schopna rozdělit text do zmíněných sekvencí a následně je přidat do pole sekvencí. Dále je možné vidět, že metoda má návratovou hodnotu typu pole objektů *Sequence*. Toto získané pole je následně přiřazeno pomocné proměnné, která se dále používá pro zobrazování požadovaných titulek.

Výpis 4.19: Metoda pro rozdělení titulek.

```
fileprivate static func parseSubtitles(_ payload: String)
-> [Sequence]? {
    var parsedSequences: [Sequence] = []
    let scanner = Scanner(string: payload)
    while !scanner.isAtEnd {
```

```
            var indexString: NSString?
            scanner.scanUpToCharacters(from: .newlines,
            into: &indexString)
            var firstString: NSString?
            scanner.scanUpTo(" --> ", into: &firstString)
            scanner.scanString("-->", into: nil)
            var lastString: NSString?
            scanner.scanUpToCharacters(from: .newlines,
            into: &lastString)
            var textString: NSString?
            scanner.scanUpTo("\r\n\r\n", into: &textString)
            if let text = textString {
                textString = text.trimmingCharacters
                (in: .whitespaces) as NSString
                textString = text.replacingOccurrences
                (of: "\r", with: "") as NSString
            }
            if let indexString = indexString,
                let index = Int(indexString as String),
                let start = firstString,
                let end = lastString,
                let text = textString {
                let sequence = Sequence(index, start,
                end, text)
                parsedSequences.append(sequence)
            }
        }
        return parsedSequences
}
```

4.11 Reklamy

Mezi další funkcionality přehrávače jsou zahrnuty reklamy typu VAST, které jsou zde do knihovny implementovány pomocí knihovny třetí strany firmy *Google*, která se nazývá *Interactive Media Ads*, zkráceně *IMA*. Firma vývojářům poskytuje volně dostupné SDK obsahující různé metody pro správu a realizaci reklam v přehrávači [33].

4.11.1 Ad tag

Přehrávač je schopen přehrávat reklamy pomocí tagů. Obecně řečno je tag HTML kód, který prohlížeč používá k načtení reklamy z reklamního serveru (*Ad server*). Důležité je zmínit, že se především jedná pouze o přesměrování na určitý obsah, nežli odkaz na samotný obsah. Typů tagů je několik, přičemž některé mohou například odkazovat na reklamy kliknutí (což způsobí přesměrování na požadovaný obsah).

Popis jednotlivých částí tagu

- **http://ad.doubleclick.net/** - Adresa hostitele pro reklamní server. Nejedná se o webové stránky vydavatele, ale o nezávislou společnost, která nemá nic společného s publikovaným obsahem.
- **/ADJ** - Kód specifikující typ reklamního volání a jaká může být odpověď (např. XML). Místo ADJ například může být i ADF či ADX.
- **/publisher**- Jedná se o kód webu, který služba *Doubleclick* používá k rozlišení vlastností vydavetele od jiného.
- **/zone** - Specifikuje oblast či kanál, na kterou je reklama zaměřena. Může se například jednat o sport, módu, vaření apod.
- **topis=abc** - Podobně jako zóna. Touto částí tagu lze definovat určité téma napříč zónami.
- **sbtpc=def** - Definuje tzv. subtropickou úroveň. Prakticky opět slouží k cílení na určité obsahy.
- **kw=xyz**- Specifikuje způsob, jak popsat stránku pro kontextové cílení, přičemž je povoleno více klíčových slov.
- **title=1** - Nastavuje jedinečnou hodnotu pro každé reklamní volání na určité stránce. Toho se využívá například v případě, kdy jsou na stránce dvě a více reklam. Prohlížeč se tak nebude pokoušet současně zobrazovat stejnou reklamu na více místech.
- **slot=728x90.1** - Definuje umístění reklamy.
- **sz=728x90**- Specifikuje velikost reklamy pro reklamní server.
- **ord=7268140825331981** - Pokud se uživatel například z jedné stránky, kde jsou již reklamy, přesune na jinou stránku a poté zpět na původní, dojde ke změně reklam. Tato část tagu tedy slouží pro zábranu zobrazování stejných reklam, což umožní uživateli zobrazovat více odlišných reklam [34].

Příklad takové reklamy uvádí například sám *Google*, která je znázorněna ve výpisu 4.20, která je tzv. typu pre-roll. To znamená, že se uváděná reklama zobrazí před spuštěním požadovaného videa. Dalším typem může být mid-roll (reklama zobrazená v průběhu přehrávání) či post-roll (zobrazená na konci).

Výpis 4.20: Příklad reklamy.

```
let kTestAppAdTagUrl =
"https://pubads.g.doubleclick.net/gampad/ads?sz=640x480&"
+"iu=/124319096/external/single_ad_samples&ciu_szs=300x
250&impl=s&"+"gdfp_req=1&env=vp&output=vast&unviewed_po
sition_start=1&" +"cust_params=deployment%3Ddevsite
%26sample_ct%3Dlinear&correlator=";
```

4.11.2 Implementace reklam do knihovny

Implementace SDK do knihovny proběhla dle tutoriálů na oficiálních stránkách poskytovatele [33].

Nezbytným krokem je import modulu *GoogleInteractiveMediaAds* do příslušné třídy a také implementace delegatů *IMAAdsLoaderDelegate* a *IMAAdsManagerDelegate*, které slouží pro správné načítání a zobrazení požadovaných reklam. Pro případné mid-roll reklamy je také nutné sledovat přesnou časovou pozici videa, aby SDK vědělo, kdy požadovanou reklamu načíst. Metody delegátů také reagují například na pozastavení reklamy a případnou chybu. Ta může nastat z mnoha důvodů, přičemž nejčasteji se jedná o žádné nebo přílis slabé připojení k internetu či invalidní formát reklamy. V takovém případě je nutné nastavit, aby taková chybná reklama byla ignorována a následně došlo ke spuštění požadovaného obsahu. Jak již bylo řečeno, metody delegátů tohle vše řeší. Metoda pro řešení chybné reklamy je znázorněna ve výpisu 4.21. Funkcí této metody je výpis logu chyby do konzole a následné zavolání funkce přehrávače *play()* pro přehrávání videa. Uživatel tedy tak v reálném použití v takové situaci ani nepozná, že nastala nějaká chyba.

Výpis 4.21: Metoda delegátu pro řešení chybných reklam.

```
public func adsManager(_ adsManager: IMAAdsManager!,
didReceive error: IMAAdError!) {
    NSLog("AdsManager error:
    \(String(describing: error.message))")
    play()
}
```

Reklamu zapsanou ve výpisu 4.20 lze poté použít v metodě requestAds(), přičemž při jejím zavolání dojde poté k zobrazení reklamy v požadovaný čas. Metoda je zobrazena ve výpisu 4.22.

Výpis 4.22: Žádost o reklamu.

```
    var request = IMAAdsRequest(
```

```
        adTagUrl: kTestAppAdTagUrl,
        adDisplayContainer: adDisplayContainer,
        contentPlayhead: contentPlayhead,
        userContext: nil)

    adsLoader!.requestAds(with: request)
```

4.12 Dekódování playlistu

Pokud aplikace, kde je knihovna využívána, zavolá metodu knihovny *decodePlaylist*, dojde k dekódování přiřazeného JSON řetězce, který obsahuje celý playlist pro daný produkt. Metoda pro dekódování je znázorněna ve výpisu 4.23. Funkce využívá systémové třídy *JSONDecoder* a její metody *decode*, která ve vývojářem vytvořené struktuře dekóduje získaná data.

Výpis 4.23: Dekódování playlistu.

```
public func decode(playlist: String) {
    guard let url = URL(string: playlist) else { return }
    do {
        let data = try Data(contentsOf: url)
        let videosParsedObjects = try JSONDecoder().decode
        (Videos.self, from: data)
        videos.append(videosParsedObjects)
```

Pokud je dekódování chybné, zachytí se chyba v sekci metody *catch* a následně ji vypíše do konzole. Případná zpráva je znázorněna ve výpisu 4.24.

Výpis 4.24: Zachycení chyby dekódování.

```
  catch {
        print("JSON CONTENT PARSING FAILED",error)
        }
```

Jak již bylo řečeno, dekódér ukládá objekty na základě vytvořené struktury. Ta implementuje systémový protokol *Decodable* a nesmí obsahovat žádné objekty, které v řetězci nejsou. Pokud nějaké takové objekty jsou, například z důvodu, že se někde v nějakém playlistu mohou vyskytovat, je vhodné tento objekt vytvořit jako *optional*. Pokud by dekódér při dekódování nenanarazil na tento objekt v řetězci a k tomu by nebyl *optional*, mohlo by celé dekódování být zrušeno (neboť by nevěděl, jakou hodnotu objektu přiřadit). Struktur pro dekódování playlistu je mnoho. Ve výpisu 4.25 je k nalezení alespoň krátká ukázka, jak taková struktura vypadá. Důležité je

také zmínit, že v případě těchto koknrétních struktur je nutné, aby jednotlivé názvy objektů byly stejné jako názvy objektů v JSON řetezci.

Výpis 4.25: Struktura pro dekódování playlistu.

```
public struct Playlist: Decodable {
    public let title: String
    public let subtitle: String?
    public let streamInfos: [StreamInfos]?
    public let subInfos: [SubInfos]?
    public let videoTimeFlags: [VideoTimeFlags]?
    public let adsEnabled: Bool?
}

public struct StreamInfos: Decodable {
    public let type: String
    public let url: String
}
```

Po úspěšném dekódování jsou data přidána do pomocné proměnné *videos* usnadňující přístup k datům z jiných metod.

4.13 Stahování přehrávaného obsahu

V případě, že by si uživatel chtěl později přehrát určitý obsah, ale neví, zda bude mít k dispozici připojení k internetu, je možné stáhnout přehrávaný obsah na lokální úložiště zařízení. Toto stažené video lze poté později přehrát v aplikaci a uživatel jej nalezne pod jménem, jenž získá předem z řetězce JSON produktů.

4.13.1 Stahování formátu MP4

O ukládání na zařízení se stará systémová třída *FileManager*, která umožňuje soubory nejen vytvářet, ale také je různě přesouvat, kopírovat apod. V prvním kroce je potřebné získat referenci na *FileManager*, jako je znázorněno ve výpisu 4.26.

Výpis 4.26: Reference na FileManager.

```
guard let documentsUrl: URL = FileManager.default.urls
(for: .documentDirectory, in: .userDomainMask).first
else { return }
```

V dalším kroku dojde k přidání cesty k obsahu a nastavení odkazu z parametru metody, která je v datovém typu *URL*. Druhý a poslední parametr metody pro

stažení je datového typu *String*, jenž poslouží jako identifikace obsahu do úložiště. Po nastavení těchto hodnot je možné nastavit síťovou konfiguraci, jako je znázorněno níže ve výpisu 4.27.

Výpis 4.27: Nastavení hodnot pro úložiště.

```
let destinationFileUrl =
documentsUrl.appendingPathComponent("\(name).mp4")
    let fileURL: URL = url
    let sessionConfig = URLSessionConfiguration.default
    let session = URLSession(configuration: sessionConfig)
    let request = URLRequest(url: fileURL)
```

V případě úspěchu dochází ke kopírování souboru do lokálního úložiště. Jak si je možné všimnout v dalším výpisu 4.28, přístup k datům je řešen přes další systémovou třídu *UserDefaults* sloužící jako úložiště založené na kombinaci hodnoty a klíče.

Výpis 4.28: Kopírování souboru.

```
try FileManager.default.copyItem(at: tempLocalUrl,
to: destinationFileUrl)
        self.offlineVideoNames =
UserDefaults.standard.stringArray
(forKey: "OfflineVideoNames") ?? [String]()
        self.offlineVideoLinks =
UserDefaults.standard.stringArray
(forKey: "OfflineVideoLinks") ?? [String]()
        self.offlineVideoNames.append(name)
        self.offlineVideoLinks.append
        ("\(destinationFileUrl)")
        let defaults = UserDefaults.standard
   UserDefaults.standard.set(self.offlineVideoNames,
 forKey: "OfflineVideoNames")
UserDefaults.standard.set(self.offlineVideoLinks,
forKey: "OfflineVideoLinks")
```

V případě, že by kdekoliv během síťové komunikace či kopírování souboru nastala chyba, je tato chyba vypsána do konzole.

4.13.2 Stahování formátu HLS

Nejprve je nutné implementovat delegát *AVAssetDownloadDelegate*, jenž poskytuje metody pro zjištění, v jakém aktuálním stavu se stahování nachází. Lze tak například

zjistit, kolik procent je již staženo či zda se vyskytla chyba. Metoda pro stažení je znázorněna níže ve výpisu 4.29. Jak si lze všimnout, metoda opět pracuje se stejnými parametry. Důležité je nastavit konfiguraci, která získá identifikátor pomocí identifikátoru balíčku (*Bundle ID*).

Výpis 4.29: Stahování formátu HLS.

```
func downloadHlsWith(link: URL, title: String) {
    configuration = URLSessionConfiguration.background
    (withIdentifier: downloadIdentifier)
    downloadSession = AVAssetDownloadURLSession
    (configuration: configuration!,
    assetDownloadDelegate: self,
    delegateQueue: OperationQueue.main)
    let asset = AVURLAsset(url: link)
    let downloadTask =
    downloadSession?.makeAssetDownloadTask
    (asset: asset, assetTitle: title,
    assetArtworkData: nil, options: nil)
    videoHLSTitle = title
    downloadTask?.resume()
}
```

Metoda, zobrazená ve výpis 4.30, prezentuje v ovládacím panelu progres stahování v procentech. Díky této metodě uživatel vidí, kolik procent je již staženo pro offline přehrávání.

Výpis 4.30: Progres stahování.

```
func urlSession(_ session: URLSession,
assetDownloadTask: AVAssetDownloadTask,
didLoad timeRange: CMTimeRange,
totalTimeRangesLoaded loadedTimeRanges:
[NSValue], timeRangeExpectedToLoad:
CMTimeRange) {
    var percentComplete = 0.0
    for value in loadedTimeRanges {
        let loadedTimeRange =
        value.timeRangeValue
        percentComplete +=
        loadedTimeRange.duration.seconds /
        timeRangeExpectedToLoad.duration.seconds
```

```
    }
    percentComplete *= 100
    let roundedPercentage =
    Double(round(1000*percentComplete)/1000)
    self.downloadVideoButton.setTitle
    ("\(roundedPercentage)%", for: .normal)
    }
}
```

Poslední metoda, která je z delegátu použita, slouží pro vykonání funkce po dokončení stahování. V této metodě probíhá nastavení titulu tlačítka na "Staženo"a dále se zde také nastavuje klíč, pod kterým je možno stažené video přehrát. Metodu si je možné prohlédnout níže ve výpisu 4.31.

Výpis 4.31: Nastavení klíče pro stažené video.

```
public func urlSession(_ session: URLSession,
assetDownloadTask: AVAssetDownloadTask,
didFinishDownloadingTo location: URL) {
    offlineHLSVideoLinks =
    UserDefaults.standard.stringArray
    (forKey: "testVideoPath") ?? [String]()
     UserDefaults.standard.set
     (location.relativePath, forKey: "testVideoPath")
     self.downloadVideoButton.backgroundColor = .clear
     self.downloadVideoButton.setTitle
     (Constants.Titles.downloaded, for: .normal)
}
```

4.14 Airplay

Vyvinutá knihovna také podporuje funkci *Airplay*, která například umožňuje zobrazit přehrávaný obsah z mobilního zařízení na televizi pomocí *AppleTV* či přenést zvukovou stopu do reproduktorů. Zobrazení výběru zařízení, kde má být přehrávání zobrazeno, probíhá automaticky při implementaci třídy *MPVolumeView*, jenž je součástí knihovny *MediaPlayer*. Implementace je zobrazena ve výpisu 4.32. Znázorněnou funkci lze vyvolat stisknutím přiřazeného tlačítka s titulem *AirPlay* v ovládacím panelu, což zobrazí tabulku s dostupnými zařízeními. Po stisku na požadované dojde na vybraném zařízení k přenesení obrazu i zvuku, přičemž obsah je stále možný ovládat pomocí mobilního zařízení.

Výpis 4.32: Implementace AirPlay.

```
private func showAirPlaySettings() {
    let rect = CGRect(x: -100, y: 0, width: 0, height: 0)
    let airplayVolume = MPVolumeView(frame: rect)
    airplayVolume.showsVolumeSlider = false
    mainMaskView.addSubview(airplayVolume)
    for view: UIView in airplayVolume.subviews {
        if let button = view as? UIButton {
            button.sendActions(for: .touchUpInside)
            break
        }
    }
    airplayVolume.removeFromSuperview()
}
```

4.15 Podpora přehrávání chráněného obsahu

Další funkcionalitou přehrávače mělo být přehrávání zabezpečeného obsahu DRM pomocí *Fairplay*, ale vývojáři byl odmítnut přístup k produkčnímu SDK ze strany Apple. SDK je zpřístupněno pouze majiteli účtu vývojového týmu a také vlastníkovi licencovaného obsahu. Vývojový balíček také není zpřístupněn třetím stranám jednající jménem vlastníků licencovaného obsahu [17].

V následující sekci je tedy alespoň znázorněn princip implementace této funkcionality. V prvním kroku je potřebné do třídy, která se stará o přehrávání, implementovat funkce delegáta *AVAssetResourceLoaderDelegate*. Z tohoto delegátu by pro základní nastavení měla stačit prozatím jediná metoda, která se automaticky zavolá při pokusu o přehrávání zabezpečeného obsahu.

Metoda se nazývá *shouldWaitForLoadingOfRequestedResource loadingRequest*, ve které by měla probíhat žádost na licenční server s požadovaným klíčem a certifikátem. V případě úspěchu a správné konfiguraci funkčnosti v kódu server vrací dešifrovaná data, která lze již použít v přehrávači. Tuto funkčnost však nebylo možné ověřit.

Důležitá část kódu této funkce, která by byla použita pro tuto funkcionalitu, je znázorněna níže ve výpisu 4.33. Znázorněné části kódu obsahují hodnoty typu *String*, přičemž některé z nich by byly nahrazeny získanými daty.

Výpis 4.33: Implementace FairPlay.

```
let url = requestedUrl
```

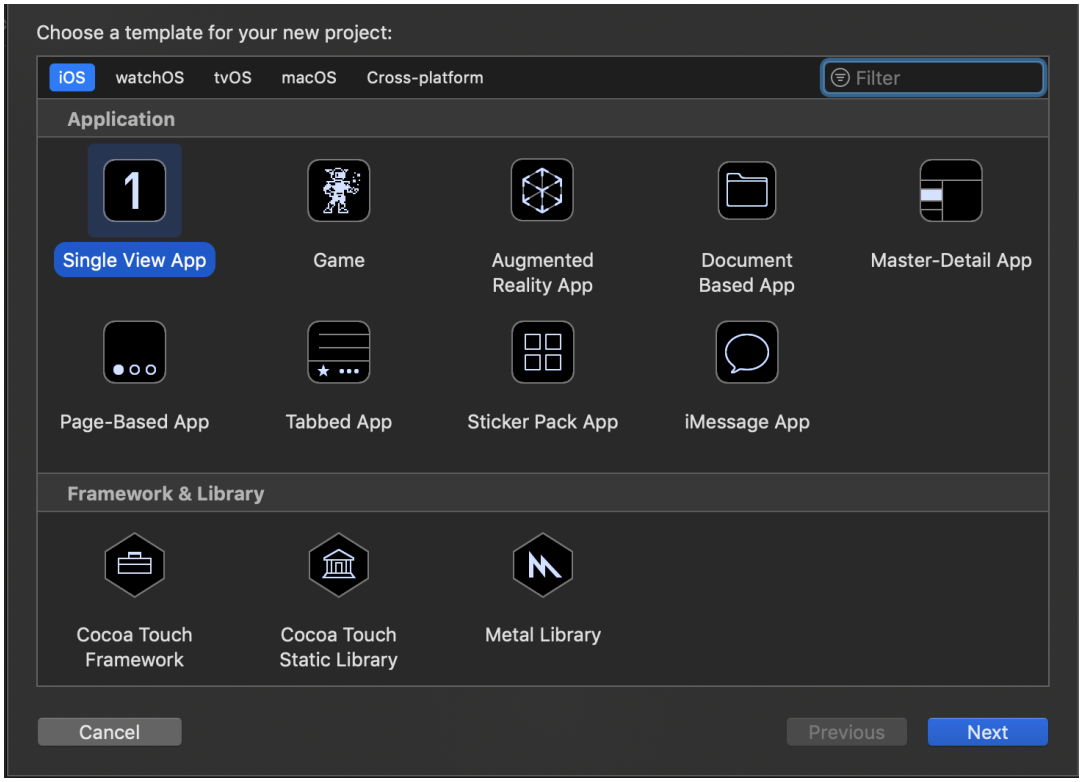
```
let certificateURL =
Bundle.main.url(forResource: "certificate",
withExtension: "der")
let certificateData = try?
Data(contentsOf: certificateURL)
let contentId = "contentID"
let decodedData = contentId.data
(using: String.Encoding.utf8)
let serverContentData = try?
loadingRequest.streamingContentKeyRequestData
(forApp: certificateData,
contentIdentifier: contentIdData,
options: nil)
.
.
let contentKey = URL(string: "contentKeyContext")
let request = URLRequest(url: contentKey)
    request.httpMethod = "POST"
    request.httpBody = serverContentData
 let session =
 URLSession(configuration:
 URLSessionConfiguration.default)
 let task = session.dataTask(with: request) {
 data, response, error in
    if let data = data {
    dataRequest.respond(with: data)
    loadingRequest.finishLoading()
    } else {
        .
        .
    }
task.resume()
}
```

5 Vývoj testovací aplikace

Aby bylo možné otestovat funkčnost jednotlivých funkcionalit vyvíjené knihovny, byla vytvořena testovací alikace, do které je knihovna implementována. Zmíněná aplikace obsahuje celkem čtyři obrazovky (seznam videí, přehrávač, stažený obsah a o aplikaci).

5.1 Vytvoření projektu

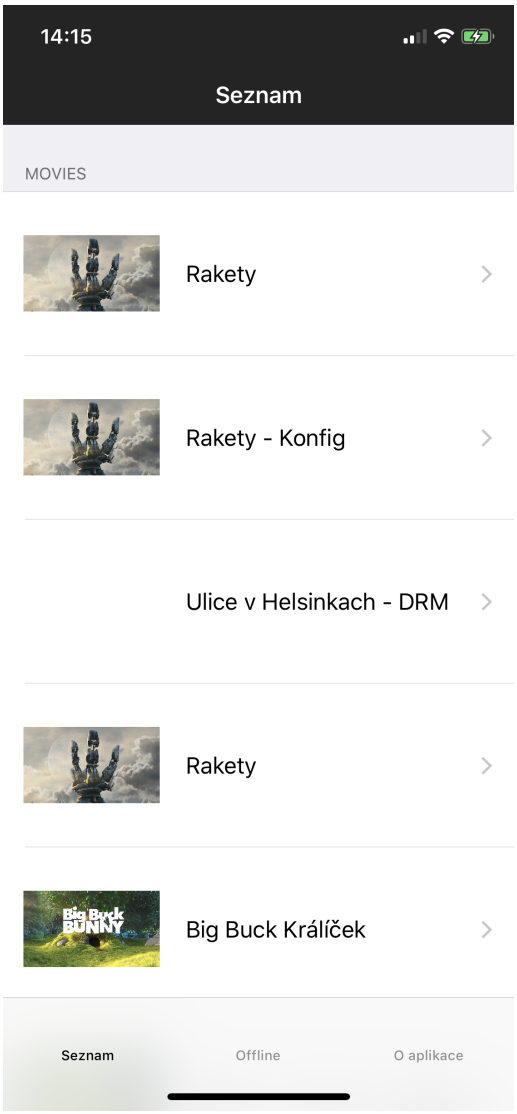
Pro začátek vývoje je opět nutné vytvořit nový projekt. Proces probíhá téměř identicky jako pro vývoj knihovny. Jediný zásadní rozdíl je pouze ve výběru šablony nového projektu, který je zde *Single View App*. Testovací aplikaci jako takovou lze vytvořit i v projektu knihovny. Je však vhodné si tyto dva projekty rozdělit a spravovat je separátně. Zvolení šablony je možné zhlednout na obrázku 5.1.



Obr. 5.1: Výběr šablony projektu.

5.2 Seznam videí

Po spuštění testovací aplikace je třeba zobrazit jednotlivá videa v seznamu. Ta jsou v aplikaci zobrazena ve formě tabulky, přičemž při kliknutí na určitý řádek dojde k inicializaci konkrétního playlistu s požadovaným obsahem. Každý obsah seznamu je specifikován názvem a příslušným obrázkem. Pro zobrazení těchto objektů je nutné tato data nejprve získat. Tato implementace probíhá ve třídě s označením *PlaylistViewController*. Vytvořený seznam lze vidět na obrázku 5.2.



Obr. 5.2: Seznam produktů.

5.2.1 Dekódování seznamu videí

Při spuštění aplikace se tedy nejprve zavolá prvně metoda, která obsahuje parametr pro zadání webové adresy na získání JSON řetězce, na jejímž základě dojde k rozdělení jednotlivých objektů a následnému vykreslení hodnot do tabulky.

Název této metody (i s parametrem) je *decode(products: String)*. Ve funkci dochází k dekódování JSON řetězce pomocí nativní funkce *JSONDecoder().decode(type: Decodable.Protocol, from: Data)*, přičemž do prvního parametru je vložena struktura odpovídající hodnotám z řetězce a do druhého parametru poté data z řetězce. Pokud dekódování řetězce proběhne bez jakýchkoliv problémů, dojde ke vložení již jednotlivých dat do pomocné proměnné, jenž se nazývá *products*. Tato proměnná, která je datového typu pole, je zde z důvodu, aby byl kód přehlednější při další implementaci. V případě, že během dekódování dojde k chybě, je tato chyba vypsána do konzole. Metoda pro dekódování řetězce je znázorněna níže ve výpisu 5.1.

Výpis 5.1: Dekódování řetězce produktu.

```
private func decode(products: String) {
    guard let url = URL(string: products) else {
    return
    }
    do {
        let data = try Data(contentsOf: url)
        let videosParsedObjects = try
        JSONDecoder().decode(Videos.self, from: data)
        print("JSON PARSING WAS SUCCESSFUL")
        self.products?.append(videosParsedObjects)
    } catch {
        print("JSON CONTENT PARSING FAILED",error)
    }
}
```

Jednou z velmi častých chyb, které mohou při dekódování nastat je ta, že určitá část řetězce není ve struktuře zaznamenána. Děje se tak především v případě, kdy je ve struktuře objekt, který se v JSON řetězci nenachází. Dekódér poté neví, jakou hodnotu tomuto objektu přiřadit a vyhodnotí tedy dekódování jako chybné. Stejný případ může nastat i v dalším příkladu použití, kdy je například seznam produktů a jen některé produkty obsahují určitý objekt. Pro ostatní objekty tak opět dekódér neví, co přiřadit. Takový problém lze jednoduše vyřešit tím, že se tento objekt nastaví jako *optional*, tedy libovolný. Struktury pro dekódování produktů jsou znázorněny ve výpisu 5.2.

Výpis 5.2: Struktury pro dekódování produktů.

```
import Foundation

struct Videos: Decodable {
    let categories: [Categories]
}

struct Categories: Decodable {
    let id: String
    let name: [Name]
    let product: [Product]
}

struct Product: Decodable {
    let id: String
    let name: [Name]
    let pictures: Pictures?
    let playlist: String
}

struct Name: Decodable {
    let lang: String
    let text: String
}

struct Pictures: Decodable {
    let resolution: String

    enum CodingKeys : String, CodingKey {
        case resolution = "1920"
    }
}
```

Děkódování do struktur je zde řešeno pomocí protokolu *Decodable*, jenž patří mezi systémové protokoly. Dalším možným řešením je například možné využít knihovny třetí strany, která se nazývá *Alamofire*.

5.2.2 Předání potřebných dat k vytvoření tabulky

Pro nastavení tabulky *UITableView* je potřeba specifikovat hlavní tři metody *UITableViewDataSource*.

Tyto metody se nazývají:

- *numberOfSections()* - Informuje o počtu sekcí.
- *numberOfRowsInSection()* - Vrátí počet řádků v sekcích.
- *cellForRowAt()* - Specifikuje, s jakým druhem buňky se bude pracovat a co bude jejím obsahem.

Metodám lze poskytnout data pouze v případě, že dojde k úspěšnému dekódování dat. V případě, že by se tak z jakéhokoliv důvodu data nepodařilo získat, je možné nastavit výchozí hodnoty a tím se zamezí nechtěnému pádu aplikace. Data jsou zde již vkládána ze zmíněné pomocné proměnné. Naplnění zmíněných metod daty a přidání statických hodnot je znázorněno na obrázku 5.3.

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(PlayerItemTableViewCell.self, forIndexPath: indexPath)
    guard let products = products else { return cell }
    let text = products[0].categories[indexPath.section].product[indexPath.row].name[1].text
    let image = products[0].categories[indexPath.section].product[indexPath.row].pictures?.resolution
    cell.configureCell(title: text, image: image)
    return cell
}

override func numberOfSections(in tableView: UITableView) -> Int {
    return products?[0].categories.count ?? 0
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return products?[0].categories[section].product.count ?? 0
}

override func tableView(_ tableView: UITableView, titleForHeaderInSection section: Int) -> String? {
    return products?[0].categories[section].name[section].text ?? ""
}
```

Obr. 5.3: Metody knihovny pro vytvoření tabulky.

5.2.3 Inicializace přehrávače

Pokud je tabulka nastavena a zobrazuje správně požadovaná data, je možné přejít k implementaci metody pro zobrazení již požadoveného obsahu v přehrávači. Ke zjištění, na který řádek tabulky uživatel klikl, slouží metoda delegátu *tableView(tableView: UITableView, didSelectRowAt indexPath: IndexPath)*. Při stisknutí určitého řádku dojde k volání metody knihovny *KevPlayer*, jenž již byla zmíněna v popisu knihovny. Jak už bylo řečeno, do parametru metody se vloží odkaz na požadovaný seznam videí, který je aplikaci vrácen po případném úspěšném dekódování. Se získanými daty poté dochází k inicializaci přehrávače, jenž je zde v testovacím

projektu implementován do samostatné třídy *PlayerViewController()*, která je popsána v další sekci. Funkce obsahuje velké množství řádku kódu, což je způsobené velkým počtem požadovaných objektů pro vytvoření přehrávače. V ideálním případě by bylo vhodné aktuální metodu rozdělit na více funkcí. V případě testovacího projektu je však možné ignorovat velikost funkce. Metodu pro zobrazení přehrávače s požadovanými daty je možné vidět na obrázku 5.4.

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    guard let playlist = playlist else { return }
    guard let products = products else { return }
    playlistContent = playlist.decodePlaylist(JSON: products[0].categories[indexPath.section].product[indexPath.row].playlist)

    guard let streamInfos = playlistContent[0].streamInfos else { return }
    for urlLink in streamInfos {
        urlLinks.append(urlLink.url)
    }

    if urlLinks[0].contains("/ao.mp4") {
        urlLinks.remove(at: 0)
    }

    var definition = [KevPlayerResourceDefinition]()
    var startingNumber = -1

    for urlDefinition in urlLinks {
        startingNumber = startingNumber + 1
        if let urlDefinition = URL(string: urlDefinition) {
            definition.append(KevPlayerResourceDefinition(url: urlDefinition, definition: "\(startingNumber)"))
        }
    }
    let name = products[0].categories[indexPath.section].product[indexPath.row].name[1].text
    print(name)
    var subtitle: String?
    if let subInfos = playlistContent[0].subInfos {
    let subtitles = subInfos[0].subtitle
        subtitle = subtitles
    }
    let isAdsEnabled = playlistContent[0].adsEnabled
    let cover = products[0].categories[indexPath.section].product[indexPath.row].pictures?.resolution

    navigationController?.pushViewController(PlayerViewController(url: nil, name: name, subtitles: subtitle, isAdEnabled: isAdsEnabled!, definition: definition,
        isOnline: true, cover: cover), animated: true)
    urlLinks.removeAll()
    tableView.deselectRow(at: indexPath, animated: true)
}
```

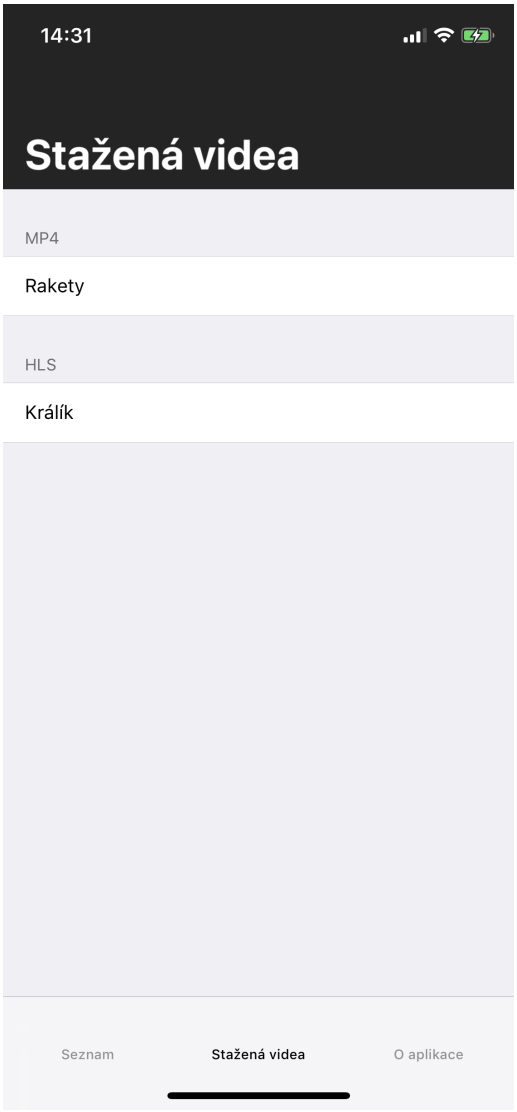
Obr. 5.4: Metoda pro zobrazení přehrávače.

5.3 Přehrávač

Jedná se o třídu *PlayerViewController*, kde je implementována vyvinutá knihovna. Po inicializaci přehrávače se správným předáním parametrů dochází k volání metody knihovny *player.setVideo(resource: KevPlayerResource)*, která tyto parametry opět předá příslušným objektům a dojde k zobrazení obsahu. Před samotným spuštěním videa je také možné specifikovat, co a jak bude zobrazeno či spuštěno. V této třídě je možné přehrávač zobrazit dvěma způsoby, přičemž záleží, zda se jedná o online či stažený obsah (offline). Tento fakt je třídě předáván pomocí parametru *isOnline: Bool*. Rozdělení je z důvodu výběru kvalit, kdy u online přenosu je možné kvalitu specifikovat a u staženého záleží na staženém obsahu.

5.4 Stažený obsah

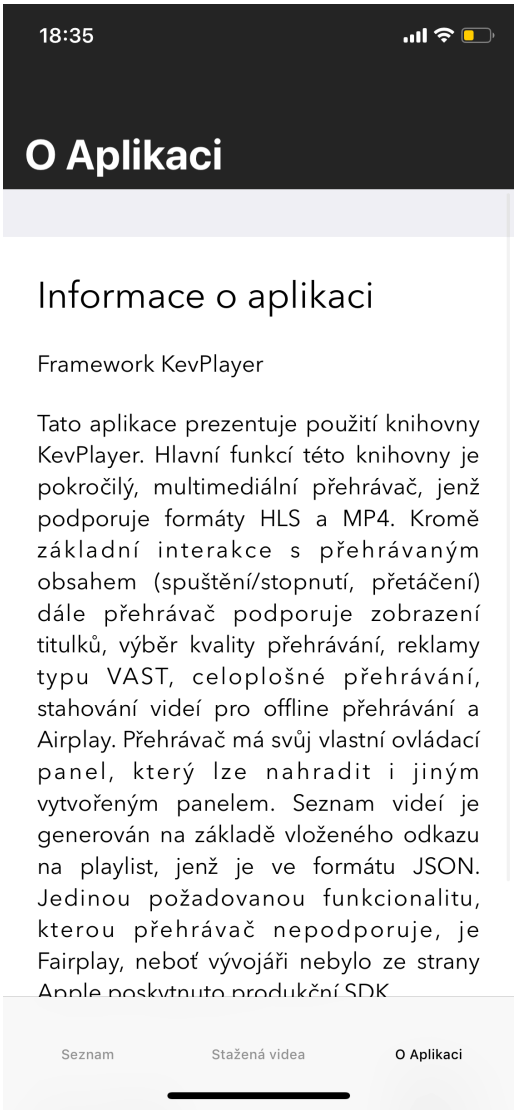
Tato třída, jenž se nazývá *OfflinePlaylistViewController*, opět zobrazuje tabulku obsahující seznam stažených videí. Při stisknutí na určitý řádek dojde k inicializaci přehrávače a přehrávání obsahu na základě jména videa. Obsah je také možné mazat, čímž samozřejmě také dojde k vymázání videa z úložiště. Vytvořenou tabulku lze vidět na obrázku 5.5.



Obr. 5.5: Seznam stažených produktů.

5.5 O Aplikaci

Třída, jejíž pojmenování je *AboutViewController*, je opět tvořena tabulkou, která obsahuje několik různých typů buněk. V těchto buňkách jsou stručně popsány funkcionality knihovny a také návod, jak tuto testovací aplikaci ovládat. Obrazovku je možné si prohlédnout na obrázku 5.6.



Obr. 5.6: O aplikaci.

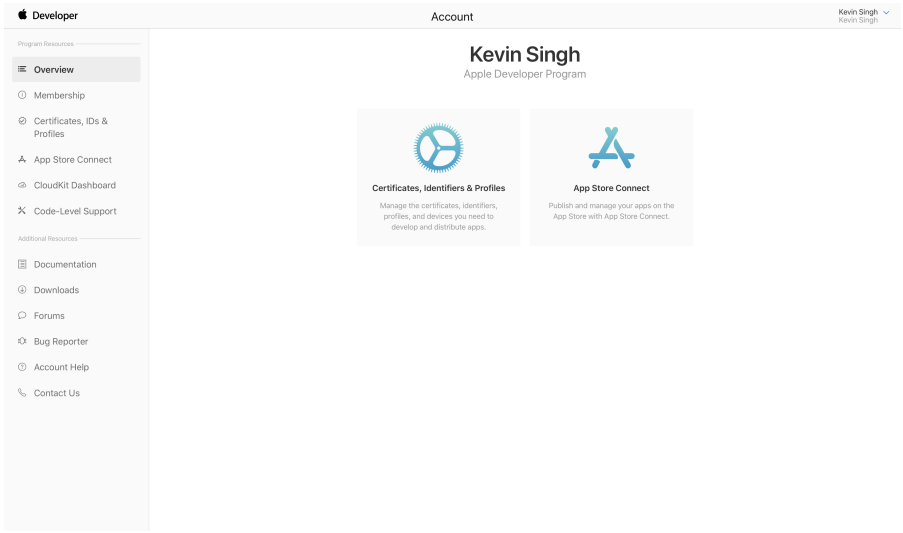
6 Nezbytné kroky pro publikaci

6.1 Publikace aplikace do obchodu

Poslední fáze vývoje aplikace je distribuce uživatelům se systémem iOS pomocí virtuálního obchodu *AppStore*. Pokud je aplikace bez chyb a neobsahuje žádná vážná varování, je možné ji publikovat. Tím je umožněno zpřístupnění vyvíjené aplikace co nejvíce lidem, kteří si ji mohou otestovat a vyzkoušet tak, co vše přehrávač umí. Před samotným vložením aplikace je však nutné provést několik zásadních kroků.

6.2 Vývojářský účet

Ze všeho nejdříve je nutné si vytvořit vývojářský účet či *Apple ID* na vývojářských stránkách společnosti *Apple* <`http://apple.developer.com`>. Zde má vývojář přístup k podrobnější dokumentaci, správě svých aplikací, fóru či možnost stáhnout beta verze nových operačních systémů.

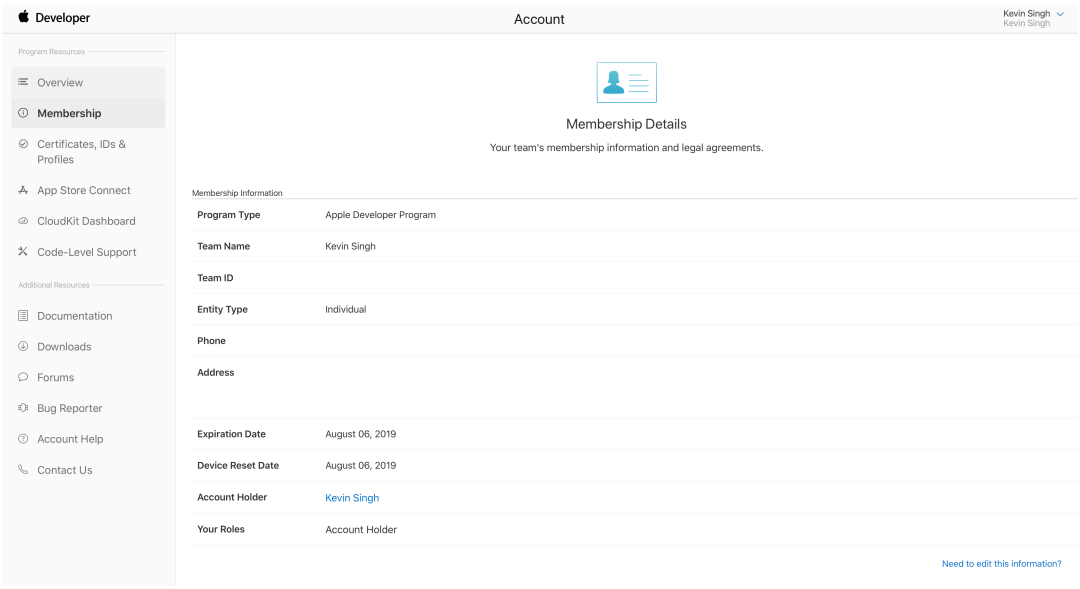


Obr. 6.1: Vývojářský účet.

6.3 Vývojářská licence

V rámci vývojářského účtu je také možnost zakoupit vývojářskou licenci sloužící k publikaci a správě aplikací v obchodě. Cena této licence pro samostatné vývojáře činí pro rok 2019 99 USD, přičemž její platnost je omezena po dobu jednoho roku. Po vypršení této doby je nutné licenci opět zakoupit. Cena pro firmy je 299 USD.

Pokud by se vývojář rozhodl, že obnovení licence nezaplatí, veškeré jeho aplikace budou v obchodě *AppStore* skryty (aplikace žádní uživatelé neuvidí a nebudou si je moci tak stáhnout) do té doby, dokud svoji licenci opět neprodlouží.



Obr. 6.2: Vývojářská licence.

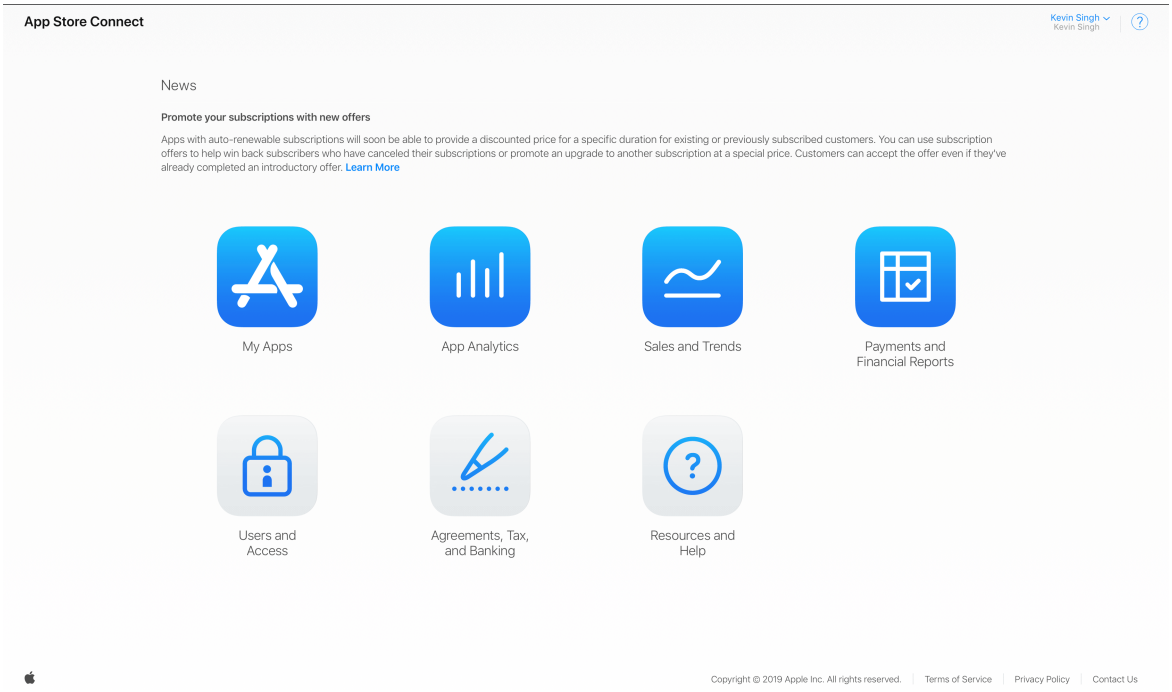
6.4 App Store Connect

Předchozí kroky se zaměřovaly na nastavení účtu jako takového. Pro publikaci aplikací je však nezbytné přejít na stránky <`https://appstoreconnect.apple.com`>. Zde je možné nahrávat a spravovat aplikace vývojáře a také vidět statistiky a analýzy prodeje či používání aplikací apod.

Po rozkliknutí jednotlivých aplikací je možné upravovat jejich náhled a popis v obchodě *AppStore*, sledovat stav (zda jsou či nejsou povolené k publikaci) a také například upravit cenu pro stáhnutí, podporované regiony či ji zcela z obchodu odstranit. Zde je také nutné nastavit název aplikace, popis, ikonu, snímky z aplikace (screenshots) a klíčová slova pro vyhledání aplikace. Tato slova jsou omezena na 100 znaků. Je tedy potřeba si důkladně promyslet, která vyhledávácí slova budou co nejvýstižnější.

Velmi důležitou funkcionalitou je také možnost pozvat testery či další vývojáře k otestování připravované aplikace a získat tak podstatnou zpětnou vazbu. Možné je pozvat až 10 000 testerů, přičemž testování konkrétní verze aplikace je časově limitováno na 90 dní. Po této době je potřeba vydat novější verzi (tzv. nový build), čímž se čas obnoví.

Testeři si mohou aplikace, kam byli k testování pozvání, stáhnout potomocí aplikace *Testflight*, která je zdarma ke stažení.



Obr. 6.3: Prostředí App Store Connect.

6.5 Recenze aplikace

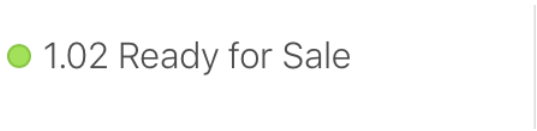
Pokud je vývojář s případným testováním spokojen či by již chtěl aplikaci publikovat, může aplikaci zaslat k publikaci do obchodu. To však neznamená, že aplikace bude úspěšně publikována.

Před samotnou publikací je aplikace ještě testována inženýry ze společnosti *Apple*. To především z důvodu, aby aplikace například neobsahovala nepovolené knihovny, nevykonávala nevyžádané funkce apod. Testování neprobíhá pouze z pohledu funkcí, ale také například z uživatelského prostředí, kde se kupříkladu testuje, zda jsou všechna tlačítka viditelná i na tom nejmenším/největším podporovaném zařízení. Zároveň se také bere důraz na žádosti o využívání různých komponent, jako mohou být fotoaparát, reproduktor či mikrofon. Pokud vývojář nespecifikuje, z jakého konkrétního důvodu žádá přístup k těmto technologiím, je to špatně.

Délka recenze závisí na složitosti testované aplikace, ale průměrně se jedná o tři až pět dnů. Jsou však i případy, kdy recenze může trvat déle než dva týdny.

V případě, že testování dopadne negativně, je vývojáři odesláno oznámení o negativním výsledku obsahující chyby k opravení. Tímto se může vydání aplikace do

obchodu znatelně prodloužit. V opačném případě, kdy je vše v pořádku, je možné aplikaci ihned publikovat. Publikovaná aplikace se v AppStore objeví do dvaceti čtyř hodin.



Obr. 6.4: Verze aplikace + schválený stav po testování.

6.6 Design aplikace

Uživatelské rozhraní by se mělo držet tzv. *Apple Guidelines*, ve kterých firma Apple přímo specifikuje a ukazuje, jak se mají vyvíjené aplikace správně chovat. V *AppStore* je však možné nalézt spoustu aplikací, které se těchto konvencí nedrží. Do jisté míry je tedy možné použít i vlastní design. Vždy však velmi záleží na recenzentovi aplikace, zda takovou aplikaci schválí. Z hlediska ovládání aplikace je však doporučené, aby se vývojář těchto norem držel, což budoucím uživatelům aplikace hlavně zjednoduší její ovládání.

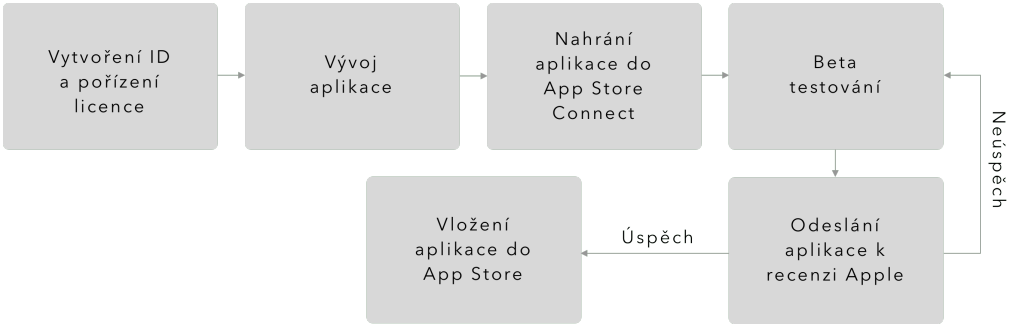
6.7 tvOS

U platformy tvOS nejsou žádné velké rozdíly oproti jiným platformám. Zde však platí, že veškeré knihovny musí být vyrobeny pro systém tvOS, což se vztahuje i na frameworky třetích stran. Je také důležité se ujistit, že aplikace pro tvOS se neodkazuje na knihovny postavené pro systém iOS.

Hlavní svazek (*bundle*) aplikace, kterou chce vývojář vložit do obchodu, může obsahovat 4 GB a až 20 GB zdrojů na vyžádání.

6.8 Proces znázorněný diagramem

Níže na obrázku 6.5 je celý proces publikace aplikace znázorněn pomocí diagramu.



Obr. 6.5: Vložení aplikace do App Store.

7 Závěr

V diplomové práci byl vyvinut multimediální přehrávač pro iOS, jenž slouží pro interaktivní přehrávání audiosouborů či videosouborů.

Nejprve se práce zabývala popisem formátů, které jsou knihovnou podporovány. Konkrétněji se jedná o MP4, HLS a MPEG-Transport Stream. Mezi popis formátů také patří MPEG-DASH, ten ale není v knihovně zahrnut, neboť systémy firmy *Apple* tento formát nepodporují. Dále je zde k nalezení popis způsobu zabezpečení multimediálního obsahu DRM, ke kterému jsou uvedeny i tři typy - *FairPlay*, *PlayReady* a *Widevine*. Poté se práce zabývala popisem dalšího typu zabezpečení nazývané AES, analytického nástroje *Google Analytics* a reklamními formáty VAST, VPAID a VMAP. Dále byl proveden rozbor dostupných řešení a vytvořen návrh knihovny přehrávače. Vybráno bylo řešení pomocí *AVFoundation*, protože jako jediné není založené na webové technologii. Z tohoto důvodu tedy probíhal vývoj zcela od začátku a to pouze v nativním programovacím jazyce *Swift*.

Poté se již práce zaměřovala na samotný vývoj knihovny. Popis vývoje začíná nastavením projektu ve vývojovém prostředí *Xcode* a instalací manažera závislostí knihoven třetích stran *CocoaPods*. Následně pokračuje nastavením analytického nástroje *Google Analytics* a implementací inicializace přehrávače. Zde je také vysvětleno, jaké parametry přehrávač očekává.

Následně se práce zabývala popisem ovládacího panelu, s jehož pomocí je možné ovládat veškeré funkcionality přehrávače. Mezi tyto komponenty patří jméno přehrávaného obsahu, aktuální a celkový čas přehrávání, posuvník a několik tlačítek. Tato tlačítka se používají pro ovládání titulků, stáhnutí obsahu, přepínání kvality, aktivace *AirPlay*, ztišení, přetočení přehrávače z vertikální či horizontální polohy a přepínání přehrávaného obsahu.

Dále se již práce zabývala popisem vývoje jednotlivých funkcionalit, což je doplněno i ukázkami kódu.

Po popisu vývoje samotné knihovny byla také vytvořena testovací aplikace, do které byla implementována vyvinutá knihovna. Ta zobrazuje seznam videí, stažených položek a informace o vývoji. Všechny obrazovky aplikace i vzhled přehrávače s různými stavy je možné vidět v příloze A.

Na závěr byl také zpracován popis postupu pro vložení aplikace do virtuálního obchodu *App Store*.

Knihovna byla otestována a je plně funkční. Jediné, co se nepodařilo zprovoznit je přehrávání DRM, neboť nebyl zpřístupněn certifikát *FairPlay* firmou *Apple*. Ten byl potřebný pro výměnu potřebných klíčů. Princip, jak by ale tato funkcionalita mohla být zprovozněna, je v textu znázorněn.

Literatura

[1] *Video | MPEG. MPEG | The Moving Picture Experts Group website* [online]. Dostupné z: <https://mpeg.chiariglione.org/standards/mpeg-4/video>.

[2] *MP4 Audio File Format Description. CoolUtils File Converters* [online]. Copyright © 1998 [cit. 24.11.2018]. Dostupné z: <https://www.coolutils.com/Formats/MP4>.

[3] */ MPEG. MPEG | The Moving Picture Experts Group website* [online]. Dostupné z: <https://mpeg.chiariglione.org/standards/mpeg-4/audio>.

[4] *Create PDF, Extract text from PDF, Generate Barcodes, Read Barcodes .NET and ASP.NET components - ByteScout* [online]. Copyright © Copyright 2016 [cit. 24.11.2018]. Dostupné z: <https://bytescout.com/blog/2014/09/mp4-file-format.html>.

[5] *Apple Developer Documentation. 401 Authorization Required* [online]. Copyright © 2018 Apple Inc. All rights reserved. [cit. 24.11.2018]. Dostupné z: <https://developer.apple.com/documentation/http_live_streaming>.

[6] *Apple Developer Documentation. 401 Authorization Required* [online]. Copyright © 2018 Apple Inc. All rights reserved. [cit. 24.11.2018]. Dostupné z: <https://developer.apple.com/documentation/http_live_streaming/understanding_the_http_live_streaming_architecture>.

[7] *Live Streaming Solutions and Video Hosting Platform - DaCast* [online]. Copyright © 2018 DaCast, All rights reserved. [cit. 24.11.2018]. Dostupné z: <https://www.dacast.com/blog/http-live-streaming/>.

[8] *Apple Developer Documentation. 401 Authorization Required* [online]. Copyright © 2018 Apple Inc. All rights reserved. [cit. 24.11.2018]. Dostupné z: <https://developer.apple.com/documentation/http_live_streaming/deploying_a_basic_http_live_stream>.

[9] */ MPEG. MPEG | The Moving Picture Experts Group website* [online]. Dostupné z: <https://mpeg.chiariglione.org/standards/mpeg-dash/media-presentation-description-and-segment-formats>.

[10] *Encoding.com. Enterprise Media Processing | Encoding.com* [online]. Dostupné z: <https://www.encoding.com/mpeg-dash/>.

[11] *MPEG-DASH browser support and device compatibility* [online]. Copyright © Bitmovin 2019. All Rights Reserved. [cit. 25.04.2019]. Dostupné z: <https://bitmovin.com/mpeg-dash-browser-support-device-compatibility/>.

[12] *ResearchGate | Share and discover research* [online]. Copyright © ResearchGate 2018. All rights reserved. [cit. 24.11.2018]. Dostupné z: <https://www.researchgate.net/publication/283072319_Enhancing_MPEG_DASH_performance_via_server_and_network_assistance>.

[13] *| Digitální knihovna VUT v Brně. Domovská stránka repozitáře | Digitální knihovna VUT v Brně* [online]. Copyright © [cit. 24.11.2018]. Dostupné z: <https://dspace.vutbr.cz/xmlui/handle/11012/55046>.

[14] *Headend Information, IPTV Headend and IP Headend* [online]. Dostupné z: <https://www.headendinfo.com/ts-transport-stream-mpts-spts/>.

[15] *AfterDawn - Software downloads, reviews, tech news and guides* [online]. Copyright © 1999 [cit. 24.11.2018]. Dostupné z: <https://www.afterdawn.com/glossary/term.cfm/mpeg2_transport_stream>.

[16] *Digital rights management: Model, technology and application* [online].[cit. 24.11.2018]. Dostupné z: <https://ieeexplore.ieee.org/document/7961371>.

[17] *FairPlay Streaming* [online]. Dostupné z: <https://developer.apple.com/streaming/fps/>.

[18] *Microsoft Corporation PlayReady* [online]. Copyright © 2018 Microsoft. All Rights Reserved. [cit. 24.11.2018]. Dostupné z: <https://www.microsoft.com/playready/overview/>.

[19] *Widevine* [online]. Dostupné z: <https://www.widevine.com>.

[20] *Study of the AES Realization Method on the Reconfigurable Hardware - IEEE Conference Publication.* [online].[cit. 24.11.2018]. Dostupné z: <https://ieeexplore.ieee.org/document/6835549>.

[21] *Adaptive streaming video infrastructure encoding and HTML5 player* [online]. Copyright © [cit. 24.11.2018]. Dostupné z: <https://bitmovin.com/docs/encoding/faqs/what-is-hls-aes-encryption>.

[22] *Google Analytics | Google Developers. Google Developers* [online]. Dostupné z: <https://developers.google.com/analytics/>.

[23] *IAB* [online]. Dostupné z: <https://www.iab.com/guidelines/digital-video-ad-serving-template-vast-3-0/>.

[24] *IAB* [online]. Dostupné z: <https://www.iab.com/guidelines/digital-video-ad-serving-template-vast/>.

[25] *IAB* [online]. Dostupné z: <https://www.iab.com/guidelines/digital-video-player-ad-interface-definition-vpaid-2-0/>.

[26] *IAB* [online]. Dostupné z: <https://www.iab.com/guidelines/digital-video-multiple-ad-playlist-vmap-1-0-1/>.

[27] *Kaltura Video Platform - Powering Any Video Experience. Kaltura Video Platform - Powering Any Video Experience* [online] Copyright © 2018 Inc. All rights reserved. [cit. 09.12.2018]. Dostupné z: <https://corp.kaltura.com/>.

[28] *Adaptive streaming video infrastructure encoding and HTML5 player. Adaptive streaming video infrastructure encoding and HTML5 player* [online]. Copyright © Bitmovin 2018. All Rights Reserved. [cit. 09.12.2018]. Dostupné z: <https://bitmovin.com/>.

[29] *Brightcove | The Leading Online Video Platform | Video Hosting.* [online]. Copyright © 2018 Brightcove Inc. All rights reserved. [cit. 09.12.2018]. Dostupné z: <https://www.brightcove.com/en/>.

[30] *AVFoundation* [online]. Dostupné z: <https://developer.apple.com/av-foundation/>.

[31] *AVAsset - AVFoundation | Apple Developer Documentation.* [online]. Dostupné z: <https://developer.apple.com/documentation/avfoundation/avasset>.

[32] *Apple | Time and Media Representations.* [online]. Dostupné z: <https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/06_MediaRepresentations.html#//apple_ref/doc/uid/TP40010188-CH2-SW1>.

[33] *Google Developers* [online]. Dostupné z: <https://developers.google.com/interactive-media-ads/docs/sdks/ios/>.

[34] *How to Read Ad Tags and Understand Ad Tag Variables* [online]. Dostupné z: <https://www.adopsinsider.com/ad-ops-basics/what-are-ad-tags/>.

Seznam symbolů, veličin a zkratek

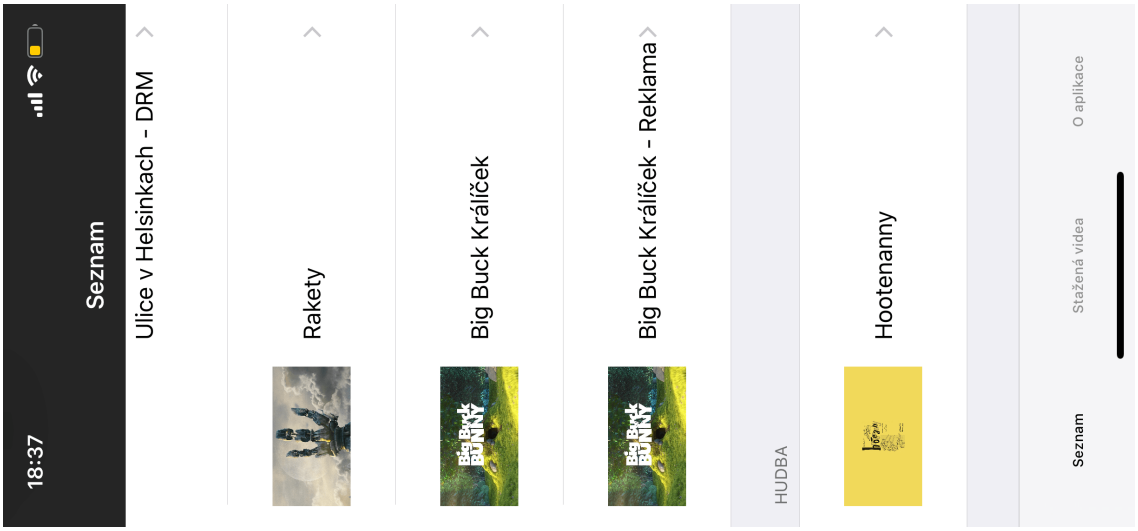
AAC	Advanced Audio Coding
CD	Kompaktní disk
DCT	Diskrétní cosínova transformace
DES	Data Encryption Standard
DRM	Digital Rights Management
DVB	Digital Video Broadcasting
HLS	HTTP Live Streaming
MP4	MPEG-4
MPEG-TS	MPEG-Transport Stream
QoS	Quality of Service
VAST	Video Ad Serving Template
VOD	Video on Demand
VMAP	Video Multiple Ad Playlist
VPAID	Video Player-AD API definition

Seznam příloh

A Všechny obrazovky knihovny a testovací aplikace

V následující příloze je možné vidět všechny vytvořené obrazovky testovací aplikace, ve které je vytvořená knihovna implementována.

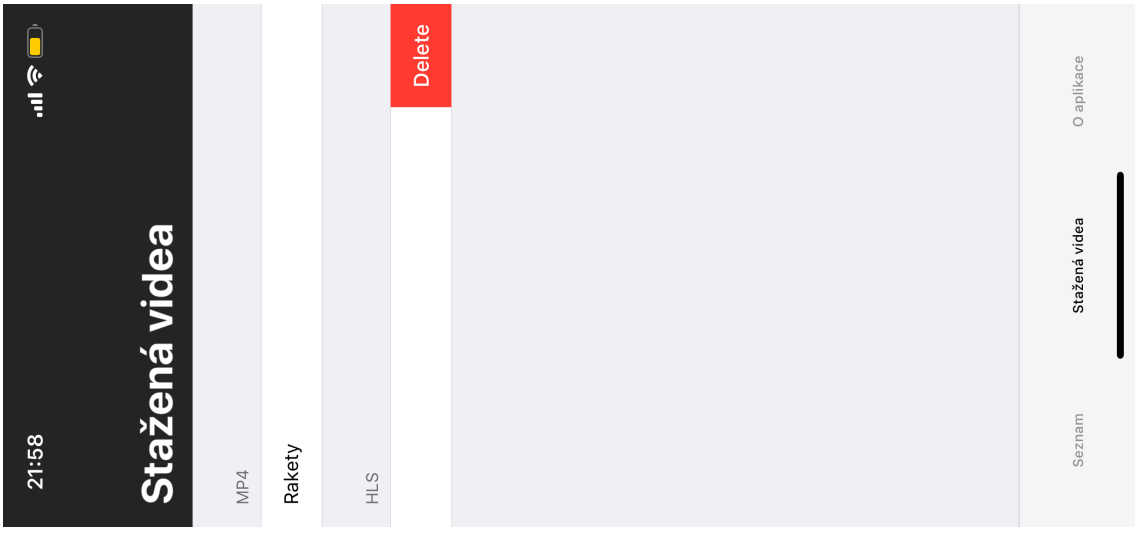
Obr. A.1: Sekce videa.



Obr. A.2: Sekce hudba.

Obr. A.3: Seznam stažených videí.

Stažená videa jsou rozdělena na základě formátu. Aktuálně je zde rozdělení pro MP4 a HLS.



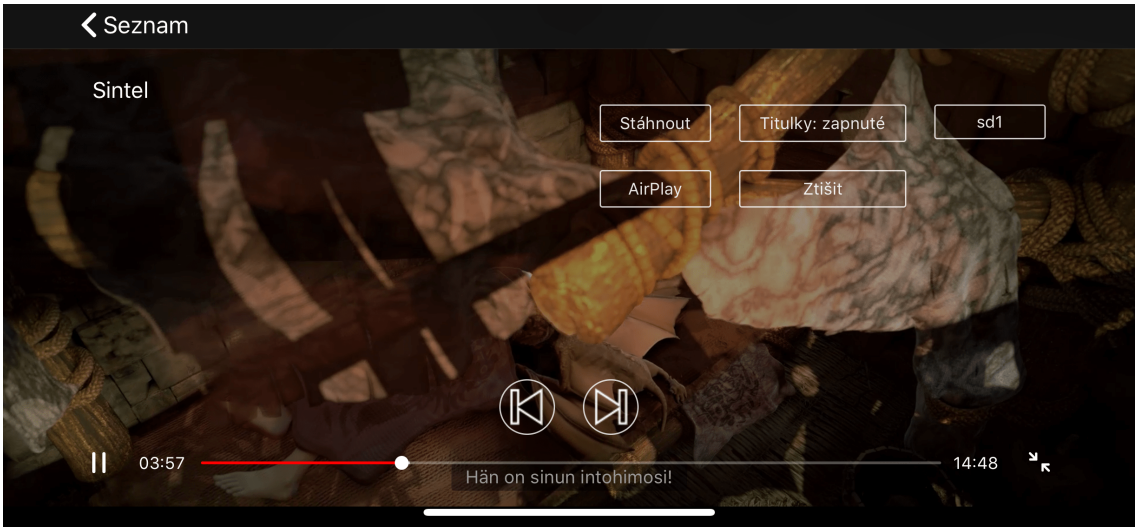
Obr. A.4: Ukázka smazání uloženého videa.

Animace tlačítka pro smazání se objeví po posunutí vybraného řádku směrem doleva.

Obr. A.5: Záložka informací o aplikaci.

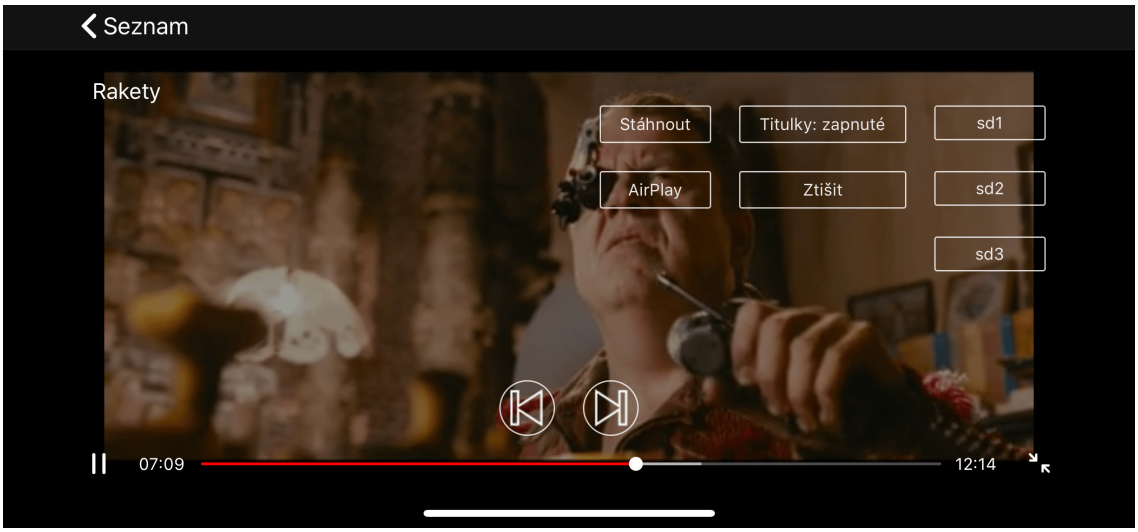
Obr. A.6: Posunutí sekce informace o aplikaci.

Obr. A.7: Zobrazení přehrávače ve vertikální poloze.



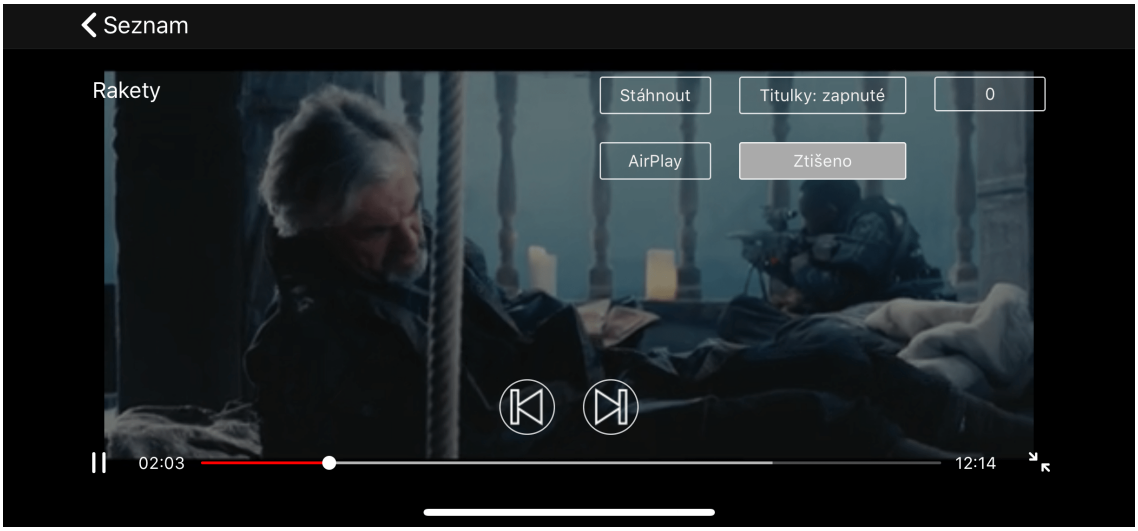
Obr. A.8: Zobrazení přehrávače v horizontální poloze.

Jak si lze všimnout, v testovací aplikaci je možné vidět výběr kvality jen v horizontální poloze.

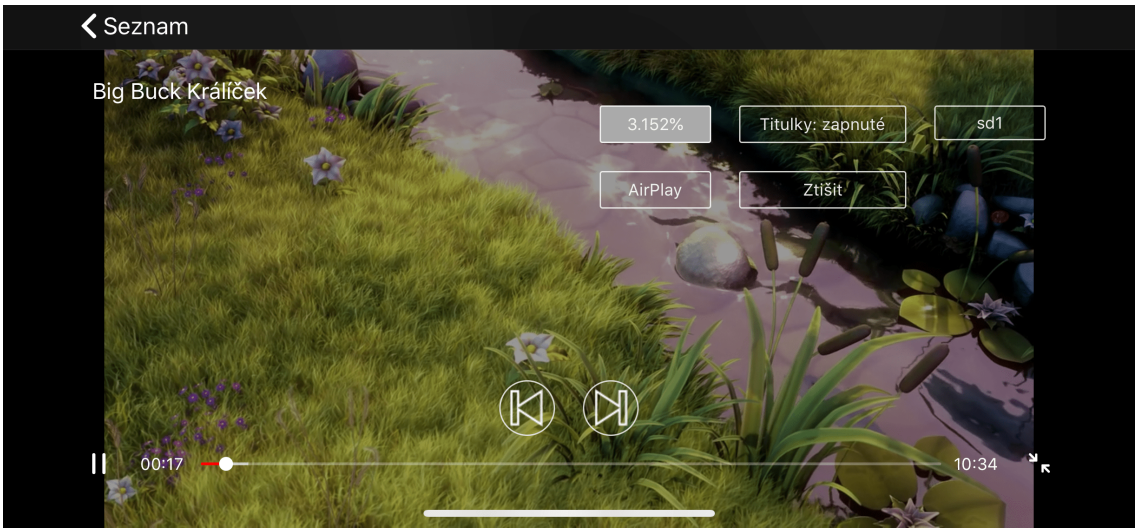


Obr. A.9: Rozkliknutí výběru kvality.

Prvotně lze vidět pouze jedno tlačítko, zde s číslem 0. Po rozkliknutí tohoto tlačítka dojde k zobrazení ostatních tlačítek pomocí animace.



Obr. A.10: Zobrazení stisknutého tlačítka.

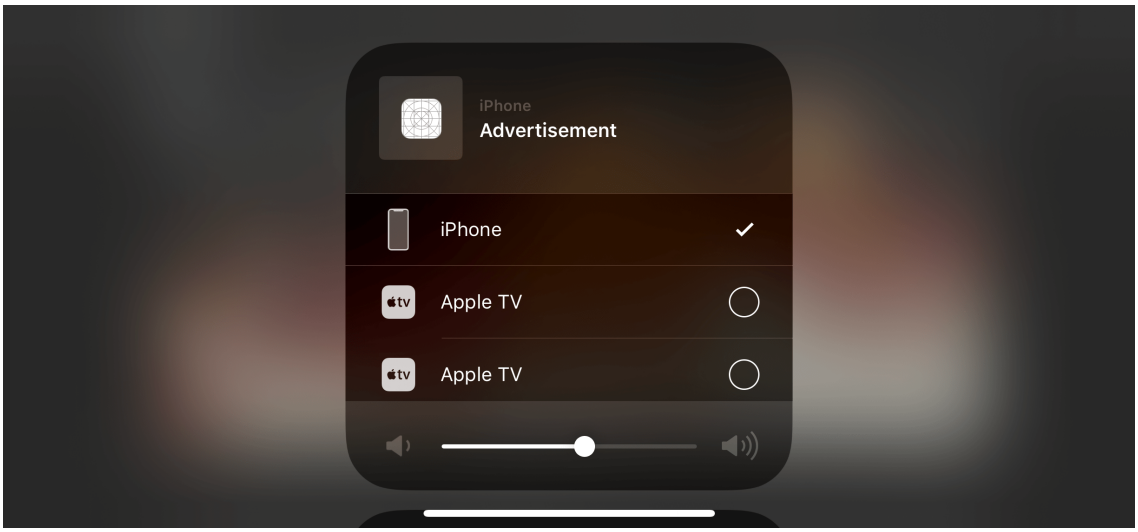


Obr. A.11: Zobrazení procesu stahování.

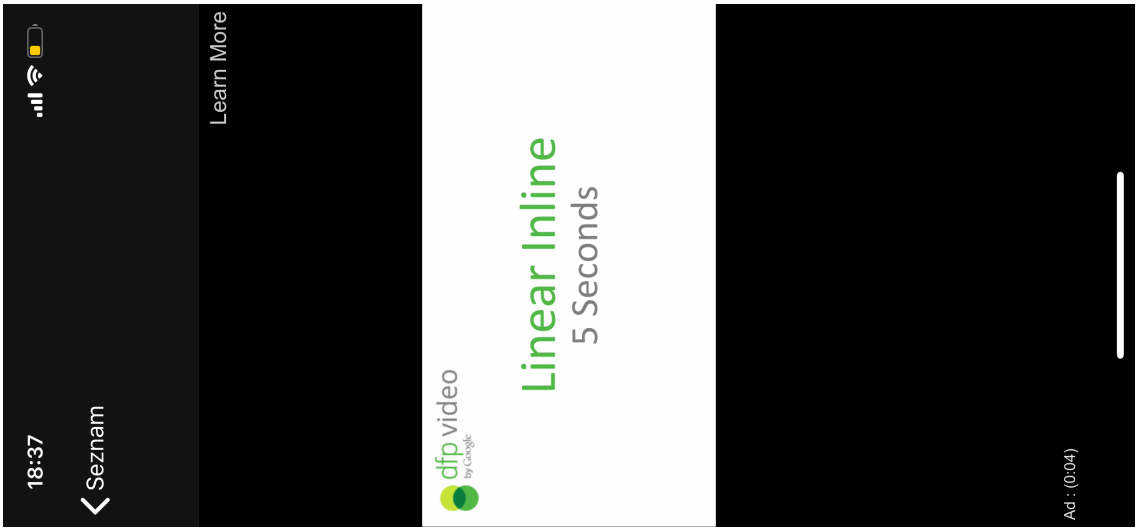
Po dokončení stažení dojde zpět ke změně pozadí tlačítka (průhledné).

Obr. A.12: Zobrazení nabídky AirPlay ve vertikální poloze.

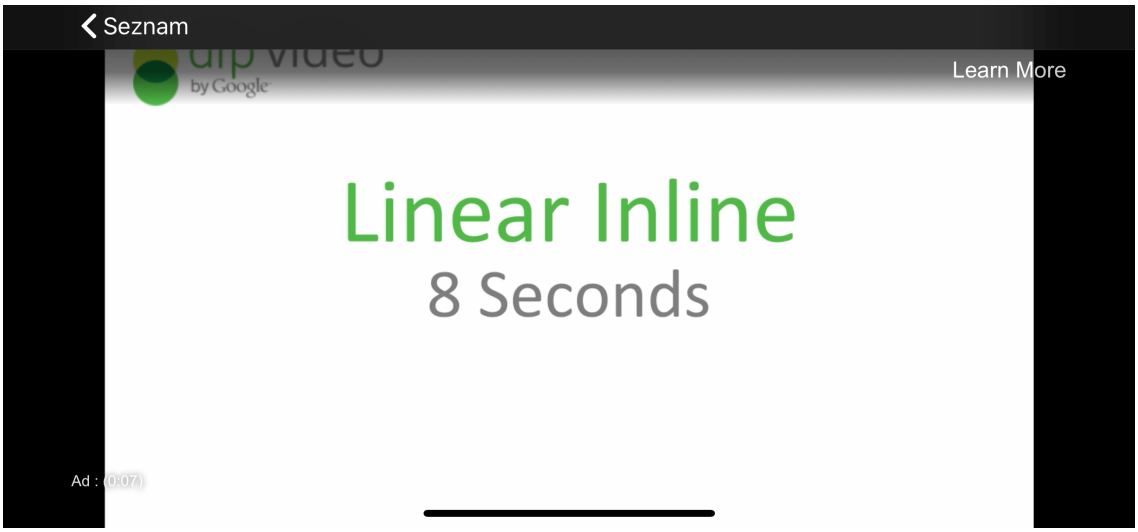
Po stisknutí na tlačítko AirPlay dojde k zobrazení následující nabídky, která je zobrazena na obrázku A.12.



Obr. A.13: Zobrazení nabídky AirPlay v horizontální poloze.

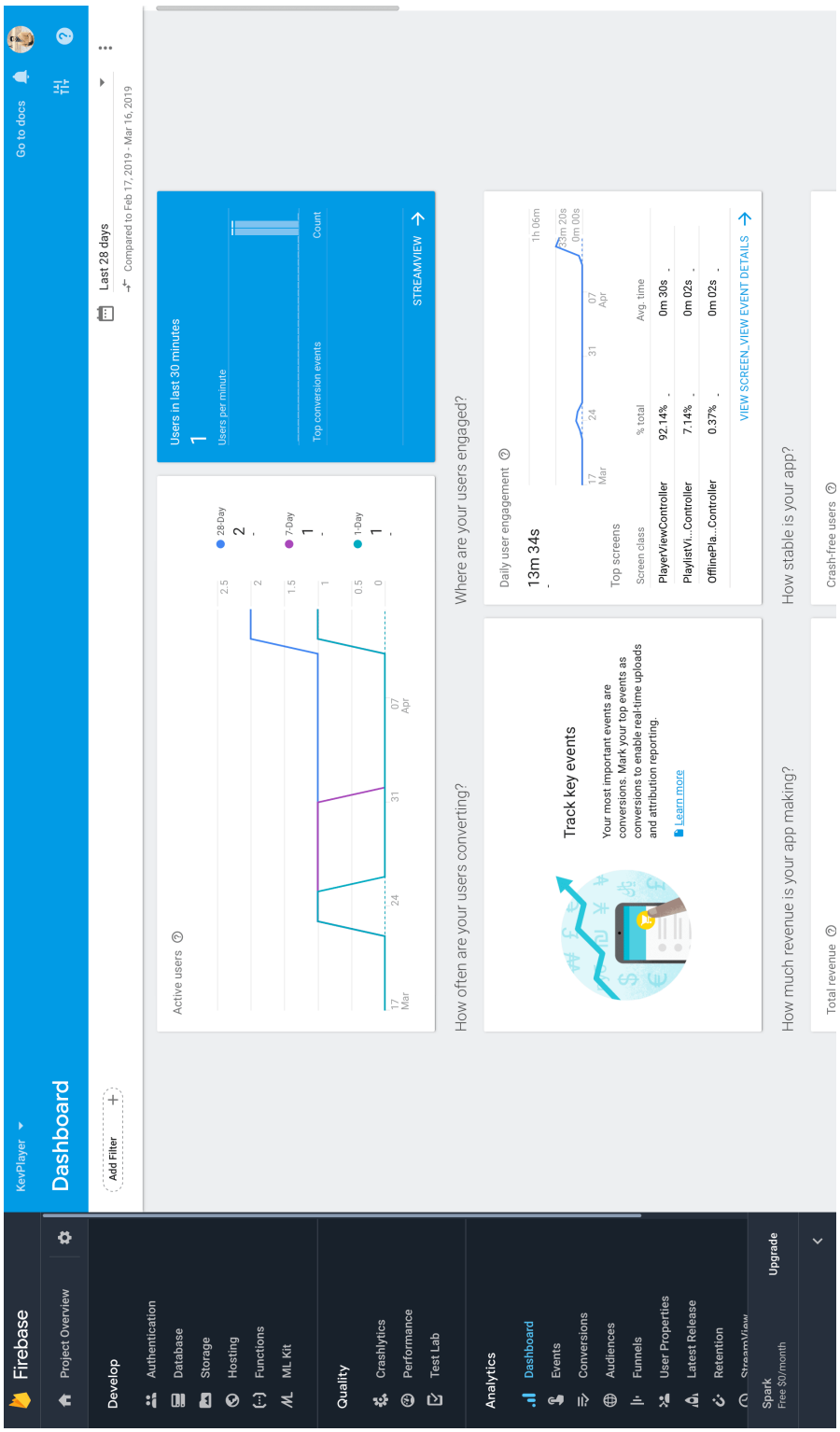


Obr. A.14: Zobrazení reklamy ve vertikální poloze.

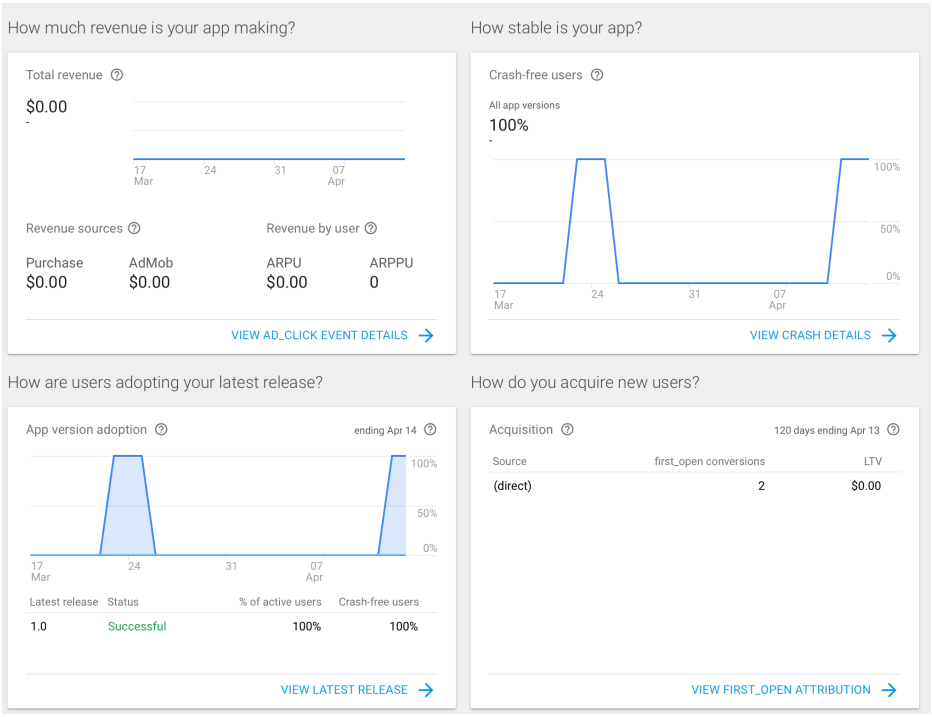


Obr. A.15: Zobrazení reklamy v horizontální poloze.

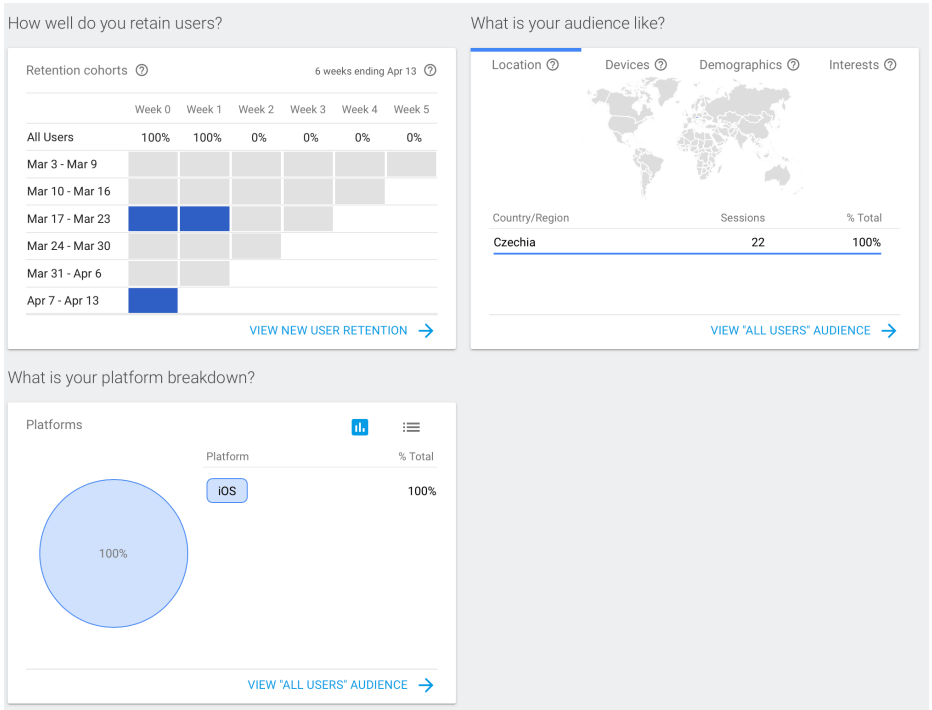
B Statistiky z Google Analytics



Obr. B.1: Hlavní obrazovka Google Analytics.

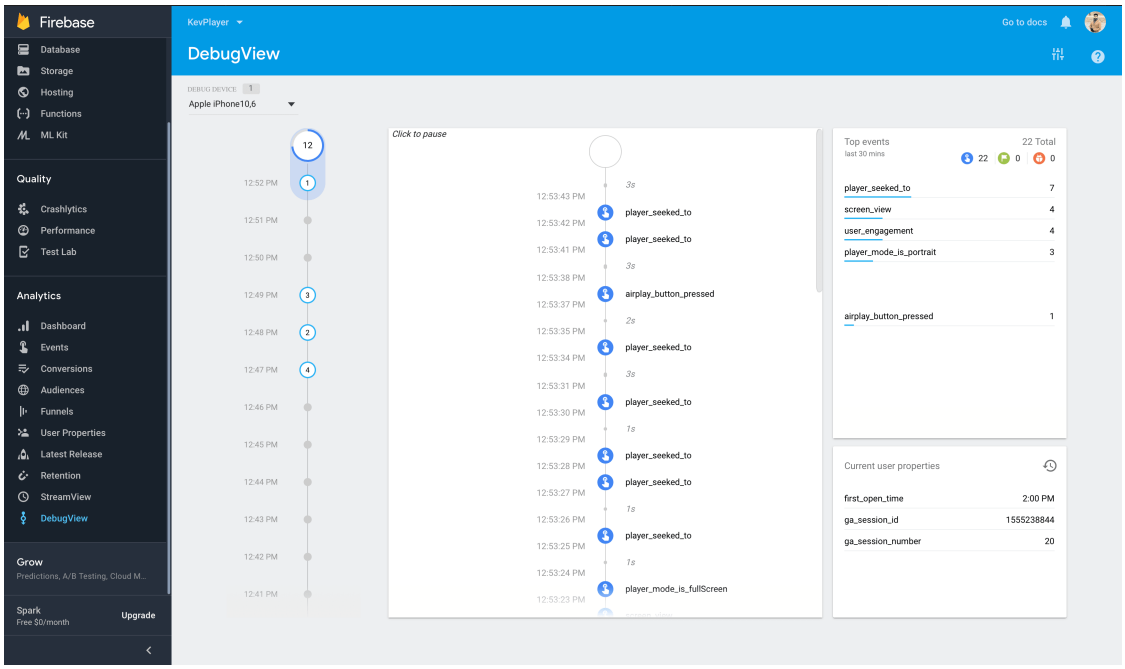


Obr. B.2: Statistiky na hlavní obrazovce.

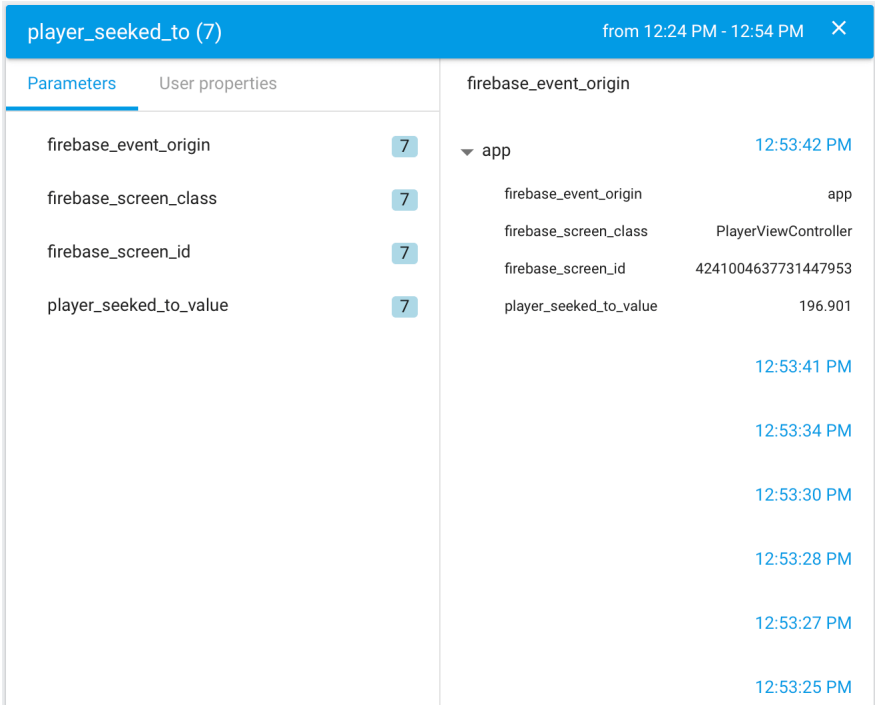


Obr. B.3: Další statistiky.

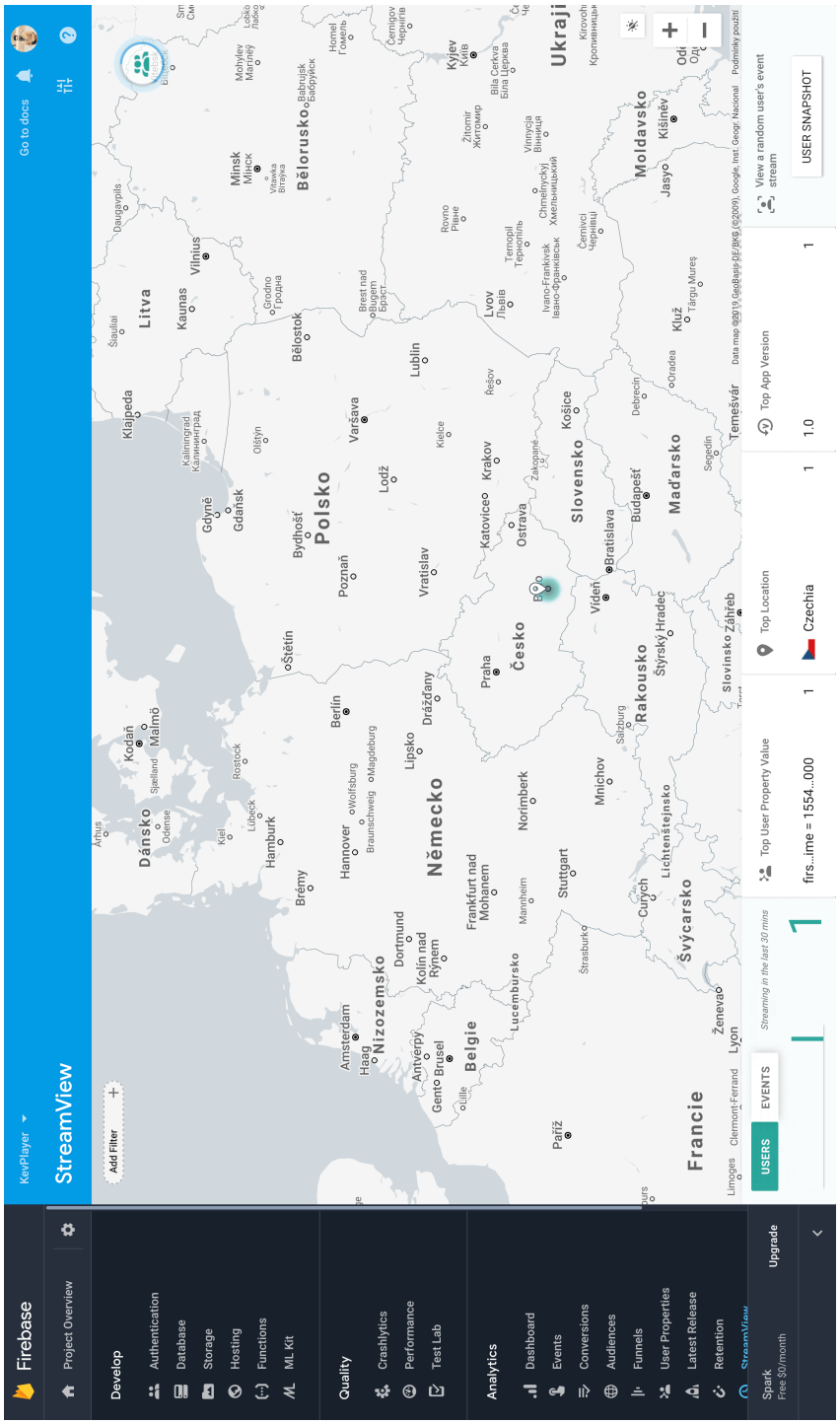
Obr. B.4: Výpis a četnost používání logů.



Obr. B.5: Zobrazení logů v debug módu.



Obr. B.6: Zobrazení detailu parametru.



Obr. B.7: Aktuální používání aplikace v reálném čase.

C Obsah přiloženého CD

K diplomové práci je také přiloženo elektronické médium ve formě CD nacházející se na zadní části desek. Na médiu je možné nalézt elektronickou verzi diplomové práce, zdrojové kódy knihovny a také testovací aplikace, které byly vytvořeny ve vývojovém prostředí *Xcode* v jazyce *Swift*.

- `/` ..... kořenový adresář přiloženého CD
  - `diplomova-prace-Kevin-Singh` ..... elektronická verze diplomové práce
  - `KevPlayer` ..... Adresář zdrojových kódů vyvinuté knihovny
  - `testKevPlayerApp` ..... Adresář testovací aplikace

D Návod ke spuštění testovací aplikace

Pro spuštění testovací aplikace je nutné otevřít soubor pojmenován *testKevPlayerApp.xcworkspace* ve vývojovém prostředí Xcode. Soubor je k nalezení na přiloženém CD ve složce testovaciAplikace. Důležité je také poznamenat, že vývojové prostředí Xcode je pouze pro zařízení s operačním systémem macOS.

Pokud je vývojovým prostředím vyžadováno, je potřebné nastavit vývojářský účet v nastavení projektu. To lze nastavit přechodem do nastavení projektu, kde v sekci *Signing* je možné zvolit účet. Nastavení je znázorněné na obrázku D.1 níže.

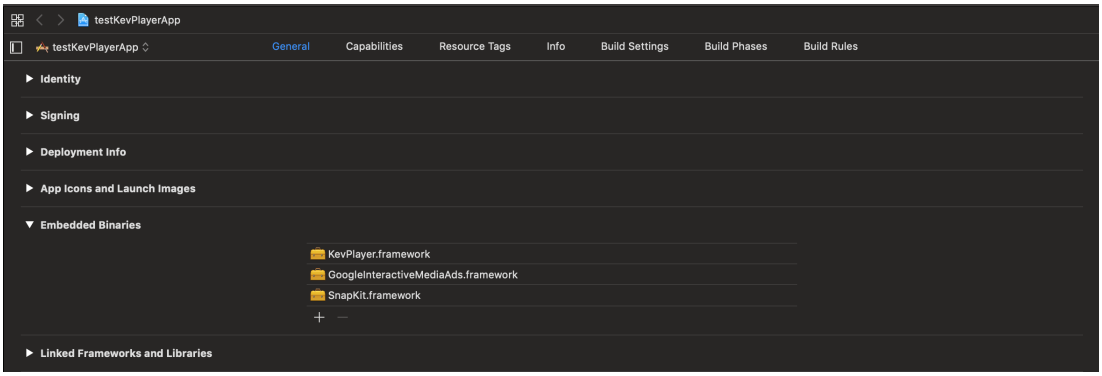
Obr. D.1: Nastavení účtu ve vývojovém prostředí.

Dále je nutné zvolit, na kterém zařízení se projekt spustí. Vyvinutá knihovna nepodporuje simulátor, z tohoto důvodu je tedy potřebné připojit reálné zařízení (iPhone) k vývojovému prostředí s operačním systémem iOS 11 a výše. Mobilní telefon lze k zařízení se systémem macOS připojit kabelem nebo bezdrátově, pokud je tato funkcionalita ve vývojovém prostředí vývojářem nastavena. Po připojení a povolení přístupu k iPhone je možné spárované zařízení zvolit v nabídce zařízení. Poté již stačí projekt spustit a po úspěšném sestavení se aplikace spustí na požadovaném zařízení. Volba připojeného zařízení je znázorněna na obrázku D.2.

Obr. D.2: Volba připojeného zařízení.

E Návod pro implementaci knihovny

Nejprve je potřebné knihovnu tzv. sestavit (provést build), což je možné pomocí kláves *cmd* + *B*. Poté je možné sestavenou knihovnu vidět ve vývojovém prostředí či ve Finderu zařízení macOS (ve složce projektu) pod složkou *Products*. Sestavenou knihovnu lze poté vložit do požadovaného projektu přetáhnutím či stisknutím tlačítka "+"do *Embedded binaries*, což je možné nalézt v nastavení projektu. Stejným způsobem je také nutné implementovat knihovny, které v sobě vyvinutá knihovna implementuje. Sekci s již implementovanými knihovnami je možné si prohlédnout na obrázku E.1.



Obr. E.1: Implementované knihovny v projektu.