Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Framework pro záznam videa na základě události pro platformu iOS

Dominik Chmelík

Bakalářská práce

2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Dominik Chmelík**
Osobní číslo:	**I19096**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Framework pro záznam videa na základě události pro platformu iOS**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Tato bakalářská práce se zabývá návrhem a implementací frameworku pro platformu iOS, který bude umožňovat nahrávání videa z kamery mobilního zařízení, a to na základě událostí vyvolaných z aplikace. V teoretické části představí použité technologie a programovací jazyk Swift. Součástí bude taktéž porovnání různých řešení této problematiky a následná implementace jednoho z řešení. Framework bude vyvíjen ve spolupráci se společností nextap solutions s.r.o.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

LACKO, Ľuboslav. *Vývoj aplikací pro iOS*. Přeložil Martin HERODEK. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.
MCCUNE, Bob. *Learning AV Foundation: a hands-on guide to mastering the AV Foundation framework*. Upper Saddle River, NJ: Addison-Wesley, [2014]. Addison-Wesley learning series. ISBN 978-0321961808.
NEUBURG, Matt. *iOS 14 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics*. O'Reilly Media, 2020. ISBN 9781492092094.

Vedoucí bakalářské práce: **Ing. Jan Panuš, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 13. 5. 2022

Dominik Chmelík

Poděkování

Rád bych poděkoval svému vedoucímu, panu Ing. Janu Panušovi, Ph. D., za ochotu a cenné rady při psaní této bakalářské práce. Dále chci poděkovat své rodině a přátelům za jejich podporu při studiu.

ANOTACE

Cílem této bakalářské práce je návrh a implementace frameworku určeného pro nahrávání videa na základě události, pro operační systém iOS, s využitím programovacího jazyka Swift a frameworku AVFoundation. Součástí teoretické části práce je popis použitých technologií, jako jsou programovací jazyk Swift, operační systém iOS, vývojové prostředí Xcode, framework AVFoundation a další. Praktická část práce se zabývá návrhem a porovnáním několika řešení. Dále výběrem vhodného řešení a jeho následnou implementací.

KLÍČOVÁ SLOVA

Framework, iOS, macOS, AVFoundation, Swift, Xcode, Git, video

TITLE

Framework for event-based video recording for iOS platform.

ANNOTATION

The goal of this bachelor's thesis is the design and implementation of a framework intended for recording event-based videos, for the iOS operating system. This will be achieved by using the Swift programming language and the AVFoundation framework. The theoretical part of the work will describe used technologies such as the Swift programming language, the iOS operating system, the Xcode development environment, and more. The practical part of the work deals with the design and with comparing several possible solutions. Furthermore, the selection of a suitable solution and its subsequent implementation.

KEYWORDS

Framework, iOS, macOS, AVFoundation, Swift, Xcode, Git, video

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

WPF	Windows Presentation Foundation
UI	User Interface
WWDC	Worldwide Developer Conference
ABI	Application Binary Interface
LSP	Language Server Protocol
HTTP	Hypertext Transfer Protocol
CSV	Concurrent Version System
DVCS	Distributed Version Control System
VGA	Video Graphics Array
HD	High Definition
FHD	Full High Definition
UHD	Ultra High Definition
FPS	Frames Per Second
MP4	Moving Picture Experts Group 4 (MPEG-4)
MOV	Přípona video souboru ve formátu QuickTime File Format (QTFF)

ÚVOD

Představme si situaci, kdy bychom chtěli využít kameru našeho mobilního zařízení pro nahrávání krátkých úseků videa, které obsahují námi sledované události. Při využití systémové aplikace kamery bychom museli po celou dobu sledování nahrávaný záznam ukládat do paměti zařízení. Toto by vedlo k plýtvání pamětí, jejíž kapacita by byla pro takovéto účely většinou nedostatečná. Dokonce i v případě, že bychom uvažovali nekonečně velkou paměť, měli bychom problém s následným zpracováním výsledného videa. Museli bychom totiž záznam manuálně projít a vyexportovat pouze části obsahující požadované události. Takovéto záznamy mohou být i několik hodin dlouhé, proto by manuální zpracování vyžadovalo hodně usilí a času uživatele. Také by se mohlo stát, že by uživatel omylem přeskočil některou z událostí a nevyexportovat by ji. Proč bychom ale měli ukládat celý záznam, když tyto události mohou mezi sebou mít značný časový rozestup? Většina záznamu nás nezajímá, pouze zabírá cennou paměť.

Hlavním cílem této bakalářské práce je tedy navrhnout a implementovat tuto funkcionalitu, která odstraní nevýhody zmíněné výše. Tato funkcionalita bude vyvíjena ve formě frameworku, a to konkrétně pro platformu iOS.

Teoretická část práce představí a popíše využité technologie a nástroje. Bude zde popsán termín framework, představeny operační systémy iOS a macOS, zmíněno vývojové prostředí Xcode, programovací jazyk Swift, framework AVFoundation a další.

Praktická část práce se poté bude zabývat návrhem několika možných řešení, jejich porovnáním a následným výběrem jednoho funkčního řešení vhodného pro implementaci vlastního frameworku. Dále v ní budou popsány základní prvky frameworku, jako je jeho rozhraní a možnosti konfigurace. Tyto prvky jsou důležité pro vývojáře využívající tento framework, protože umožňují komunikaci s frameworkem a v případě potřeby i změnu jeho konfigurace, jako je například kvalita videa nebo délka nahrávaných úseků.

1 FRAMEWORK

Protože se budeme v této práci zabývat vývojem frameworku, bylo by vhodné alespoň okrajově zmínit, co to takový framework je a proč je výhodné frameworky používat. Dále si představíme některé druhy frameworků a popíšeme si rozdíly mezi frameworky a knihovnami.

1.1 Co je to framework

Framework je v oblasti programování nástroj vytvořený programátory, z důvodu urychlení a zjednodušení vývoje softwaru. Zjednodušeně řečeno, je to v podstatě balíček knihoven a dalších užitečných funkcionalit, které jsou připraveny k použití v nových projektech, aniž by vývojáři museli implementovat již jednou vyřešenou problematiku. Toto vede k zjednodušení a urychlení vývoje. Použití frameworku sice přináší tyto výhody, ale pokud s ním vývojáři pracují poprvé, je nutné se s ním nejdříve seznámit a nastudovat si jeho použití. Z tohoto důvodu první použití většinou zabere více úsilí a času. Pokud jsou ale vývojáři s konkrétním frameworkem seznámeni, vývoj se při opakovaném použití velmi zrychlí. [33]

1.2 Typy frameworků

V dnešní době je na výběr z mnoha programovacích jazyků, které se používají pro různé platformy. V případě frameworků to není jiné, je vyvíjeno mnoho frameworků v různých programovacích jazycích a pro různé platformy. Proto se frameworky rozdělují podle použití do několika kategorií, jako jsou frameworky pro tvorbu webů a webových aplikací, dále například mobilní a desktopové frameworky. [33]

1.2.1 Webové frameworky

Jak už lze z názvu odvodit, webové frameworky slouží pro zjednodušení vývoje webových stránek a webových aplikací. Tyto frameworky se dále dělí na frontendové a backendové. Frontendové slouží pro vývoje uživatelského rozhraní, tedy jak bude webová stránka nebo aplikace vypadat, zatímco backendové slouží pro návrh chování nebo například pro komunikaci s databází. [33]

1.2.2 Mobilní frameworky

Tyto frameworky, používané při vývoji aplikací pro mobilní zařízení, se dělí na frameworky multiplatformní (cross-platform) a frameworky nativní. Při použití multiplatformích frameworků lze danou aplikaci vyvíjet pro více platforem najednou, například pro operační systémy iOS a Android. U nativních frameworků tomu tak ale není, proto je nutné předem zvolit cílovou platformu pro kterou bude aplikace vyvíjena a zvážit, zda je výhodnější použít nativní či multiplatformní framework. [33] Jedním z nejznámějších frameworků pro vývoj multiplatformních mobilních aplikací je Xamarin, ve kterém lze vyvíjet aplikace pro operační systémy iOS a Android v programovacím jazyce C#. [27]

Protože je tato práce zaměřená na platformu iOS, nesmím opomenout dva frameworky pro vývoj nativních iOS aplikací, mezi které patří frameworky UIKit a jeho nástupce SwiftUI. Při použití frameworku UIKit je grafické uživatelské rozhraní většinou tvořeno v tzv. storyboard souborech, pomocí grafického editoru Interface Builder, integrovaného ve vývojovém prostředí Xcode. V případě použití frameworku SwiftUI je tomu jinak, protože se uživatelské rozhraní píše přímo v programovacím jazyce Swift. [15]

1.2.3 Desktopové frameworky

Pro vývoj desktopových aplikací je na výběr taktéž z mnoha frameworků. Mezi tyto frameworky patří například WPF (Windows Presentation Foundation) pro vývoj uživatelského rozhraní aplikací pro Windows. Dále například framework Swing, který slouží

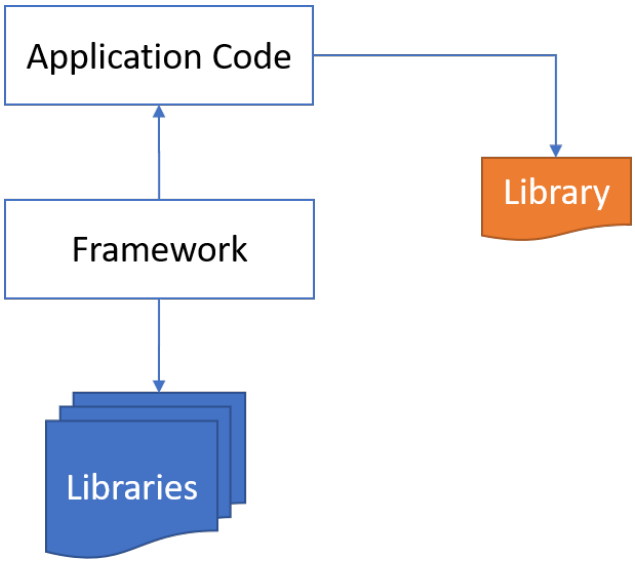
pro vývoj multiplatformních aplikací v jazyce Java, nebo framework Cocoa určený pro vývoj nativních aplikací pro operační systém macOS. [28]

1.2.4 Frameworky pro vývoj her

Hry je možné vyvíjet v různých jazycích, nejčastěji se setkáme s jazyky C++, C#, Java, pro webové hry pak s jazyky JavaScript nebo Python [21]. Vývoj her je náročná a komplexní záležitost, proto vznikly herní frameworky, aby vývoj usnadnily a zrychlily. Tyto frameworky nejčastěji ustadňují mimo jiné práci s grafikou a fyzikou. [25]

1.3 Framework vs knihovna

Pojmy framework a knihovna jsou často zaměňované i zkušenými programátory. Knihovny obsahují znovupoužitelný kód, který je programátorovi k dispozici, zatímco framework je většinou složen z více knihoven. Při použití knihovny má programátor plnou kontrolu, kdy a jak tuto knihovnu použije, ale při použití frameworku o volání a použití knihoven rozhoduje framework samotný. [31]



Obrázek 1: Framework a knihovna. [31]

2 POUŽITÉ TECHNOLOGIE

2.1 Operační systém iOS

Na konferenci dne 9. 1. 2007 zakladatel společnosti Apple, Steve Jobs, představil první mobilní telefon iPhone, který neobsahoval zabudovanou klávesnici, ale byl ovládán dotykem. Tuto revoluční dotykovou technologii Apple nazval jako Multi-Touch a oznámil, že ji patentoval. Dále na konferenci bylo zmíněno, že iPhone využívá existujícího operačního systému OS X, určeného pro desktopové počítače Apple. [2] Tímto započal vývoj mobilního operačního systému iPhone OS, jehož název byl později v roce 2010 zkrácen na iOS. [9] Aktuální verze v době psaní této práce byla verze 15.

Obrázek 2: Logo operačního systému iOS 15. [3]

Operační systém iOS byl nejen pro mobilní telefony iPhone, ale také pro další zařízení, mezi která patří i tablety iPad. Tato dvě zařízení zpočátku nabízela podobné funkcionality, ale postupem času se do operačního systému iOS začaly implementovat speciální funkcionality dostupné pouze na iPad. Tyto funkcionality zahrnovaly rozdělení pracovní plochy na více obrazovek, což umožňuje multitasking, dále například podporu Apple Pencil. To znamená, že některé funkce byly v mobilních telefonech iPhone vypnuty. Z tohoto důvodu se společnost Apple v roce 2019 rozhodla rozdělit operační systém iOS na dvě větve. První původní větev s názvem iOS a druhou větev s názvem iPadOS určenou pro tablety iPad. Číslování verzí iPadOS vycházelo z číslování iOS, proto první verze iPadOS byla 13. [20]

Název	Aktuální verze	Datum vydání	Datum poslední verze
iOS 15	15.4 (v době psaní této práce)	24. 9. 2021	14. 3. 2022
iOS 14	14.6	17. 9. 2020	24. 5. 2021
iOS 13	13.7	19. 9. 2019	1. 9. 2020
iOS 12	12.4.8	17. 9. 2018	15. 7. 2020
iOS 11	11.4.1	19. 9. 2017	9. 7. 2018
iOS 10	10.3.4	13. 9. 2016	22. 7. 2019
iOS 9	9.3.9	15. 9. 2015	22. 7. 2019
iOS 8	8.4.1	17. 9. 2014	13. 8. 2015
iOS 7	7.1.2	18. 9. 2013	30. 6. 2014
iOS 6	6.1.6	19. 9. 2013	21. 2. 2014
iOS 5	5.1.1	12. 10. 2011	7. 5. 2012
iOS 4	4.3.5	22. 6. 2010	25. 7. 2011
iOS 3	3.2.2	17. 6. 2009	11. 8. 2010
iOS 2	2.2.1	11. 7. 2008	27. 1. 2009
iOS 1	1.1.5	29. 6. 2007	15. 7. 2008

Tabulka 1: Přehled verzí iOS. Zdroj dat: [14].

2.1.1 Vývoj aplikací pro iOS

Ačkoli je možné iOS aplikace vyvíjet i na operačním systému Windows, ve vývojovém prostředí Visual Studio, s pomocí frameworku Xamarin, je nutné mít pro vydání aplikace do App Store počítač s operačním systémem macOS. Tuto skutečnost je možné obejít virtualizací nebo pomocí tzv. Hackintosh počítače. [23]

Hackintosh je počítač, na kterém je nainstalovaný operační systém macOS, ale nebyl vyroben společností Apple. Jedná se tedy o neoriginální hardware. Prakticky se nejedná o nelegální krok, dokud se nesnažíte tento počítač prodat. Postavení Hackintosh není ale úplně jednoduché, protože není podporován veškerý hardware, proto je nutné skládat počítačové sestavy, které jsou s projektem Hackintosh kompatibilní. [13]

Dle mého názoru je tedy nejvhodnější vyvíjet aplikace na originální počítači Macintosh, a to s využitím vývojového prostředá Xcode a moderního programovacího jazyka Swift. Programovací jazyk Swift, vývojové prostředí Xcode a operační systém macOS si představíme v následujících kapitolách.

2.2 Operační systém macOS

Protože budu framework vyvíjet na originálním hardwaru společnosti Apple, tedy počítači Macintosh, je dle mého názoru vhodné alespoň v jedné kapitole popsat operační systém macOS. Zmíním zde základní informace o tomto operačním systému a uvedu přehled jednotlivých verzí tohoto operačního systému.

Obrázek 3: Logo operačního systému macOS. [4]

Poté co byl spoluzakladatel společnosti Apple, Steve Jobs, v roce 1985 vyhozen ze společnosti Apple, založil další společnost jménem NeXT. Nato v roce 1987 tato společnost představila operační systém NeXTSTEP, založený na unixové architektuře. Mezitím se společnosti Apple přestalo dařit, a proto se rozhodla koupit operační systém NeXTSTEP společně s Jobsovou společností NeXT. Poté v březnu roku 2001 byl společností Apple představen operační systém Mac OS X, založaný na již zmíněném operaním systému NeXTSTEP. Jeho první verze nesla označení 10.0. Cheetah. [7]

Na WWDC konferenci v roce 2016, při představení nové verze systému, byl původní název Mac OS X změněn na macOS. Nová verze tedy nesla název macOS 10.12. Sierra. [17]

Obrázek 4: Logo operačního systému macOS Monterey. [5]

Název	Aktuální verze	Datum vydání první verze
macOS Monterey	12.3.1 (v době psaní této práce)	25. 10. 2021
macOS Big Sur	11.6.5	12. 11. 2020
macOS Catalina	10.15.7	7. 10. 2019
macOS Mojave	10.14.6	24. 9. 2018
macOS High Sierra	10.13.6	25. 9. 2017
macOS Sierra	10.12.6	20. 9. 2016
OS X El Capitan	10.11.6	30. 9. 2015
OS X El Yosemite	10.10.5	16. 10. 2014
OS X El Mavericks	10.9.5	22. 10. 2013
OS X El Mountain Lion	10.8.5	25. 7. 2012
OS X El Lion	10.7.5	20. 7. 2011
Mac OS X Snow Leopard	10.6.8	28. 8. 2009
Mac OS X Leopard	10.5.8	26. 10. 2007
Mac OS X Tiger	10.4.11	29. 4. 2005
Mac OS X Panther	10.3.9	24. 10. 2003
Mac OS X Jaguar	10.2.8	24. 8. 2002
Mac OS X Puma	10.1.5	25. 9. 2001
Mac OS X Cheetah	10.0.4	24. 3. 2001

Tabulka 2: Přehled verzí macOS. Zdroj dat: [18].

2.3 Programovací jazyk Swift

2.3.1 Úvod do Swiftu

Swift je moderní programovací jazyk vyvíjený společností Apple. Cílem projektu Swift bylo vytvořit bezpečný, rychlý a přehledný programovací jazyk, který lze použít pro rozmanité projekty, od nejjednodušších až po náročné, které zahrnují mimo jiné mobilní, desktopové a cloudové aplikace. [1]

Kód, který je napsaný v programovacím jazyce Swift, je optimalizovaný tak, aby co nejefektivněji využil hardware zařízení od společnosti Apple [23]. Je vhodný pro programování aplikací pro operační systémy iOS, iPadOS, macOS, tvOS a watchOS [32].

Bezpečnost kódu je dalším požadavkem autorů. Swift by měl eliminovat nebezpečný kód, který může být způsoben například zapomenutím inicializace proměnných nebo přeteče-

ním paměti. Proto přiřazení hodnoty nil vede k chybě při kompilaci. Samozřejmě je někdy žádoucí, aby mohla proměnná nabývat hodnoty nil, proto má Swift pro tyto situace vlastnost známou jako optionals. Proměnná označená jako optional smí nabývat hodnoty nil, ale při jejím použití je vynuceno kontrolovat přítomnost hodnoty. [1]

Další klíčová vlastnost, která má za následek nejen zvýšení bezpečnosti, ale také zvýšení rychlosti vývoje aplikací, je automatická správa paměti. Z důvodu automatické správy paměti a absence ukazatelů, které lze nalézt například v programovacím jazyce C, je Swift vhodný i pro začínající vývojáře. [23]

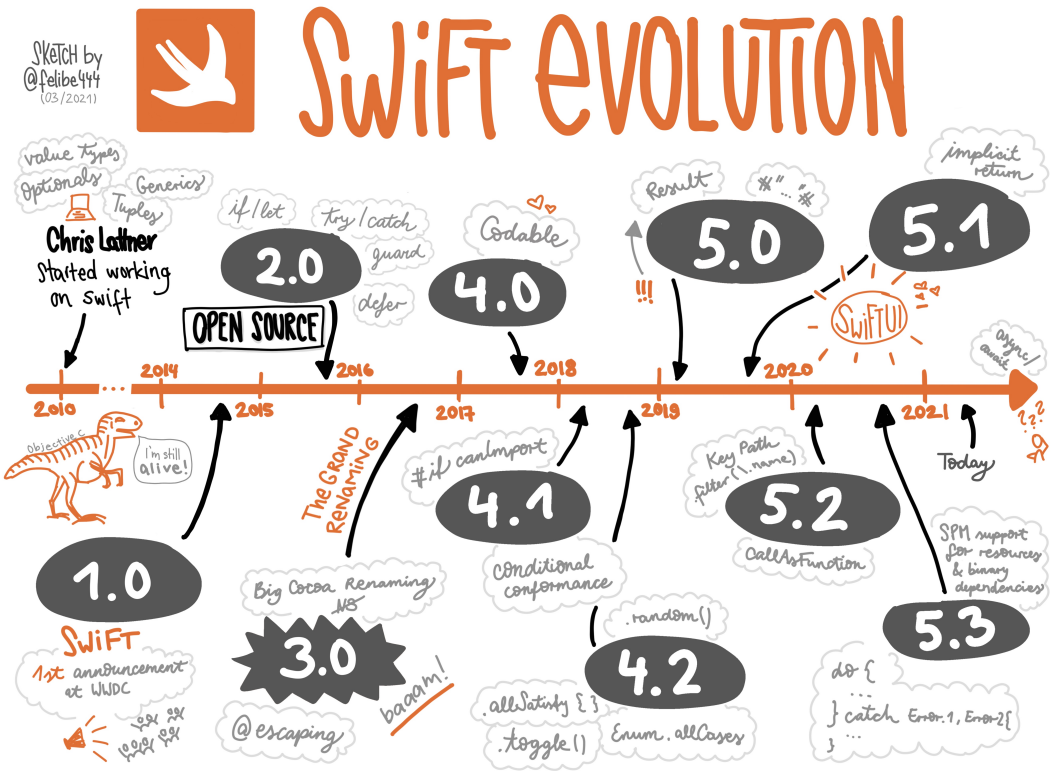
Obrázek 5: Logo programovacího jazyka Swift. [6]

2.3.2 Historie a vývoj

V červnu roku 2014 byl na konferenci WWDC (Worldwide Developer Conference) představen nový moderní programovací jazyk Swift, jakožto nástupce staršího jazyka Objective-C, který se používal pro vývoj aplikací pro zařízení od společnosti Apple. Ještě tentýž rok byla k dispozici jeho první verze. [23]

Později na konferenci WWDC v roce 2015 byla představena druhá verze jazyka Swift 2, která přinesla řadu změn, které byly založeny na připomínkách vývojářské komunity. V prosinci téhož roku společnost Apple oficiálně vydala programovací jazyk Swift jako open-source a zveřejnila ho ve svém veřejném repozitáři `https://github.com/apple`. Dále byl zveřejněn repozitář `https://github.com/apple/swift-evolution`, který dokumentuje seznam provedených změn při vývoji. Veškerá potřebná dokumentace je dostupná na oficiálním webu `https://www.swift.org`. [19]

Verze jazyka Swift 3, vydaná v roce 2016, přispěla otevřenosti podporou kompilace zdrojového kódu pro ostatní platformy, jako je například operační systém Linux [19]. Dobrou



Obrázek 6: Vývoj programovacího jazyka Swift. [11]

zprávou pro vývojáře byla i zpětná kompatibilita se starším programovacím jazykem Objective-C, protože se aplikace do roku 2015 vyvíjely převážně v tomto jazyce. Toto umožňuje začít používat Swift pro psaní nových funkcionalit v již existujících projektech založených na jazyce Obejctive-C. A naopak používat starší frameworky psané v Objective-C v nových projektech. [23]

Jedním z hlavních cílů swift 4, vydaného v roce 2017, byla zpětná kompatibilita zdrojového kódu s předchozí verzí Swift 3. To umožňovalo kompilovat starší projekty pomocí nového kompilátoru jazyka Swift 4. Dalším cílem bylo stabilizovat Application Binary Interface (ABI), umožňující šíření frameworků v binárním formátu. [19]

Od verze Swift 5 je umožněno použít zkompilovaný framework pro verzi 5 i s budoucími verzemi. Dále byl ve verzi 5.1. představen LSP (Language Server Protocol), který umožňuje podporu funkcionalit, jako jsou zvýraznění syntaxe a automatické doplňování, bez podpory jazyka editorem. Tyto funkcionality jsou přímo integrované v jazyce, takže každý editor podporující LSP podporuje Swift. Operační systém Windows je jazykem Swift oficiálně podporován od verze 5.3. [19]

2.4 Vývojové prostředí Xcode

Xcode je vývojové prostředí vydané společností Apple v roce 2003, které vychází z prostředí Project Builder, původně vyvíjeného společností NeXT. Toto vývojové prostředí je zdarma ke stažení v App Store. Je určené pro vývoj aplikací, frameworků a dalších softwarových projektů pro zařízení Applu. Patří mezi ně zařízení s operačními systémy iOS, iPadOS, macOS, watchOS a tvOS. Projekty je možné psát v jazycích Swift, Objective-C, C, C++ nebo v jejich kombinacích. Toto vývojové prostředá má navíc integrovanou podporu pro verzovací systém Git, takže není potřeba používat klienty třetích stran. [36]

Obrázek 7: Ikona vývojového prostředí Xcode. [10]

Název Xcode není pouze názvem vývojového prostředí, je to název balíku všech užitečných nástrojů potřebných při vývoji aplikací a dalšího softwaru, tyto nástroje dohromady tvoří právě vývojové prostředí Xcode. Tento balík zahrnuje mnoho nástrojů určených mimo jiné k tvorbě, testování a distribuci aplikací. [22]

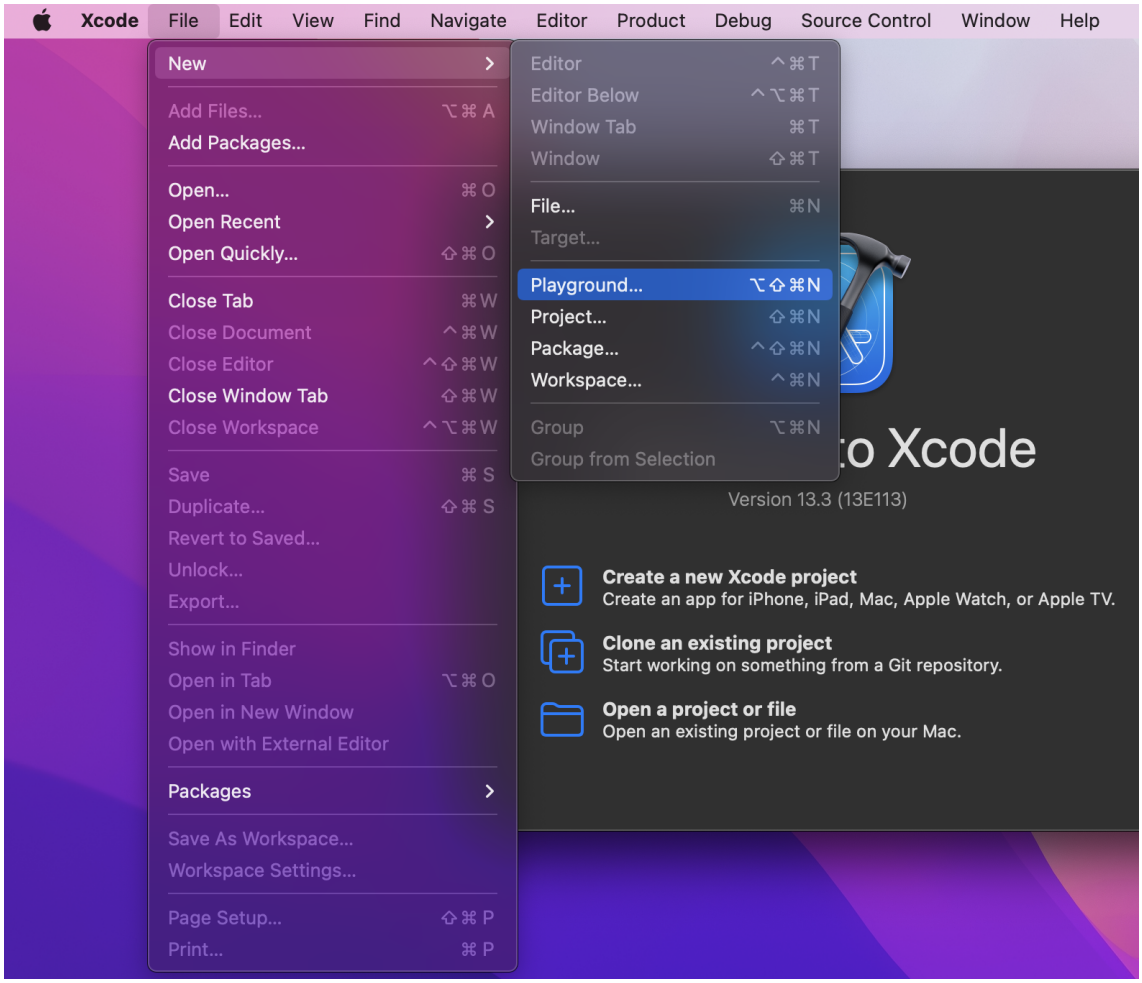
2.5 Playground

S vydáním Xcode 6 a programovacího jazyka Swift byl vydán i takzvaný Playground, který je vhodný pro experimentování s jazykem Swift, aniž by bylo nutné vytvořit projek reálné aplikace. Protože se nemusí kompilovat a sestavovat celý projekt aplikace, je experimentování efektivnější a zabere méně času. Playground je vlastně konzolová aplikace, která má navíc možnost zobravovat výsledky nejen v textové, ale i grafické podobě, jako jsou grafy. Další přívětivou vlastností je, že po napsání je kód okamžitě zpracován a vypsán výsledek. Vývojové prostředí Xcode je zdarama, proto je Playground vhodný nejen pro vývojáře, ale pro kohokoliv, kdo se chce naučit s programovacím jazykem Swift,

ale nepotřebuje pracovat na projektu aplikace. Protože práce v Playground je snadná, je vhodný i pro výuku programování ve školství. [23], [22]

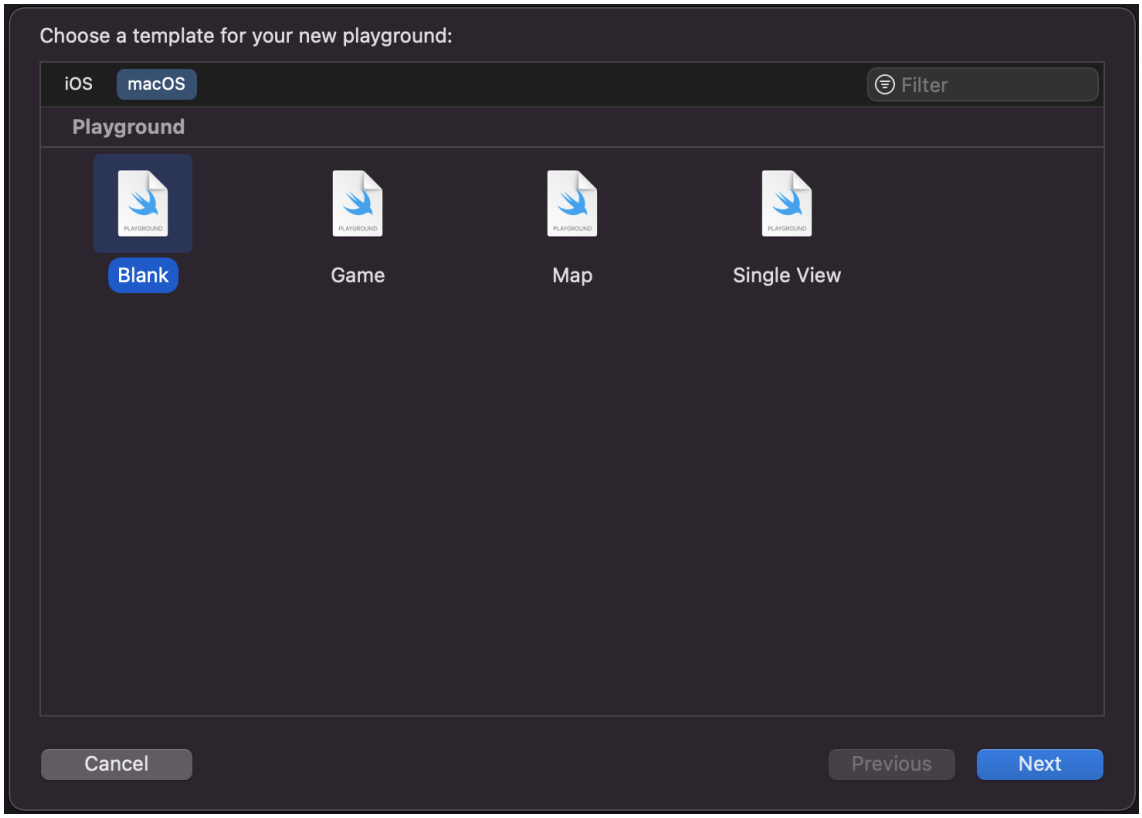
Při vývoji frameworku pro tuto práci jsem Playground využíval pro experimentování, proto zde popíšu vytvoření projektu Playground, představím jeho základní prvky a ukážu grafický výstup na jednoduchém programu.

Ještě ve verzi Xcode 12 bylo možné projekt Playground vytvořit přes uvítací obrazovku, v nové verzi Xcode 13 se tato možnost nenabízí. Playgound je možné vytvořit přes řádek nabídek vývojového prostředí Xcode, a to přes příkaz **File** → **New** → **Playground**. Na obrázku 8 je zobrazen příkaz pro vytvoření nového projektu Playground. Projekt je možné vytvořit také pomocí klávesové zkratky, která je taktéž zobrazena na obrázku 8, napravo od příkazu **Playground**.



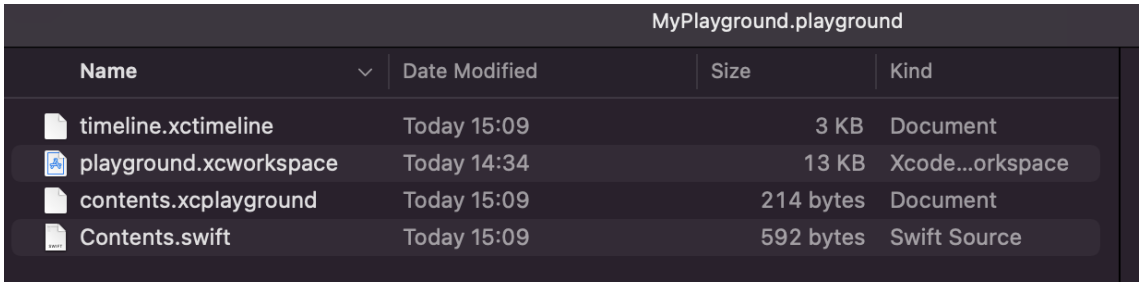
Obrázek 8: Vytvoření projektu Playground pomocí řádku nabídek. [35]

Po kliknutí na tlačítko nebo po zadání klávesové zkratky je zobrazeno okno (obrázek 9) pro výběr typu projektu. Sám jsem vždy používal pouze šablonu **Blank**, proto zde nebudu popisovat ostatní tři šablony, mezi které patří **Game**, **Map** a **Single View**.

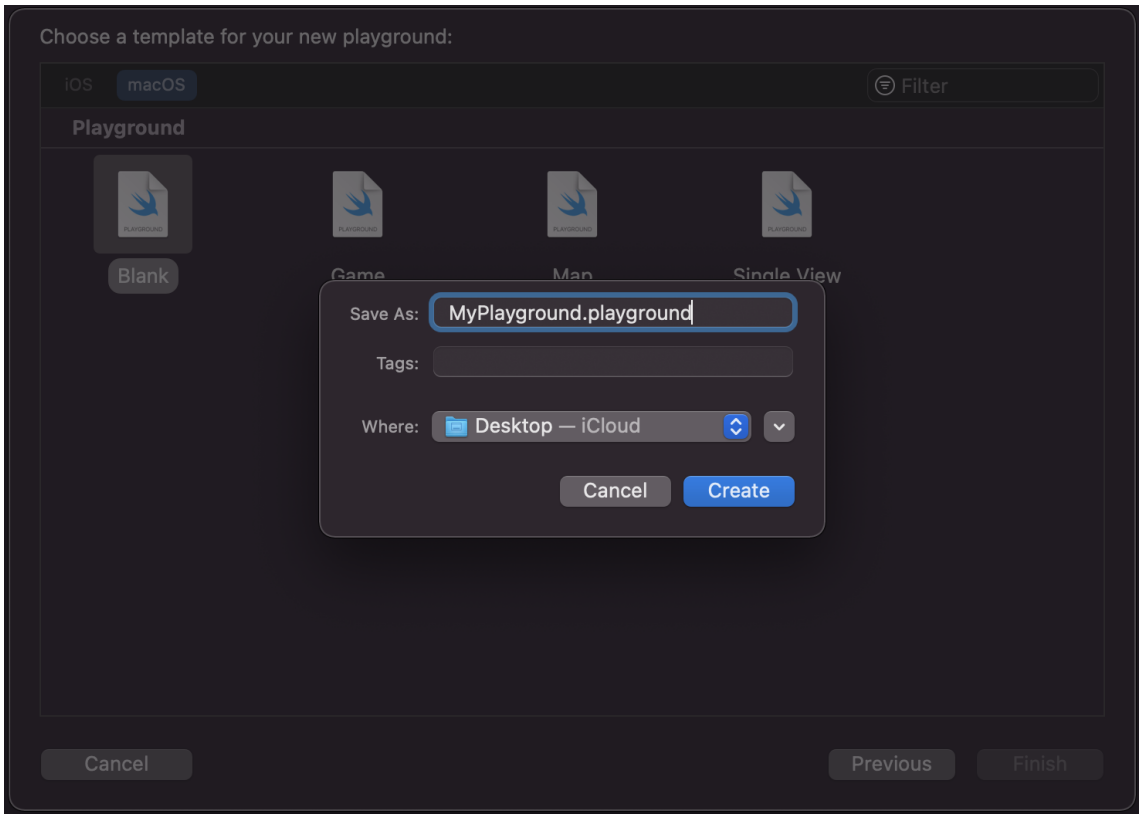


Obrázek 9: Okno pro výběr šablony nového projektu Playground. [35]

Po zvolení šablony a kliknutí na tlačítko **Next**, se vývojové prostředí Xcode zeptá na umístění nového projektu (obrázek 11). Všimněte si, že celý projekt je obsažen v jediném souboru s příponou „.playground“. Ve skutečnosti se ale nejedná o soubor, ale o balíček, který obsahuje pomocné soubory a soubor se zdrojovým kódem v jazyce Swift. Obsah balíčku je vypsán na obrázku 10.



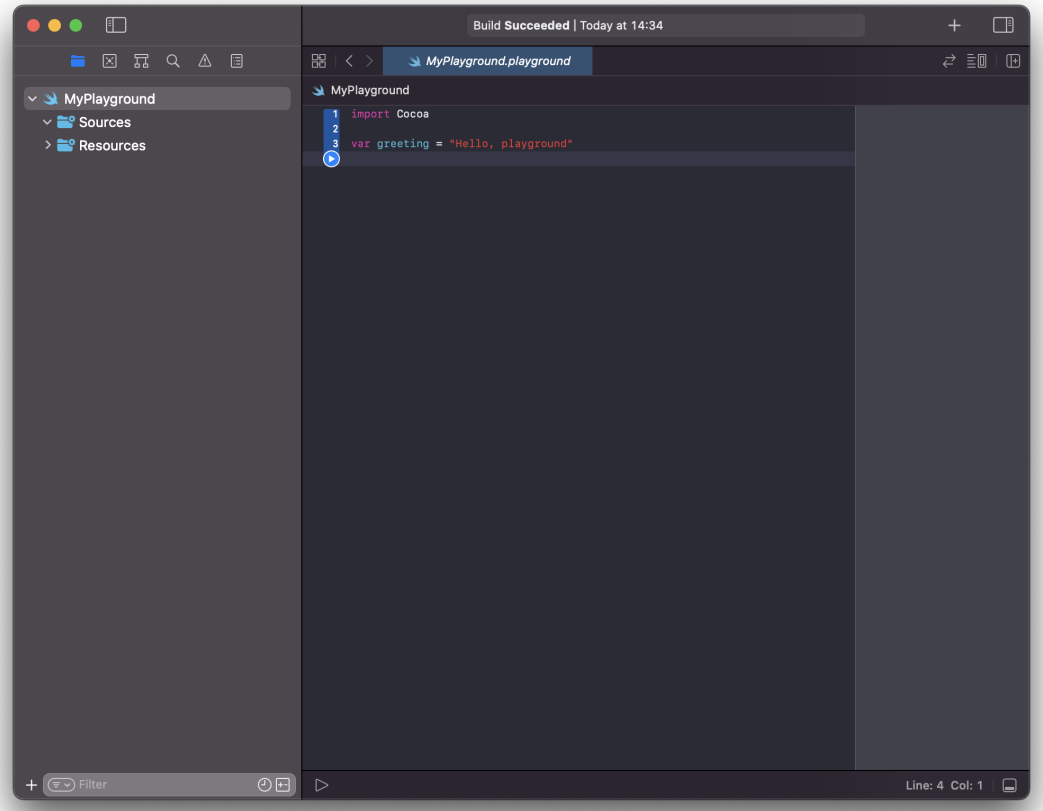
Obrázek 10: Obsah balíčku projektu Playground. [24]



Obrázek 11: Okno pro zvolení umístění nového projektu Playground. [35]

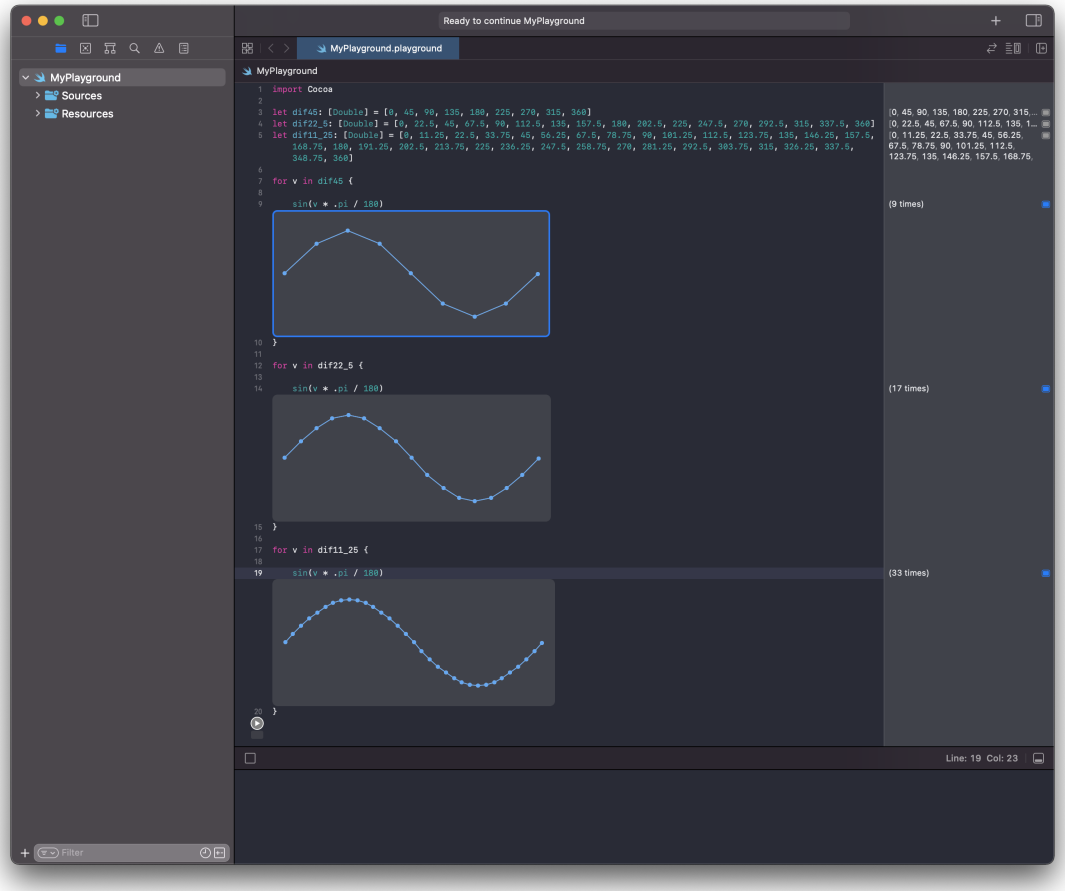
Po vytvoření projektu se otevře vývojové prostředí Xcode s jednoduchým kódem, který vypíše řádek textu do konzole. Na obrázku 12 si všimněte dvou složek v projektu MyPlayground, a to konktétně složek Sources a Resources.

Složka Sources umožňuje snadný přístup k dalším zdrojovým souborům programovacího jazyka Swift. V případě potřeby je v ní možné vytvořit soubor se zdrojovým kódem, takže nemusí být všechem kód napsán v jediném souboru, což vede ke zpřehlednění projektu. Dále složka Resources slouží pro vkládání dalšího obsahu, ke ketrému je potřeba přistupovat z kódu projektu. Tyto soubory mohou zahrnovat například obrázky. [29]



Obrázek 12: Ukázka zdrojového kódu při vytvoření nového projektu Playground. [35]

V následující ukázce jsem si připravil jednoduchý program, který vykresluje průběh goniometrické funkce sinus v grafické podobě. V tomto programu jsou deklarována tři pole, která představují vstupní hodnoty funkce sinus. Všechna tato pole obsahují hodnoty od 0 do 360 stupňů, ale liší se počtem hodnou. První pole obsahuje 9 hodnot, mezi kterými je rozdíl 45 stupňů, druhé pole 17 hodnot s rozdílem 22,5 stupně a nakonec třetí pole s 33 hodnotami a rozdílem 11,25 stupňů. Program má demostrovat vliv počtu vstupních hodnot na detail vykreslení grafů, které je možné vidět na obrázku 13.



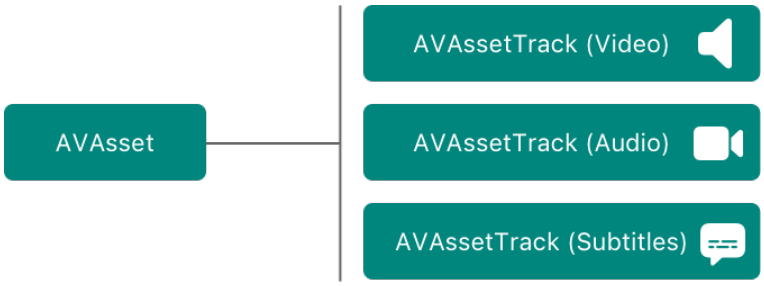
Obrázek 13: Grafický výstup programu v projektu Playground. [35]

2.6 Framework AVFoundation

AVFoundation je framework od společnosti Apple, který je určen pro práci s audiovizuálními médii. Umožňuje nahrávat, zpracovávat, importovat a exportovat tato média. [26]

2.6.1 Struktura médií a jejich přehrávání

Jednou z hlavních tříd v tomto frameworku je třída AVAsset, která slouží jako kontejner pro uchovávání audiovizuálních médií. Tento kontejner je složen z jedné nebo více instancí třídy AVAssetTrack, která obaluje konkrétní typ média, jako je například video, audio nebo titulky, dále tato třída obsahuje metadata. AVAsset může obsahovat kombinace různých druhů médií a spojit je do jednoho celku, umoňuje tedy například k videu přidat zvukovou stopu a titulky. Pro lepší představu se podívejte na obrázek 14. [8]

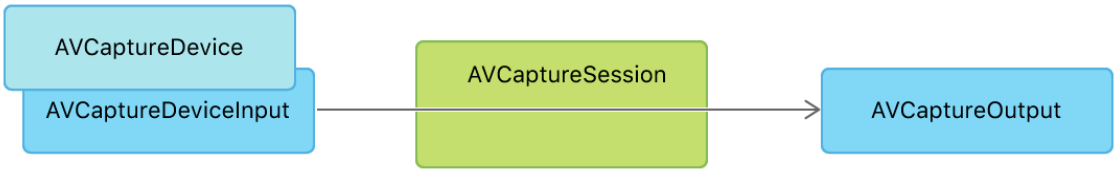


Obrázek 14: AVAsset a AVAssetTrack. [8]

Jednou z dalších velmi užitečných tříd je třída AVPlayer, která slouží pro přehrávání audiovizuálních médií, ať už z lokálního nebo vzdáleného uložiště. Jedná se tedy o přehrávač médií, který umí přehrávat média načtěná například z disku zařízení i živé vysílání přijímané přes HTTP. Pokud by bylo potřeba přehrávat více médií za sebou, není nutné implementovat vlastní řešení, lze použít již vytvořenou třídu AVQueuePlayer, která je potomkem přídy AVPlayer a umožňuje přehrávat média ve frontě. AVPlayerLooper je další potomková třída, která umožňuje přehrávat média od začátku do konce ve smyčce, aniž by bylo potřeba implementovat vlastní řešení. Stav přehrávače AVPlayer je uložen v objektu třídy AVPlayerItem, který obsahuje právě přehrávané médium a další informace, jako je čas a stav přehrávání. [8]

2.6.2 Nahrávání médií

AV Foundation poskytuje nástoje pro nahrávání videa, audia a pořizování fotografií na operačních systémech iOS a macOS. Je možné tedy vyvíjet aplikaci kamery s vlastním uživatelským rozhraním a chováním. Jedním z hlavních prvků je třída AVCaptureSession, která spojuje jeden nebo více vstupů a jeden nebo více výstupů. Pod vstupem si je možné představit vstupní zařízení, jako jsou kamery nebo mikrofony. Výstupem je poté médium v podobě souborů uložených na disku, nebo buffer s obraovými nebo zvukovými daty, která lze použít pro zpracování v reálném čase. Na obrázku 15 je možné vidět na levé straně vstupní zařízení a na pravé straně výstupní proud dat. [8]



Obrázek 15: Spojení vstupů a výstupů pomocí AVCaptureSession. [8]

Základní třídy používané při nahrávání médií.:

AVCaptureSession

Tento objekt propojuje vstupy a výstupy. Může propojovat i více vstupů s více výstupy současně. Koordinuje tok dat z vstupních zažízení do výstupů. Dále umožňuje konfigurovat chování toku dat. [8]

AVCaptureInput

Rodičovská abstraktní třída, která reprezentuje objekty, které poskytují vstupní data do AVCaptureSession. [8]

AVCaptureOutput

Rodičovská abstraktní třída, která reprezentuje výstup dat z AVCaptureSession. [8]

AVCaptureConnection

Reprezentuje spojení mezi vstupem (AVCaptureInput) a výstupem (AVCaptureOutput). [8]

AVCaptureDevice

Reprezentuje fyzické nebo virtuální zařízení pro nahrávání videa nebo audia. [8]

AVCaptureDeviceInput

Konkrétní potomek třídy AVCaptureInput. Objekt této třídy poskytuje vstupní data pro AVCaptureSession ze vstupních zařízení. [8]

AVCaptureVideoPreviewLayer

Potomková třída třídy CALayer, která umožňuje zobrazit náhled videa v reálném čase. [8]

2.6.3 Čtení a ukládání médií

Framework AVFoundation umožňuje kromě čtení a ukládání médií také konverzi těchto médií do jiných formátů. Pokud se jedná o video, umí také například extrahovat snímky z videa ve formě fotografií nebo naopak z fotografií poskládát video. Dále umožňuje mixování zvukových stop a vytváření video kompozic. [8]

Mezi nejpoužívanější třídy patří.:

AVAssetExportSession

Umožňuje exportovat AVAsset média v požadovaném formátu. [8]

AVAssetImageGenerator

Umí extrahovat snímky z videa v požadovaném čase. Lze využít například pro vygenerování náhledu videa, tzv. thumbnail. [8]

AVAssetReader

Objekt třídy umožňující čtení médií a jejich metadat z disku zařízení, tato média poté vrátí jako instanci třídy AVAsset. [8]

AVAssetReaderOutput

Abstraktní rodičovská třída pro AVAssetReaderTrackOutput a AVAssetReaderVideoCompositionOutput. [8]

AVAssetReaderTrackOutput

Objekt této třídy umí číst data média z jedné stopy, ať už se jedná o video nebo audio. [8]

AVAssetReaderVideoCompositionOutput

Objekt této třídy umí číst složená videa z jedé nebo více stop daného média. [8]

AVOutputSettingsAssistant

Používá se pro vytvoření konfigurace pro objekty zapisující média na disk zařízení, jako jsou objekty tříd AVAssetWriter a AVAssetWriterInput. AVOutputSettingsAssistant umí také automaticky vygenerovat nastavení z předvoleb. Tyto předvolby

jsou reprezentovány strukturou AVOutputSettingsPreset, která obsahuje několik předvoleb v podobě statických konstant, které lze předat kontruktoru AVOutputSettingsAssistant. [8]

AVAssetWriter

Objekt této třídy umí zapisovat média (instance třídy AVAsset) společně s jejich metadaty na disk zařízení. [8]

AVAssetWriterInput

Tento objekt zapisuje data média do jedné stopy objektu třídy AVAssetWriter. Lze mít více těchto vstupů pro jeden AVAssetWriter a zapisovat do každé stopy zvlášť. [8]

AVAssetWriterInputPixelBufferAdaptor

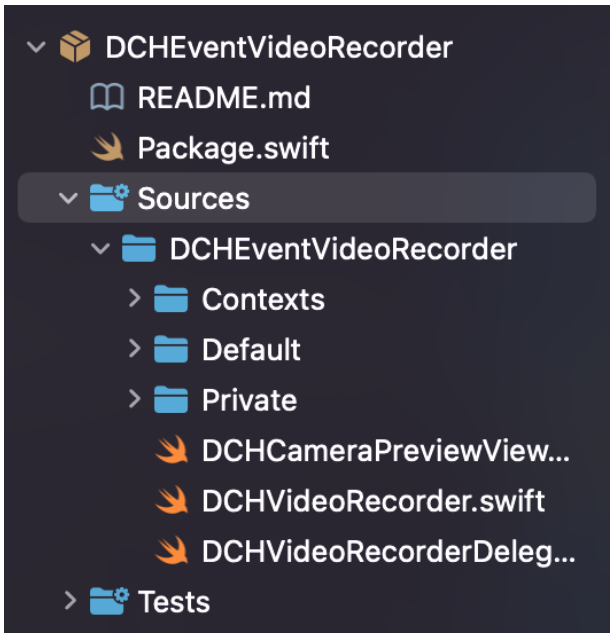
Skrze tento objekt je možné efektivně alokovat snímky videa ve formě pixel bufferů do poolu, ze kterého vstup writeru získává data pro zápis do souboru. [8]

2.7 Swift Package Manager

Jedním z hlavních požadavků na výsledný framework je jeho snadné přidání do nových nebo existujících projektů, z tohoto důvodu jsem se rozhodl ho implementovat v podobě Swift balíčku, s využitím tzv. Swift Package Manageru.

Swift Package Manager, je nástroj, který automatizuje distribuci a správu zdrojových kódů v tzv. balíčcích. Dále umožňuje automaticky stahovat a přidávát závislosti, které balíček vyžaduje. Programovací jazyk Swift rozděluje zdrojové kódy do modulů. Projekty mohou být složeny z jednoho nebo z více těchto modulů. Výhodou modularizace je možnost přepoužití těchto modulů, v případě potřeby, aniž by vývojáři museli požadovanou funkcionalitu implementovat znovu. [30]

Každý balíček je složen ze zdrojových souborů obsažených v adresáři *Source* a souboru manifestu, *Package.swift*, který obsahuje metadata o balíčku [30]. Na obrázku 16 je zobrazena finální struktura frameworku, který jsem implementoval jako balíček. Všimněte si souboru *Package.swift* a adresáře *Source*.



Obrázek 16: Ilustrační struktura balíčku. [35]

Níže je možné vidět obsah souboru *Package.swift*. Zde si všimněte atributu name, který představuje název balíčku, dále sekce platforms, která udává podporované platformy, v našem případě obsahuje pouze jednu hodnotu *.iOS(.v14)*, která udává, že bude balíček podporovat pouze operační systém iOS, konkrétně verzi 14 a novější. Protože se nejedná o spustitelný balíček, ale o knihovnu [1], sekce products obsahuje hodnotu *.library*. Balíček nemá žádné závislosi, sekce *dependencies* je tedy prázdná. Sekce *targets* obsahuje dvě hodnoty *.target*, která představuje vlastní knihovnu a hodnotu *.testTarget*, která představuje Unit testy. V této části si všimněte, že tyto testy jsou závislé na „DCHEventVideoRecorder“, tedy zdrojových souborech knihovny.

[1] Balíček může být jak v podobě spustitelného kódu, tak v podobě knihovny. [30]

```swift
// swift-tools-version: 5.6
// The swift-tools-version declares the minimum version of Swift required to build this package.

import PackageDescription

let package = Package(
    name: "DCHEventVideoRecorder",
    platforms: [
        .iOS(.v14)
    ],
    products: [
        // Products define the executables and libraries a package produces, and make them visible to other packages.
        .library(
            name: "DCHEventVideoRecorder",
            targets: ["DCHEventVideoRecorder"]),
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        // .package(url: /* package url */, from: "1.0.0"),
    ],
    targets: [
        // Targets are the basic building blocks of a package. A target can define a module or a test suite.
        // Targets can depend on other targets in this package, and on products in packages this package depends on.
        .target(
            name: "DCHEventVideoRecorder",
            dependencies: []),
        .testTarget(
            name: "DCHEventVideoRecorderTests",
            dependencies: ["DCHEventVideoRecorder"]),
    ]
)
```

Zdrojový kód 1: Ukázka souboru manifestu *Package.swift*. Zdroj: autor.

2.8 Git

V roce 2005 Linus Torvalds, který je známý jako tvůrce operačního systému Linux, vydal svůj vlastní verzovací systém, který pojmenoval Git. Linus Torvalds v době vývoje operačního systému Linux nenalezl jediný verzovací systém, který by mu vyhovoval pro verzování linuxového jádra, proto se rozhodl vytvořit svůj vlastní. Projekt Git byl vydán jako open-source a navržen tak, aby rychlý, robustní a v neposlední řadě také efektivní. [22], [34]

Narozdíl od dříve používaných verzovacích systémů, mezi které patří CVS (Concurrent Version System) a jeho pozdější náhrada Subversion, je Git, jakožto zástupce DVCS (Distributed Version Control System), distribuovaný. Hlavním rozdílem mezi těmito systémy je ten, že v případě distribuovaného verzovacího systému je historie změn obsažena v pracovní kopii každého vývojáře, kdežto v případě CVS nebo Subversion je tato histori změn pouze na jednom jediném místě. Git je vhodný jak pro vezrování na lokálním stroji, tak i pro spolupráci s ostatními vývojáři přes globální repozitáře. Mezi tyto repozitáře patří například GitHub nebo Bitbucket. [34], [12]

Obrázek 17: Logo verzovacího systému Git. [16]

3 PRAKTICKÁ ČÁST

3.1 Motivace

Představme si situaci, kdy potřebujeme zaznamenat nějaké události v průběhu několika následujících hodin. Pro příklad zvolím časový interval 24 hodin. Pokud bychom chtěli pro každou událost zaznamenat video pomocí systémové kamery v iOS, museli bychom nahrávat celých 24 hodin a následně tento záznam manuálně projít a vyexportovat pouze ty části, které by obsahovaly tyto události. Toto řešení má ale několik zásadních nevýhod. První nevýhodou je skutečnost, že velikost této 24hodinové nahrávky by byla v řádech gigabajtů, což by vedlo k plýtvání pamětí našeho zařízení. Proč bychom ale měli ukládat celý záznam, když nás zajímají pouze události, které mezi sebou mohou mít značný rozestup? I přesto, že bychom uvažovali nekonečně velké uložiště, narazili bychom na další překážku, kterou je následující zpracování videa. Pokud bychom měli projít celý záznam a vyexportovat části s událostmi, zabralo by nám to mnoho času a úsilí. Také by se mohlo stát, že by nám při kontrole takto douhého videa nějaká událost unikla a nebyla by vyexportována.

Cílem této práce je tedy navrhnout a naimplementovat framework, který umožní náhrávat takovéto události bez problémů zmíněných výše. Bude efektivně využívat paměť zařízení a ukládat pouze ty části videa, které obsahují události. Uživatel poté obdrží množinu videí, aniž by musel vynaložit další úsilí.

3.2 Ovládání a konfigurace

Následující protokol DCHVideoRecorder popisuje rozhraní, které jsem navrhl pro ovládání a konfiguraci videorekordéru. Všechny prvky tohoto rozhraní jsou krátce popsány níže.

```swift
public protocol DCHVideoRecorder: AnyObject {

    var currentConfiguration: DCHRecorderConfiguration { get }

    var stateDelegate: DCHVideoRecorderDelegate? { get set }

    var currentState: DCHVideoRecorderState { get }

    func startRecording() throws

    func stopRecording()

    func resumeRecording()

    func resetRecorder()

    func captureSegment() throws

    func changeConfiguration(_ configuration: DCHRecorderConfiguration) throws

    func setupPreview(for view: DCHCameraPreviewView)

    func provideCapturedSegments(
        completion: @escaping (Result<[DCHCapturedVideoSegment], Error>) -> Void
    )
}
```

Zdrojový kód 2: Protokol DCHVideoRecorder. Zdroj: autor.

`var currentConfiguration: DCHRecorderConfiguration { get }`

Tento atribut uchovává aktuální nastavení, které je použito pro nahrávání. Atribut je určen pouze pro čtení. Pokud je potřeba změnit nastavení, je možné použít metodu `func changeConfiguration(_ configuration: DCHRecorderConfiguration) throws`.

`var stateDelegate: DCHVideoRecorderDelegate? { get set }`

Objekt přiřazený tomuto atributu bude dostávat události o změně stavu, stane se delgátem. Objekt musí implementovat rozhraní DCHVideoRecorderDelegate.

`var currentState: DCHVideoRecorderState { get }`

Atribut uchovávající aktuální stav videorekordéru. Stav je reprezentován výčtem DCHVideoRecorderState a může nabývat těchto hodnot: none, recording, paused.

`func startRecording() throws`

Tato metoda zahájí nahrávání videa.

`func stopRecording()`

Tato metoda ukončí nahrávání videa.

`func pauseRecording()`

Tato metoda pozastaví nahravání videa.

`func resumeRecording()`

Tato metoda obnoví nahrávání videa.

`func resumeRecording()`

Pomocí této metody je možné resetovat stav videorekordérů.

`func captureSegment() throws`

Tato metoda je určena k pořízení segmentu videa. Je zavolána událostí z vnějšku.

`func changeConfiguration(_ configuration: DCHRecorderConfiguration) throws`

Metoda určená pro změnu nastavení videorekordéru. Nastavení je možné změnit pouze pokud se nenahrává, v opačném případě dojde k vyvolání výjimky.

`func setupPreview(for view: DCHCameraPreviewView)`

Pomocí této metody se předá view, které bude použito pro živý náhled videa. Toto view musí být potomkem třídy DCHCameraPreviewView.

`func provideCapturedSegments(completion: ...)`

Tato metoda vrátí nahrané segmenty, které jsou nejprve poskládány z jednotlivých snímků do videa.

Rozhraní, které je nutné implementovat, aby bylo možné dostávát události o změně stavu videorekordéru, tedy stát se jeho delegátem.

```
1     public protocol DCHVideoRecorderDelegate: AnyObject {
2
3         func videoRecorderDidChangeState(
4             _ oldState: DCHVideoRecorderState,
5             _ newState: DCHVideoRecorderState
6         )
7     }
```

Zdrojový kód 3: Protokol DCHVideoRecorderDelegate. Zdroj: autor.

`func videoRecorderDidChangeState(_ oldState: ..., _ newState: ...)`

Tato metoda je automaticky volána videorekordérem v případě, že došlo ke změně jeho stavu. Delegát tedy může reagovat na tyto změny.

Tento výčet reprezentuje stav videorekordéru, který může nabývat tří hodnot.

```
1     public enum DCHVideoRecorderState {
2         case none
3         case recording
4         case paused
5     }
```

Zdrojový kód 4: Výčet DCHVideoRecorderState. Zdroj: autor.

`case none`

Nahrávání je zastaveno, nebo ještě nebylo spuštěno.

`case recording`

Probíhá nahávání. Nelze změnit konfiguraci.

`case paused`

Nahrávání je pozastaveno, je možné pokračovat v nahrávání zavoláním metody `func resumeRecording()`.

Struktura reprezentující kompletní nastavení videorekordéru použité pro nahrávání a ukládání videa.

```
1    public struct DCHRecorderConfiguration {
2
3        let recordQuality: DCHVideoQuality
4        let exportSettings: DCHExportSettings
5        let framerate: DCHFramerate
6        let backwardOffset: TimeInterval
7
8        static let `default` = DCHRecorderConfiguration(
9            recordQuality: .medium,
10           exportSettings: DCHExportSettings(
11               quality: .medium,
12               codec: .preferred,
13               format: .mp4
14           ),
15           framerate: ._30fps,
16           backwardOffset: 3.0
17       )
18   }
```

Zdrojový kód 5: Struktura DCHRecorderConfiguration. Zdroj: autor.

`let recordQuality: DCHVideoQuality`

Kvalita, neboli rozlišení použité pro nahrávání.

`let exportSettings: DCHExportSettings`

Nastavení použité pro export video segmentů.

`let framerate: DCHFramerate`

Snímkovací frekvence použita při nahrávání.

`let backwardOffset: TimeInterval`

Tento atribut udává, jak dlouhý bude nahraný segment videa. Jedná se o hodnotu, která udává počet vteřin nahraných před zavoláním metody `func captureSegment() throws`.

`static let `default` = DCHRecorderConfiguration`

Základní konfigurace, která je použita, pokud uživatel nespecifikuje svou vlastní.

Struktura reprezentující nastavení použité při exportování segmentů. Obsahuje kvalitu videa, použitý kodek a formát souboru.

```
1    public struct DCHExportSettings {
2        let quality: DCHVideoQuality
3        let codec: DCHCodec
4        let format: DCHFormat
5    }
```

Zdrojový kód 6: Struktura DCHExportSettings. Zdroj: autor.

`let quality: DCHVideoQuality`

Rozlišení videa použité při exportu. Může nabývat hodnot low, medium, high a ultra.

`let codec: DCHCodec`

Kodek použitý při exportu.

`let format: DCHFormat`

Formát video souboru.

Výčet reprezentující rozlišení videa.

```
1    public enum DCHVideoQuality {
2        case low
3        case medium
4        case high
5        case ultra
6    }
```

Zdrojový kód 7: Výčet DCHVideoQuality. Zdroj: autor.

`case low`

VGA: 640×480

`case medium`

HD: 1280 × 720

`case high`

FHD: 1920 × 1080

`case ultra`

UHD: 3840 × 2160

Snímkovací frekvence použitá pro nahrávání. Udává, kolik snímků je zaznamenáno za vteřinu.

```
1    public enum DCHFramerate: Int {
2        case _24fps = 24
3        case _30fps = 30
4        case _60fps = 60
5    }
```

Zdrojový kód 8: Výčet DCHFramerate. Zdroj: autor.

Kodek použitý pro nahrání a uložení videa. Obsahuje pouze jednu hodnotu „preferred“. Tato hodnota znamená, že bude použit preferovaný kodek pro formát videa.

```
1    public enum DCHCodec {
2        case preferred
3    }
```

Zdrojový kód 9: Výčet DCHCodec. Zdroj: autor.

Formát použitý při nahrávání a uložení videa. Jsou podporovány dva formáty: mp4 a mov.

```
1    public enum DCHFormat {
2        case mp4
3        case mov
4    }
```

Zdrojový kód 10: Výčet DCHFormat. Zdroj: autor.

`case mp4`

MPEG-4

`case mov`

QuickTime

3.3 Návrh a implementace algoritmu

Při vývoji mě napadlo hned několik možných řešení, zmíním zde pouze některá. Všechna tato řešení měla značné nedostatky, kvůli kterým se nedala použít. Tyto nedostatky zahrnovaly neefektivní využití operační paměti, vysoké vytížení procesoru a nedostatečnou přesnost nahraných segmentů videa. Pouze jeden způsob se mi podařil implementovat s efektivním využitím zdrojů a požadovanou kvalitou výstupu.

3.3.1 Nahrávání krátkých úseků

Tato metoda spočívala v nahrávání krátkých úseků videa (pomocí AVAssetWriter) jdoucích po sobě tak, aby spojení těchto úseků nebylo poznat. Výhodou této metody by bylo snadné mazání starších úseků z disku zařízení, protože lze snadno zjistit jejich datum vytvoření z metadat. Toto řešení se mi ale nepodařilo implementovat, protože spuštění a zastavení nahrávání není okamžité, proto na sebe úseky videa nenavazovaly. Mezi těmito úseky vždy chybělo několik snímků. Tento problém jsem se snažil vyřešit vytvořením dvou instancí třídy AVAssetWriter, aby vždy jedna instance nahrávala a druhá začala nahrávat chvíli před tím, než předchozí nahrávání ukončí. Následně jsem narazil na problém se synchronizací těchto dvou instancí, protože zpoždění nahrvání je pokažné jiné. Nedá se tedy přesně určit čas, kdy by měla následující instance začít nahrávat.

3.3.2 Cyklická fronta (operační paměť)

Narozdíl od předešlé metody nedocházelo k ukládání do souboru, ale k ukládání jednotlivých snímků do datové struktury zvané jako cyklická fronta. Velikost této fronty se počítá jako požadovaná délka segmentu v sekundách vynásobená počtem snímků za sekundu. To

znamená, že v případě 5sekundových segmentů s 30 snímky za sekundu by byla velikost 5 * 30 = 150 prvků. Každý tento prvek představuje jeden snímek videa. Při přetečení této velikosti dojde k přepisování nejstarších snímků od začátku. Tímto je docíleno toho, že fronta obsahuje posledních 150 snímků nahraných kamerou. Při pořízení segmentu je možné vytvořit dočastnou kopii této fronty a poskládat z nich video od nejstarších snímků po nejnovější. Toto video poté uložit na disk zařízení.

Toto řešení se mi jevilo jako optimální, ale i zde jsem narazil na překážku. Každý jednotlivý zachycený snímek má před uložením na disk několik megabajtů, dokonce i při nastavení menšího rozlišení videa a nižší snímkovací frekvence. Ke kompresi dochází až při zápisu na disk, čímž jsem narazil na problém s využitím operační paměti. Pokud by chtěl uživatel nahrávat s výšší kvalitou nebo delší segmenty, docházelo by k pádu aplikace z důvodu nedostatku operační paměti.

3.3.3 Cyklická fronta (souborový systém)

Tato metoda vychází z metody předešlé, tedy myšlenky využít cyklickou frontu, ale s tím rozdílem, že nedochází k ukládání snímků do operační paměti, ale na disk zařízení. Každý jednotlivý snímek je uložen do souborového systému jako fotografie. V operační paměti je pouze potřeba držet cesty k těmto souborům, pro případnou další manimulaci s nimi a informaci, zda byl snímek použit v některém segmentu videa. Pokud je snímek použit, je nutné ho na disku ponechat, v opačném případě dojde k jeho smazání, čímž je docíleno efektivního využití paměti. Při pořízení segmetu je vytvořena struktura, která obsahuje cesty do souborového systému, aby bylo poté možné tyto segmenty poskládát z fotografií do výsledného videa.

Tento způsob nevykazoval další nedostatky, proto jsem ho zvolil pro výslednou implementaci frameworku.

Následující struktura *DCHPixelBuffer* představuje pořízený snímek. Tato struktura zapouzdřuje obrazová data v podobě CVPixelBufferu a atribut date, který představuje čas pořízení snímku. Tento čas je poté použit jako unikátní identifikátor snímku uloženého na disk zařízení, aby bylo možné tento soubor jednoznačně idetifikovat.

```
struct DCHPixelBuffer {
    let date: Date
    let pixelBuffer: CVPixelBuffer
}
```

Zdrojový kód 11: Struktura DCHPixelBuffer. Zdroj: autor.

`let date: Date`

Atribut *date* představuje čas pořízení snímku. Tento čas je použit jako unikátní identifikátor, ze kterého je poté tvořen název souboru uloženého na disk. Tímto je zajištěna unikátnost názvu.

`let pixelBuffer: CVPixelBuffer`

Atribut *pixelBuffer* obsahuje vlastní obrazová data zaznamenaná při pořízení snímku.

Cyklická fronta je reprezentována protokolem DCHRecordingBuffer. Tento protokol poté implementuje třída DCHDefaultRecordingBuffer, která implementuje jeho chování. Jedná se o třídu, která se stará o ukládání snímků na disk zařízení a také o mazání již nepotřebných snímků.

```
protocol DCHRecordingBuffer: AnyObject {
    func insert(_ buffer: DCHPixelBuffer)
    func capture() -> [String]
    func isEmpty() -> Bool
    func isReady() -> Bool
    func directory() -> String
    func release() throws
    var actualSize: Int { get }
    var size: Int { get }
}
```

Zdrojový kód 12: Protokol DCHRecordingBuffer. Zdroj: autor.

`func insert(_ buffer: DCHPixelBuffer)`

Tato metoda vytvoří unikátní název souboru s využitím již zmíněného atributu date. Dále převede obrazová data z *CVPixelBuffer* na data obrázku, která poté zapíše do

souboru s vytvořeným názvem. Pokud je přesáhnuta maximální velikost bufferu, jsou odstaněny nejstarší snímky, které nejsou použity v některém ze segmentů.

`func capture() -> [String]`

Metoda *capture* je volána při pořízení segmetu videa. Účelem této metody je poskytnout všechny cesty k souborům (fotografiím), které patří do daného segmentu.

`func isEmpty() -> Bool`

Pokud buffer neobsahuje žádné snímky, tedy soubory fotografiií, vrátí hodnotu true. V opačném případě vrátí hodnotu false.

`func isReady() -> Bool`

Pokud je buffer naplněn, tedy počet snímků je rovných maximální velikosti bufferu, vrátí hodnotu true. V opačném případě vrátí hodnotu false.

`func directory() -> String`

Vrátí cestu k adresáři, ve kterém jsou uloženy všechny snímky bufferu.

`func release() throws`

Resetuje stav bufferu a odstraní všechny uložené snímky z disku zařízení.

`var actualSize: Int { get }`

Aktuální počet snímků (souborů) uložených na disk.

`var size: Int { get }`

Maximální velikost bufferu. Tuto velikost lze vypočítat z délky segmentu ve vteřinách a počtu snímků za vteřinu, tedy vzorcem: *duration * framerate*.

DCHSegmentBuffer je protokol, který implementuje třída DCHDefaultSegmentBuffer, která reprezentuje konkrétní segment videa. Třída obsahuje základní informace o segmentu a pomocné metody.

`func items() -> [CVPixelBuffer]`

Načte všechny soubory pro tento segment, které převede na CVPixelBuffer. Poté vrátí pole těchto bufferů.

```
1     protocol DCHSegmentBuffer: AnyObject {
2         func items() -> [CVPixelBuffer]
3         func item(at index: Int) -> CVPixelBuffer?
4         func isEmpty() -> Bool
5         func isReady() -> Bool
6         var index: Int { get }
7         var frameFilePaths: [String] { get }
8         var dateOfCreation: Date { get }
9         var size: Int { get }
10    }
```

Zdrojový kód 13: Protokol DCHSegmentBuffer. Zdroj: autor.

`func item(at index: Int) -> CVPixelBuffer?`

Načte soubor obrázku z disku, který převede na CVPixelBuffer, tento buffer poté vrátí.

`func isEmpty() -> Bool`

Vrátí hodonotu true, pokud segment neobsahuje žádné snímky.

`func isReady() -> Bool`

Pokud jsou vsechny potřebné snímky uloženy na disku, vrátí hodnotu true. V opačném případě vrátí hodnotu false.

`var index: Int { get }`

Index (identifikátor) segmentu.

`var frameFilePaths: [String] { get }`

Všechny cesty k souborům (fotografiím), ze kterých je segment složen.

`var dateOfCreation: Date { get }`

Datum vytvoření segmentu.

`var size: Int { get }`

Počet snímků (fotografií), které jsou obsaženy v segmentu.

ZÁVĚR

Hlavním cílem této práce bylo implementovat framework, který umožní nahrávat krátké úseky videí pro sledované události. Před touto vlastní implementací bylo potřeba nejdříve navrhnout a porovnat několik možných řešení, dále vybrat nejlepší z nich, tedy takové, které bude nejlépe splňovat požadavky na využití systémových prostředků a přívětivost pro uživatele. Framework měl být implementován pro mobilní platformu iOS, bylo tedy nutné vybrat kompatibilní technologie s touto platformou.

Teoretická část práce definovala pojem framework, zmínila některé z jeho možných variant a popsala rozdíly mezi frameworky a knihovnami. Jejím dalším cílem bylo představit a popsat použité technologie. Čtenář se seznámil s operačními systémy iOS a macOS, které byly popsány hned v prvních kapitolách této práce. Tyto kapitoly obsahovaly mimo jiné historická fakta a seznam verzí těchto operačních systémů. Mezi dalšími popsanými technologiemi byly například programovací jazyk Swift, vývojové prostředí Xcode a framewrok AVFoundation.

Hlavní náplní praktické části bylo navrhnout několik řešení a jedno vybrané implementovat. Byly zde popsány tři návrhy řešení, společně s jejich výhodami a nevýhodami. Řešení vybrané pro implementaci bylo popsáno detailněji. Tato část také obsahovala popis rozhraní a konfigurace frameworku, tedy hlavních prvků, se kterými bude vývojář používající tento framework pracovat.

I přesto, že při návrhu a vývoji došlo k nemálo komplikacím, podařilo se framework úspěšně implementovat se všemi požadavky.

Framework je ve fázi, kdy je plně funkční, ale do budoucna by bylo vhodné přidat některé další funkcionality, jako je například možnost nahrávání audia nebo vybrání konkrétní kamery zařízení.

POUŽITÁ LITERATURA

[1] About Swift. *Swift.org* [online]. Cupertino, Kalifornie: Apple, 2022 [cit. 2022-03-14]. Dostupné z: `https://www.swift.org/about/`.

[2] Apple. *Macworld San Francisco 2007 Keynote Address* [přednáška]. Dostupné z: `https://podcasts.apple.com/cz/podcast/apple-events-video/id275834665?i=1000026524322`.

[3] Apple. iOS 15. In: *Apple* [online]. Cupertino, Kalifornie: Apple, 2021 [cit. 2022-03-29]. Dostupné z: `https://www.apple.com/cz/ios/ios-15/`.

[4] Apple. macOS. In: *Logopedia - Fandom* [online]. San Francisco, Kalifornie, USA: Fandom, 2022 [cit. 2022-04-03]. Dostupné z: `https://logos.fandom.com/wiki/MacOS`.

[5] Apple. macOS Monterey. In: *Logopedia - Fandom* [online]. San Francisco, Kalifornie, USA: Fandom, 2022 [cit. 2022-04-03]. Dostupné z: `https://logos.fandom.com/wiki/MacOS`.

[6] Apple. Swift. In: *Apple Developer* [online]. [cit. 2022-03-29]. Dostupné z: `https://developer.apple.com/swift/resources/`.

[7] Apple's Mac OS X Is Dead. Long Live MacOS. *WIRED: The Latest in Technology, Science, Culture and Business* [online]. Condé Nast: Condé Nast, 2022, 13. 6. 2016 [cit. 2022-04-03]. ISSN 1078-3148. Dostupné z: `https://www.wired.com/2016/06/apple-os-x-dead-long-live-macos/`.

[8] *AVFoundation* [online]. Cupertino, Kalifornie: Apple, 2022 [cit. 2022-06-30]. Dostupné z: `https://developer.apple.com/documentation/avfoundation`.

[9] Apple. *WWDC10 Keynote* [přednáška]. Dostupné z: `https://podcasts.apple.com/cz/podcast/apple-events-video/id275834665?i=1000083922689`.

[10] Apple. Xcode icon. In: *Logopedia - Fandom* [online].

San Francisco, Kalifornie: Fandom, 2022 [cit. 2022-03-29].
Dostupné z: https://logos.fandom.com/wiki/Xcode.

[11] BEZNUTZ, Feli. Swift evolution. In: *DieHimmelstraeumerin* [online].
Nuremberg, Bavaria, 2021, 15. 3. 2021 [cit. 2022-03-29]. Dostupné z: https://fbernutz.github.io/posts/summaries-ios-interview-topics/#swift-evolutions-interview-topics/#swift-evolution.

[12] BRABEC, Stanislav. CVS pro každého (1): základy. *Root.cz: Informace nejen ze světa Linuxu* [online]. Praha: Internet Info, 2022, 6. 5. 2002 [cit. 2022-04-20].
Dostupné z: https://www.root.cz/clanky/cvs-pro-kazdeho-zaklady/.

[13] CARNES, Beau. How to Build a Hackintosh: Install MacOS Big Sur on a PC Using OpenCore. *FreeCodeCamp* [online].
San Francisco, Kalifornie: freeCodeCamp, 2022, 3. 6. 2021 [cit. 2022-04-03].
Dostupné z: https://www.freecodecamp.org/news/build-a-hackintosh/.

[14] COSTELLO, Sam. The History of iOS, from Version 1.0 to 16.0. *Lifewire* [online].
New York, USA: Dotdash Meredith, 2022, 26. 7. 2022 [cit. 2022-07-30].
Dostupné z: https://www.lifewire.com/ios-versions-4147730.

[15] GILL, Bobby. SwiftUI vs UIKit in 2022, Which Framework Should Your App Use?. *Blue Label Labs* [online]. New York, USA: Blue Label Labs, 2022, 11. 1. 2022 [cit. 2022-04-03].
Dostupné z: https://www.bluelabellabs.com/blog/swiftui-vs-uikit/.

[16] *Git* [online]. Git, 2022 [cit. 2022-04-20]. Dostupné z: https://git-scm.com/.

[17] GREBEŇ, David. Kompletní historie OS X: Od první verze až po El Capitan. *Letem světem Applem: Apple magazín* [online].
Brno: Text Factory, 2022, 13. 3. 2016 [cit. 2022-04-03]. Dostupné z: https://www.letemsvetemapplem.eu/2016/03/13/kompletni-historie-os-x/.

[18] HASLAM, Karen. Every macOS and Mac OS X version–including the latest update. *Macworld* [online]. Boston, USA: Foundry, 2022,
22. 7. 2022 [cit. 2022-07-30]. Dostupné z: https://www.macworld.co.uk/feature/os-x-macos-versions-3662757/.

[19] HOFFMAN, Jon. *Mastering Swift 5.3: Upgrade your knowledge and become an expert in the latest version of the Swift programming language*. 6. Birmingham: Packt Publishing, 2020. ISBN 978-1-80056-215-8.

[20] IOS. *AppleInsider: Apple News, Rumors, Reviews, Prices & Deals* [online]. New York: Quiller Media, 2022 [cit. 2022-03-24]. Dostupné z: `https://appleinsider.com/inside/ios`.

[21] KAUSHIK, Neha. 5 Preferred programming languages for game development. *TechGig: Largest Tech Community | Tech News & Hackathons* [online]. India: Times Business Solutions, 2022, 23. 8. 2021 [cit. 2022-04-03]. Dostupné z: `https://content.techgig.com/technology/5-preferred-programming-languages-for-game-development/articleshow/84677550.cms`.

[22] KNOTT, Matthew. *Beginning Xcode: Swift 3 Edition*. 1. New York, USA: Apress, 2016. ISBN 978-1-4302-5004-3.

[23] LACKO, Luboslav. *Vývoj aplikací pro iOS. Přeložil Martin HERODEK*. 1. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.

[24] MacOS Monterey. *Apple Developer* [online]. Cupertino, Kalifornie: Apple, 2022 [cit. 2022-08-09]. Dostupné z: `https://www.apple.com/cz/macos/monterey/`.

[25] MAZMANYAN, Arsen. Game Engines vs Game Frameworks. *DEV Community* online]. DEV Community, 2022, 11. 6. 2021 [cit. 2022-04-03]. Dostupné z: `https://dev.to/arsenmazmanyan1104/game-engines-vs-game-frameworks-1icc`.

[26] MCCUNE, Bob. *Learning AV Foundation: A Hands-on Guide to Mastering the AV Foundation Framework* [online]. 1. Boston, USA: Addison-Wesley, 2014 [cit. 2022-06-30]. ISBN 978-0-321-96180-8. Dostupné z: `informit.com/aw`.

[27] Most Popular Mobile App Development Frameworks For App Developers. *Technostacks: A Global IT Solution Company That Empowers Ideas & Future* [online]. Ahmedabad, India: Technostacks, 2022, 8. 9. 2021 [cit. 2022-03-30]. Dostupné z: `https:`

//technostacks.com/blog/mobile-app-development-frameworks.

[28] MULONGO, Cleophas. 5 Best Frameworks For Desktop Application Development [2022]. *Technotification: Tech News, Guides And More* [online]. Udaipur, Rajasthan, India: Technotification, 2022, 14. 3. 2022 [cit. 2022-03-30]. Dostupné z: https://www.technotification.com/2019/02/desktop-application-development-frameworks.html.

[29] New Playgrounds Part 2 - Sources. *Apple Developer* [online]. Cupertino, Kalifornie: Apple, 2022, 17. 3. 2015 [cit. 2022-04-12]. Dostupné z: https://developer.apple.com/swift/blog/?id=26.

[30] Package Manager. *Swift.org* [online]. Cupertino, Kalifornie: Apple, 2022 [cit. 2022-07-24]. Dostupné z: https://www.swift.org/package-manager/.

[31] ROY, Sandip. The Difference Between a Framework and a Library. *Baeldung* [online]. Bucharest, Romania: Baeldung, 2022, 1. 1. 2022 [cit. 2022-03-30]. Dostupné z: https://www.baeldung.com/cs/framework-vs-library.

[32] Swift. *Apple Developer* [online]. Cupertino, Kalifornie: Apple, 2022 [cit. 2022-03-14]. Dostupné z: https://developer.apple.com/swift/.

[33] What is a Framework? Why We Use Software Frameworks. *Code Institute: Learn at Your Own Pace* [online]. Dublin, Irsko: Code Institute, 2022 [cit. 2022-03-30]. Dostupné z: https://codeinstitute.net/global/blog/what-is-a-framework/.

[34] What is Git. *Atlassian: Software Development and Collaboration Tools* [online]. Sydney, Austrálie: Atlassian, 2022 [cit. 2022-04-20]. Dostupné z: https://www.atlassian.com/git/tutorials/what-is-git.

[35] Xcode. *Apple Developer* [online]. Cupertino, Kalifornie: Apple, 2022 [cit. 2022-08-09]. Dostupné z: https://developer.apple.com/xcode/.

[36] Xcode. *AppleInsider: Apple News, Rumors, Reviews, Prices & Deals* [online]. New York: Quiller Media, 2022 [cit. 2022-03-24]. Dostupné z: https://appleinsider.com/inside/xcode.

SEZNAM PŘÍLOH

PŘÍLOHA A – ZDROJOVÉ KÓDY

Příloha obsahuje kompletní zdrojové soubory frameworku v podobě Swift balíčku.