МІНІСТЕРСТВО ОСВІТИ НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Завідувач кафедри,
д.е.н., доц.
_____ С.І. Левицький

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

МОБІЛЬНИЙ ДОДАТОК НА iOS ДЛЯ ВІДЕО ТА АУДІО КОНВЕРТУВАННЯ

Виконав

ст. гр. ІПЗ – 218 _____ О. В. Жирова

Керівник

к.е.н., доц. _____ О. В. Шляга

Запоріжжя

2023

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри,
д.е.н., доц.
_____ С.І. Левицький
___.___._____ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ
студенту гр. ІПЗ-218 ,
спеціальності 121 - «Інженерія програмного забезпечення»
Жировій Олександрі Володимирівні

1. Тема: Мобільний додаток на iOS для відео та аудіо конвертування
затверджена наказом № 02-10 від 27 січня 2023 р.
2. Термін здачі студентом закінченої роботи: 12 червня 2023 р.
3. Перелік питань, що підлягають розробці:
1. Здійснити огляд проблематики роботи у мережі Інтернет.
2. Розглянути варіанти та технології щодо оптимізації механізмів у подібних додатках
3. Виконати огляд та порівняння аналогів
4. Здійснити обґрунтований вибір використання технологій для розробки додатку
5. Здійснити проектування та програмування проекту
6. Розробити алгоритм coding → decoding

7. Виконання операцій життєвого циклу програмного забезпечення щодо створеного проекту

8. Оформити звіт за результатами роботи.

4. Календарний графік підготовки кваліфікаційної бакалаврської роботи.

№ етапу	Зміст	Терміни виконання	Готовність по графіку %, підпис керівника	Підпис керівника про повну готовність етапу, дата
1	Формулювання (корегування) теми кваліфікаційної бакалаврської роботи та збір практичного матеріалу за темою	16.01.23-11.02.23		
2	**І атестація** І розділ кваліфікаційної бакалаврської роботи	27.03.23-31.03.23		
3	**ІІ атестація** ІІ розділ кваліфікаційної бакалаврської роботи	24.04.23-28.04.23		
4	**ІІІ атестація** ІІІ розділ кваліфікаційної бакалаврської роботи, висновки та рекомендації, додатки, реферат	22.05.23-26.05.23		
5	Перевірка кваліфікаційної бакалаврської роботи на оригінальність	15.05.23-12.06.23		
6	Доопрацювання кваліфікаційної бакалаврської роботи, підготовка презентації, отримання відгуку керівника і рецензії	29.05.23-12.06.23		
7	**Попередній захист кваліфікаційної бакалаврської роботи**	**12.06.23-18.06.23**		
8	Подача кваліфікаційної бакалаврської роботи на кафедру	за 3 дні до захисту		
9	**Захист кваліфікаційної бакалаврської роботи**	**19.06.23-24.06.23**		

Дата видачі завдання: ___.___._____ р.

Керівник кваліфікаційної

бакалаврської роботи _____ О В. Шляга

Завдання отримав до виконання _____ О. В. Жирова

РЕФЕРАТ

Бакалаврська дипломна робота містить 95 сторінок, 34 рисунка, 3 таблиці, 12 використаних джерел.

Метою розробки є створення мобільного додатку під операційну систему iOS для відео та аудіо конвертування.

Об’єктом дослідження є – сучасний додаток для відео та аудіо конвертування.

Предметом дослідження є конвертер для обробки відео та аудіо.

Здійснено детальний огляд предметної області, сучасних аналогів, таких як: VCT Video Converter, Media Converter, PlayerXtreme. Мною виявлено, що саме розробка відео та аудіо конвертера є доцільною та потрібною у наш час. Проект реалізовано у виді зручного конвертера на мобільному пристрої, з використанням мови програмування Swift та бібліотеки CocoaPods.

Отриманий програмний продукт є для користувача простим у використанні та гнучким у налаштуванні.

SWIFT, VIDEO, AUDIO, CONVERTER, COCOAPODS, IOS, DECODER

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ТЕРМІНІВ, ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Скорочення	Слово/Словосполучення	Умови використання
DVD	Digital Versatile Disc	у тексті
PDF	Portable Document Format	у тексті
ОС	Operating System	у тексті
Android	Mobile Operating System	у тексті
iOS	Mobile Operating System	у тексті
UI	User Interface	у тексті
UX	User Experience	у тексті
WWDC	Worldwide Developers Conference	у тексті
IDE	Integrated development environment	у тексті
IB	Interface Builder	у тексті
GPU	Graphics Processing Unit	у тексті
MVC	Modal View Controller	у тексті
UML	Unified Modeling Language	у тексті
LLVM	Low Level Virtual Machine	у тексті
AVI	Audio Video Interleave	у тексті
UNIX	Computer Operating System	у тексті

ВСТУП

У наш час робота з комп’ютером чи смартфоном набирає великої популярності. Тому в мережі «Internet» існує безліч програм для обробки відео та аудіо файлів, які сьогодні набувають особливої актуальності. Наприклад, стискання відео та аудіо файлів, змінивши формат.

Практично кожен з нас хоча би раз стикався з проблемою, що файли неможливо відтворити.

Перевагою мобільного застосування є те, що він простий і інтуїтивно зрозумілий для користувача.

Цей додаток був створений тому, що така проблема як, початковий розмір файлу може бути занадто великим у зв’язку з високою якістю. І такі файли не завжди відкриваються, і їх складно кудись переслати. Прикладів багато, тому це і стало причиною створення додатку.

Створення додатку включає наступні кроки:

- проектування та створення архітектури додатку;
- реалізація роботи з відео та аудіо файлами;
- функціональне та модульне тестування додатку;

Також часто користувачі потребують таких функцій, як:

- прибрати аудіо доріжку (замінити, зберегти окремо, відокремити);
- зміна форматів аудіо та відео;
- зменшення якості відео або зробити мультимедіа файл доступним для програвання та інших пристроях та інших ОС.

Метою розробки є створення мобільного додатку під операційну систему iOS для відео та аудіо конвертування.

Об’єктом дослідження є сучасний додаток для відео та аудіо конвертування.

Предметом дослідження є конвертер для обробки відео та аудіо.

Здійснено детальний огляд предметної області, сучасних аналогів, таких як: VCT Video Converter, Media Converter, PlayerXtreme. Мною виявлено, що саме розробка відео та аудіо конвертера є доцільною та потрібною у наш час. Проект реалізовано у виді зручного конвертера на мобільному пристрої, з використанням мови програмування Swift та бібліотеки CocoaPods.

Структура роботи включає …

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

Кожен власник техніки Apple, знає, що не всі фільми та ролики можна буде переглянути з коробки, потрібно обзавестись повнофункціональним плеєром або відеоконвертером.

На будь-якому гаджеті з яблучком, будь це плеер iPod, iPhone, iPad чи комп'ютер Mac, можна відтворити ролики в таких форматах: AVI, MP4, MOV, MHV.

Однак крім формату потрібно дотримуватися відповідного кодування. Техніка Apple зможе "подружитися" з кодеками формату H.264, MJPEG, MPEG-4, при тому, що відео повинно мати бітрейт і дозвіл, а якщо не так, то ролики будуть відтворюватися з лагами.

Виявляється, користувач повинен використовувати занадто багато параметрів, для конвертації, щоб у підсумку отримати відповідні файли.

Конвертація відео – це перетворення вихідного відео на формат, який підходить для відтворення на певних пристроях або онлайн-ресурсах. Для конвертації відео використовуються спеціальні програми – конвертери чи відеоредактори.

Щоб зрозуміти мету та задачі, які повинні виконувати додаток, треба розібратися, що таке відео/аудіо конвертер.

Різні способи відтворення та записування відео та створюють сотні способів його кодування. І іноді неможливо переглянути відео лише тому, що воно знаходиться у неправильному форматі. На щастя, користувачам не доведеться ходити в магазин в пошуках необхідного їм конвертера. Створені відео/аудіо додатки є 100 % універсальним рішенням для конвертації відео, практично будь-яких форматів.

Якщо ви хочете перетворити DVD-диск на AVI-файл або відтворити QuickTime фільми на пристрої Android, iPhone і подібних, відео конвертер

відмінно виконає свою роботу. Завдяки вбудованій підтримці 4-ох і більше форматів, можливості створення відео для різних медіа-плеєрів та безлічі інших можливостей, мій відео конвертер, безумовно, зможе виконати всі ці завдання.

Розроблений мною додаток (відео та аудіо конвертер), надає цілий ряд зручних інструментів, таки як:

- конвертація в інші формати;
- зберігання відео з сайтів;
- обрізка відео та аудіо доріжки;
- зменшення або збільшення якості відео.

Конвертер дозволяє набагато більше, ніж просто конвертувати файли практично будь-яких відео форматів: DVD, AVI, QuickTime, MPEG та десятки інших. На додаток до своєї головної мети програма дозволяє розбивати відео на частини або вирізати з нього непотрібні файли.

Підтримка багатопроцесорності та оптимізовані алгоритми дозволяють виконувати конвертацію швидко та якісно.

Сьогодні можна без проблем знайти платні та безкоштовні додатки, які будуть відтворювати відеофайли будь-яких форматів як на iPhone так і на Mac. Однак для повноцінного кросплатформеного рішення до цих пір не існує. Розробники хочуть і випускають програми з однаковими іконками і назвою, але працюють вони на різних гаджетах по-різному.

Є мінімум 3 причини для того, щоб конвертувати відео в форматі, підтримувані технікою Apple.

По-перше, буде однотипна домашня колекція роликів, кожен з яких легко відтворити на будь-якому гаджеті Apple. Працювати все буде просто і ясно, підключив до медіатеки і вже можно дивитись доступну запис.

По-друге, достатньо один раз розібратися з конвертером і не шукати відповідних плеєрів для MacOS, iOS, tvOS. Всі вони працюють по-різному, одні додатки змінюють на інші і приходиться періодично шукати заміну покинутому проекту.

По-третє, вся техніка Apple оптимізована для роботи зі стандартними додатками.

Системний плеєр і браузер завжди працюють швидше і стабільніше, а головне потребують менше ресурсів смартфона, планшета і комп'ютера. Це дозволить гаджету працювати більше, ніж при використанні інших утиліт із мережі.

1.2 Актуальність застосунку

Проблема не читання формату відео знайома багатьом власникам сучасних гаджетів. А враховуючи, що популярність смартфонів з кожним роком зростає, це завдання стає актуальним для багатьох тисяч користувачів.

Вирішити її з мінімальними витратами часу та сил допоможуть спеціальні програми – конвертери відео та аудіо.

У мережі можна знайти чимало як платних, і безкоштовних конверторів. Усі вони поділяються на такі основні види:

– конвертори для мобільних та портативних пристроїв - як правило, налаштовані під конкретні типи апаратів та мають не дуже широку функціональність;

– програми з режимами конвертації, що настроюються - призначені, найчастіше, для нестандартних пристроїв і форматів відео. Мають розширений функціонал та досить складний інтерфейс;

– спеціалізовані програми – підходять для конвертації конкретних форматів аудіо та відео. Завдяки вузької спеціалізації вдається максимально спростити інтерфейс цих програм.

На ринку є безліч безкоштовних відео чи аудіо конвертерів, але багато хто з них не є повністю безкоштовними. Більшість безкоштовних програм, а точніше, відео конвертерів мають обмеження, наприклад, вони дозволяють конвертувати лише третину вхідного відеофайлу або в центр екрану додається

дратівливий водяний знак. І навіть у таких додатків є велика аудиторія, згідно з проведених мною досліджень у AppStore.

У 2023 році я можу точно сказати, що відео та аудіо конвертери є актуальними додатками.

1.3 Аналоги подібних додатків

Розглянемо аналоги подібрних додатків для відео та аудіо конвертування.

VCT Video Converter (рис. 1.1)

Функціональний конвертер відео, здатний перетворювати ролики у різні формати відео: MP4, AVI, MKV, 3GP і багато інших. Конвертер носить умовно-безкоштовний характер.

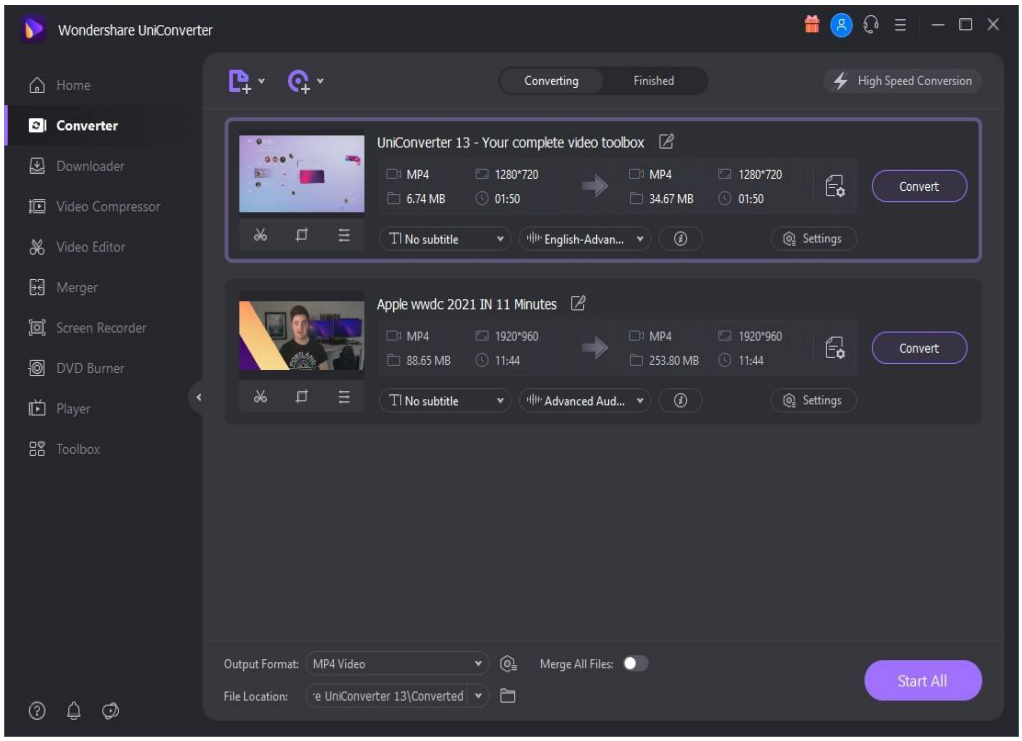

Рисунок 1.1 – Скриншот VCT Video Converter

Media Converter – видео в mp3 (рис. 1.2)

Media Converter може конвертувати практично будь-які відеофайли та аудіофайли. Вбудований універсальний відео плеєр, може відтворювати будь-який формат аудіо та відео.

Media Converter конвертує відео, mp4 m4a to mp3.

Його особливостями є:

– вилучення аудіо: Витягує аудіо з відео, Вихідний формат: MP3, M4A, OGG, WAV, FLAC, WMA, AIFF, CAF, ADX, AC3, AAC, M4R;

– перетворення відео форматів, вихідний формат: MP4, MOV, 3GP, 3G2, ASF, MKV, VOB, MPEG, WMV, FLV, AVI;

– стиснення відео: Обрізання відео, кліп-відео;

– перетворення аудіоформатів, Вихідний формат: MP3, M4A, OGG, WAV, FLAC, WMA, AIFF, CAF, ADX, AC3, AAC, M4R;

– універсальний відеоплеєр, підтримка всіх форматів: mp4, avi, mkv, rmvb, flv, wmv, wav, 3gp, mov, swf, mpeg, mpg, vob, m4v, rm, m2ts, asf, webm, asf, mpg, dat, ts, asx, mp3, f4v, ogv, divx, dv, gxf, m2p, mpeg1, mpeg2, mpeg4, mts, mxf, ogm, qt, wm;

– відкриті zip, rar, 7z та інші стислі файли.

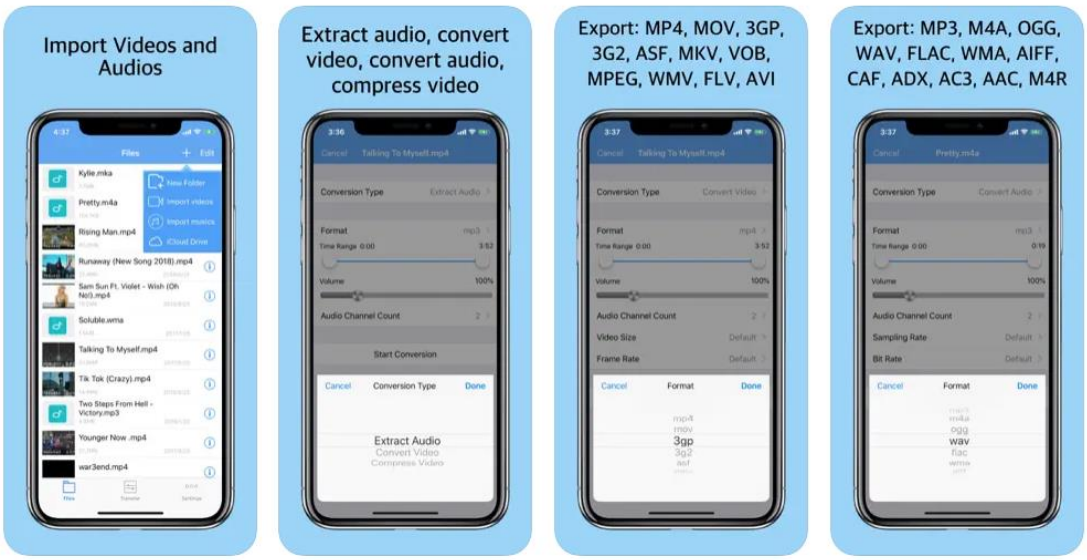

Рисунок 1.2 – Скриншот Media Converter

PlayerXtreme (рис. 1.3)

Змодельована з ідеєю про те, що відео та фільми можуть бути легкими для відтворення та можливими на всіх пристроях удома, включаючи iPhone, PlayXtreme може відтворювати та транслювати майже будь-який тип відео, наприклад MP4, AVI, MKV, MP3 тощо, музику чи фото у форматі. без конвертації та відображає весь вміст у зручному для навігації інтерфейсі. Елементи керування безперебійно працюють у всій програмі, а панель пошуку може легко знайти потрібне відео або файли. Крім того, він включає в себе легку організацію ваших мультимедійних файлів, щоб усе було відсортовано під час масового завантаження дюжини файлів. Потрібно завантажити та встановити, і користувач зможе переглядати відео на своєму iPhone, коли захоче.

Рисунок 1.3 – Скриншот PlayerXtreme

Video Converter and Media File (рис. 1.4)

Video Converter може конвертувати MP3, AVI, MPG, GIF, WMV, MP4 та багато іншого.

Відео конвертер (MP3, AVI, MPG, GIF, FLV, WMV, MP4):

– найкращий конвертер файлів:

перетворювати зображення та звуки в інші формати за допомогою ряду конвертерів файлів і форматів: конвертер PDF, конвертер зображень і фотографій і конвертер MP3.

– ідеально підходить для роботи

Ідентифікує, розпізнає та перетворює будь-які відео, записи, зображення. Він також модифікує та конвертує ваше улюблене відео та змінює розширення на потрібний формат, надсилає готові подкасти прямо з відео конвертера.

– ділитися своїми файлами з іншими інструментами та програмами

Користувач може ділитися перетвореними документами з будь-якою іншою програмою обміну повідомленнями, поштою чи соціальною мережею. Користувач може пов'язувати свої роботи з PDF Converter, Image and Photo Converter і MP3 Converter і закінчити заповнення програми за допомогою найкращих інструментів.

– Оптимізація ресурсів

Оптимізація програми дозволяє економити акумулятор і ресурси під час процесу конвертації документів у різні формати. Виконувати перетворення у фоновому режимі та отримувати сповіщення, коли перетворення буде завершено.

– Формати, які підтримуються Video Converter (MP3, AVI, MPG, GIF, FLV, WMV, MP4):

Конвертер відео в MP3: AVI MPG GIF FLV WMV MP4 може працювати з такими форматами: MP3, AVI, MPG, GIF, FLV, WMV, MP4.

– Простий і легкий у використанні інтерфейс

Video Converter дуже простий у використанні, можливо миттєво конвертувати свої відеофайли у такі формати, як: MP3, AVI, MPG, GIF, FLV, WMV, MP4 та багато іншого.

Перетворювати відео у формат MP3, а фільми та зображення – у будь-який формат, щоб він завжди був сумісний із Video Converter. Покращує

експорт ваших улюблених зображень і відео. Клієнт може мати своє відео в потрібному форматі.

Перетворення відео в MP3 та інші, записи та зображення в інші формати, створення відеофайлів з інших файлів зображень і відеозаписів аудіо є реальністю за допомогою Video Converter, перетворення зображень і створення нових відео в інших форматах легко та швидко.

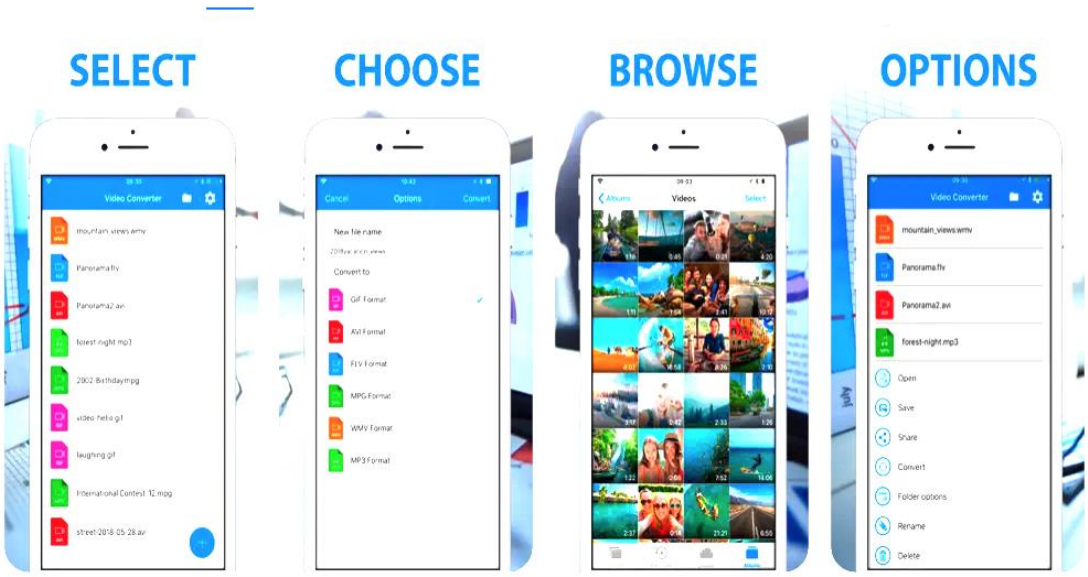

Рисунок 1.4 – Скриншот Video Converter and Media File

1.4 Мета та задачі

Мета:

Створення повного набору інструментів відео для конвертації фільмів, музики і т.д. Можливо зберегти медіафайли в будь-якому форматі для будь-якого пристрою або платформи.

Мій відео конвертер – це потужний відео конвертер, обрізувач відео, конвертер MP4, злиття відео для зменшення розміру тощо.

Задачі:

- конвертація в інші формати;
- зберігання відео з сайтів;
- прибирання аудіодоріжки з відео;

- зменшення або збільшення якості відео;
- відокремлення аудіодоріжки від відео.

1.5 Висновки з розділу

Використання інформаційних технологій в роботі з відео і аудіо спрощує ряд робочих процесів і підвищує їх ефективність.

РОЗДІЛ 2 ОГЛЯД ПРОГРАМНОГО РІШЕННЯ

2.1 Мобільна операційна система iOS та її переваги над Android

iOS є мобільною операційною системою, яку розробляє корпорація Apple для використання на смартфонах, планшетах і носіїв пристроїв. Вона стане головним конкурентом операційної системи з відкритим кодом Android.

Рисунок 2.1 – Логотип корпорації Apple Inc.

Операційна система iOS має кілька основних переваг перед Android, зокрема високий рівень, швидкість роботи, більш інтуїтивний та зручний інтерфейс, що забезпечує високий рівень задоволення користувачів. Крім того, Apple забезпечує оновлення операційної системи для всіх пристроїв, які здатні працювати з новим ПЗ. Це означає, що пристрої з операційною системою iOS залишаються актуальними протягом тривалого часу.

Подробніше про переваги:

– Інтерфейс – завдяки тому, що Android є відкритою операційною системою, багато виробників смартфонів, які вимагають Android, надають свої інтерфейси відповідно до своїх власних проявів, встановлюючи так звані

"оболонки". Це все до єдиного стандарту для вигляду операційної системи. У випадку з iOS дизайн має чіткі рекомендації, тому користувачі не мають проблем із взаємодією з новими додатками за допомогою схожого за логічним інтерфейсом.

– Швидкодія – Тому операційна система iOS розроблена для обмеженої кількості пристроїв, це дозволяє її оптимізувати під конкретне апаратне забезпечення. Тісна інтеграція між апаратною та програмною частинами дозволяє кінцевим користувачам отримувати кращі показники швидкості операційної системи.

– Захист – Хакери рідше пишуть шкідливі програми для iOS– пристроїв, після всіх програм підлягають перевірці корпорації Apple, включаючи перевірку ідентичності видавців програм. Це свідчить про вищий рівень безпеки для iOS– пристроїв порівняно з Android– пристроями. Ціна виявлення вразливості "нульового дня" (тобто невідомої попередньої вразливості) в Android є значно меншою, ніж в iOS, що також спостерігається про меншу безпеку Android.

Після аналізу даних, наведених у таблиці 2.1, можна зробити висновок, що iOS є більшою перевагою в порівнянні з Android завдяки високим стандартам якості програмного забезпечення та використання користувальницького досвіду.

Таблиця 2.1 – Порівняння операційних систем IOS та Android

Критерії	iOS	Android
Джерело	Закрите, з відкритими компонентами	Відкрите
ОС	OS X	Відкритий
Розробники	Apple Inc.	Google
Інтерфейсна частина	Тачскрін. Нормалізований інтерфейс в усій системі	Тачскрін. Через можливості кастомізації інтерфейс не нормалізований, може бути різним
Швидкодія	Висока. Завдяки взаємодії хардверної та софтверної частин	Залежить від девайсу

2.2 Основи програмування

Для створення власних додатків для операційної системи iOS необхідні такі інструменти та навички:

– Xcode: це інтегроване середовище розробки, яке включає в себе редактор коду, організатор проекту, компілятор, відладчик та інші важливі інструменти для створення додатків;

– Swift: потужна мова програмування, яка використовується для розробки додатків для iOS, macOS та tvOS. Вона має багато функцій, які допомагають продуктивно писати код;

– Створення інтерфейсів: кожен додаток має інтерфейс користувача, який складається з кнопок, переглядів, навігації, міток, зображень та інших елементів;

– Кодування логіки: код Swift відповідає тому, що відбувається в додатку та в який час. Це називається логікою, яка є вирішальною для будь-якої програми;

– Архітектура додатків: написання чіткого, розширювального та підтримуваного коду є так само місцем, як написання працюючого коду. Архітектура додатків є складною темою і потребує часу на вивчення різних рішень, які найкраще працюють у різних сценаріях.

Архітектура iOS складається з чотирьох шарів абстракцій, які розробники можуть використовувати для побудови додатків:

1. Cocoa Touch, який забезпечує основну інфраструктуру додатків та рамок, які можуть використовувати функції, такі як push– сповіщення, багатозадачність та сенсорний відвід;
2. Медіа, яке дозволяє програмі працювати з відео, аудіо та графікою;
3. Основні сервіси, які надають системні послуги, такі як Core Foundation і Foundation Frameworks;

4. Core OS, який забезпечує різні послуги, такі як безпека та локальна автентифікація, інші фреймворки.

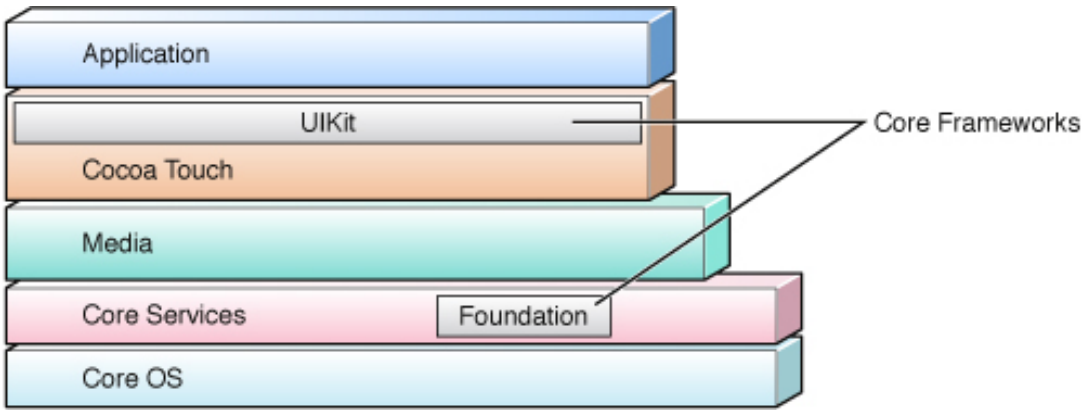

Рисунок 2.3 – Cocoa в архітектурі iOS

За рекомендаціями Apple Inc., розробники повинні писати код на вищому рівні застосування та використовувати фреймворки глибокого рівня на вищому рівні лише у випадку, коли це необхідно.

Для створення успішного додатка розробники повинні надавати оптимальний дизайн інтерфейсу (User Interface) та використовуваний досвід (User Experience) на кожному етапі. Користувачі повинні мати можливість легко переглядати, розуміти та взаємодіяти з вмістом без потреби постійного прокручування чи масштабування – це означає, що текст має бути чітко читабельним. Крім того, користувачам повинні бути доступні функції сенсорного екрану та завдання повинні виконуватися з найменшою кількістю кроків.

2.3 Мова програмування Objective– C і Swift

Objective-C, також відомий як Objective C, ObjC або Obj-C, – об’єктно-орієнтована мова програмування, розроблена Apple на основі мови C і парадигми Smalltalk. На відміну від C++, Objective– C повністю поєднується з C, а код C компілюється. Об’єктна модель побудована в стилі Smalltalk, тобто

об’єкти отримують повідомлення. Компілятор Objective– C є частиною GCC і доступний на більшості основних платформ. Мова в основному використовується для MacOS X (Cocoa) і GNUstep – двох реалізацій об’єктно–орієнтованого інтерфейсу OpenStep. Objective-C створив Бред Кокс на початку 1980-х років у своїй компанії StepStone. Однією з особливостей мови є те, що вона орієнтована на повідомлення, тоді як C++ орієнтована на функції. Це означає, що виклики методів інтерпретуються як надсилання повідомлень (з іменами та аргументами) до об’єкта, подібно до того, як це працює в Smalltalk. Цей підхід має низку переваг – будь-якому об’єкту можна надіслати будь-яке повідомлення, а об’єкт може просто переслати повідомлення іншому об’єкту для обробки (відоме як делегування), що полегшує реалізацію розподілених об’єктів (об’єктів, розташованих за різними адресами). місцях і навіть на різних комп’ютерах). Прив’язка повідомлення до відповідної функції відбувається безпосередньо під час виконання. Мова забезпечує звичайну підтримку протоколу (тобто поняття інтерфейсу об’єкта та протоколу чітко розділені). Успадкування підтримується для об’єктів (не множинних), а множинне успадкування підтримується для протоколів. Об’єкт може успадковувати інший об’єкт і підтримувати кілька протоколів одночасно. Структура імен файлів: файли з розширенням – h є заголовками з описом класів і функцій, як у C і C++, а файли з розширенням – m містять реалізації класів і методів.

Swift (наступник Objective-C) – це відкрита скомпільована мова програмування, розроблена для розробки програмного забезпечення та програм для iOS/MacOS та інших платформ. Його можна використовувати для створення програмного забезпечення для різних типів пристроїв і обладнання, включаючи смартфони та планшети, настільні ПК тощо.

Swift є основною мовою розробки в екосистемі корпорації Apple, він став заміною мови Objective-C, яка використовувалася нею раніше (та й продовжує використовуватися до цього дня). Процес створення нової мови

розпочався у 2010 році, а у червні 2014 року її представили громадськості на конференції WWDC, разом із 500-сторінковим гайдом.

У червні 2015 року мова Swift оновлена до версії 2.0, що дозволило підвищити її продуктивність, покращити синтаксис, додати нове API для виправлення помилок та можливість перевіряти сумісність функцій мови з цільовими платформами. У грудні того ж року Apple випустила бета– версію Swift 3.0, яка підтримувала операційні системи iOS, OS X і Linux, а також ліцензувала опенсорс– ліцензію Apache. Крім того, вона несумісна з ранніми версіями мови.

У 2017 році компанія представила версію Swift 4.0, а в 2018 році – версію 4.2 та бета-версію 5.0, які були випущені одночасно з iOS 12. У п’ятій версії платформи з’явився ряд нововведень, у тому числі підтримка регулярних виразів робота двійкового інтерфейсу (додатків ABI зі стандартними бібліотеками) і т.д.

Остання на момент написання статті версія Swift 5.5 запущена у вересні 2021 року, в ній було значно розширено підтримку паралельної та асинхронної обробки даних.

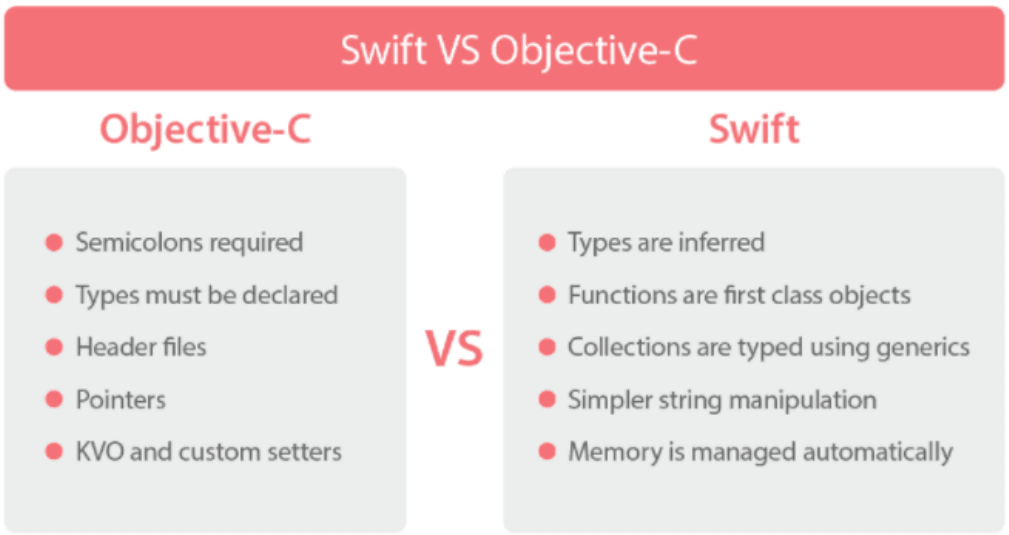

Рисунок 2.4 – Порівняння Swift та Objective-C

2.4 Візуальне середовище розробки

Xcode є інтегрованим середовищем розробки (IDE), яке міцно пов’язане з фреймворками Cocoa та Cocoa Touch. Це дозволяє створювати додатки для пристроїв на базі операційних систем iOS та OS X для сторонніх розробників. Ідея IDE виникає в тому, щоб поєднати різні інструменти, які використовуються під час розробки програмного забезпечення. У Xcode, зокрема, є редактор коду, компілятор, відладчик, конструктор інтерфейсів та симулятор (будь-який технічно є окремою програмою, запущеною з Xcode).

Основні вікна робочої області в Xcode (рисунок 2.5-2.6) розташовані зліва в напрямку годинникової стрілки:

- область навігації;
- область редактору;
- область інструментів;
- область (debug);

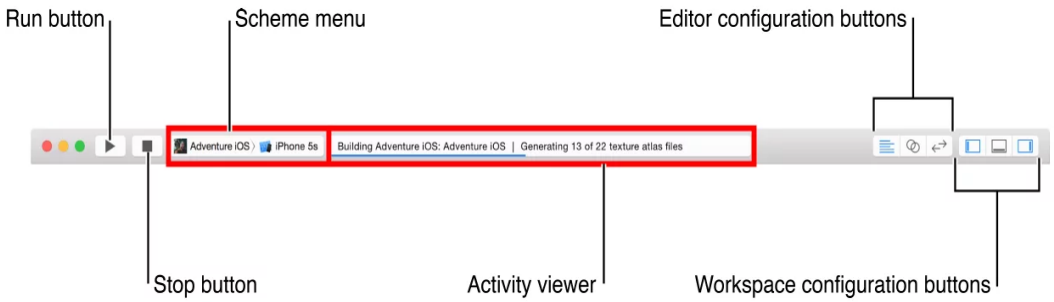

Рисунок 2.5 – Вікно робочої області Xcode

2.4.1 CocoaPods

CocoaPods є потужним інструментом, необхідним кожному розробнику iOS для керування залежностями у створених програмах. Він дозволяє легко, швидко та просто підключати різні бібліотеки та інструменти, які значно покращують процес розробки програм.

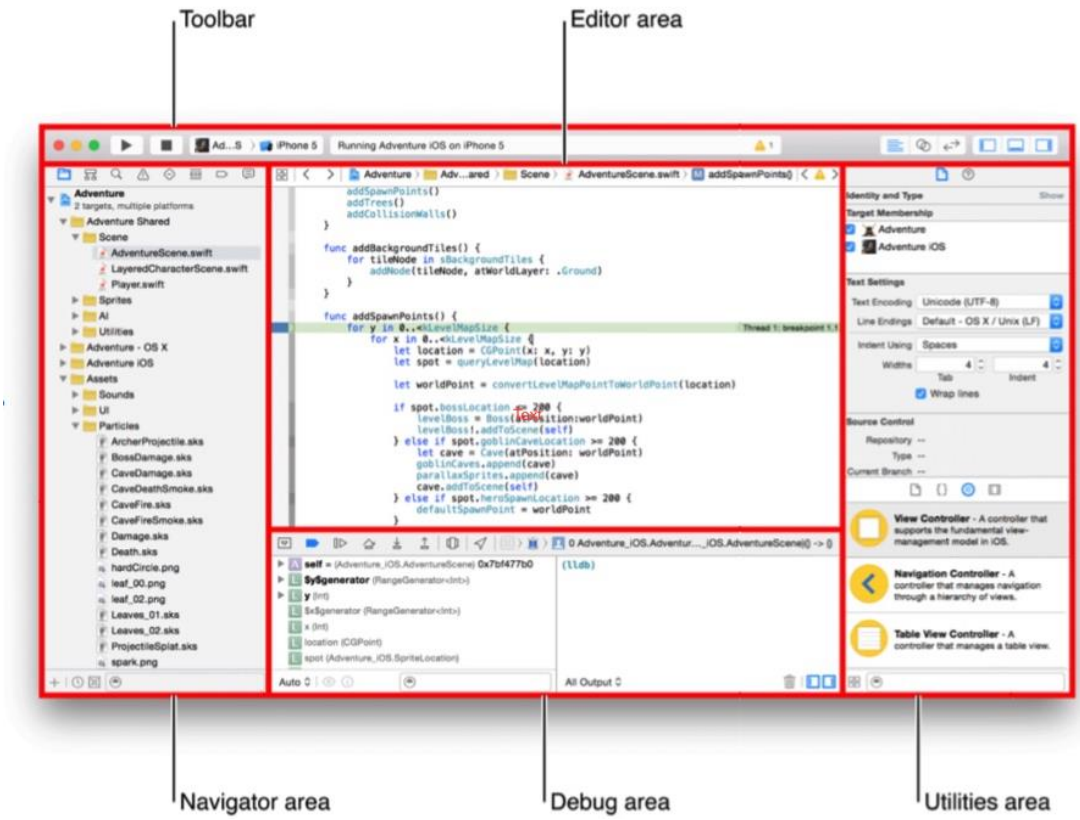

Рисунок 2.6 – Вікно робочої області детально у Xcode

При створенні мобільного додатка не доцільно писати кожну частину коду самостійно. Виходячи з цього, попередній вихідний код інших розробників надає в корінь проекту. Однак такий підхід має кілька недоліків: складно слідкувати за версіями бібліотеки та їх взаємозв'язками, немає загального місця, можна переглянути список усіх доступних бібліотек, потрібно завжди пам'ятати про оновлення вихідного коду бібліотеки, іноді проект потрібно зберегти в репозиторії без зайвих файлів. бібліотек, і той, хто завантажує проект для роботи з ним, повинен самостійно додати ці файли до проекту.

Управління залежностями за допомогою CocoaPods є корисним інструментом для розробників iOS, після чого він допоможе вирішити проблеми, пов'язані з управлінням залежностями. CocoaPods дозволяє контролювати взаємодію між іншими бібліотеками, які використовують у проекті, забезпечує структурованість проекту та забезпечує підтримку проекту в належному вигляді.

Щоб створити файл зі списком залежностей, необхідно відкрити термінал, перейти до директорії проекту та виконати команду «pod init». Ця команда створює спеціальний файл під назвою «Podfile», в якому можна описати список найважливіших залежностей для проекту.

Для встановлення нових залежностей необхідно внести опис у файл зі списком залежностей і запустити команду «pod install». Ресурс cocoapods.org є корисним джерелом для пошуку потрібних бібліотек, де можна знайти всі фреймворки та документацію до кожного з них.

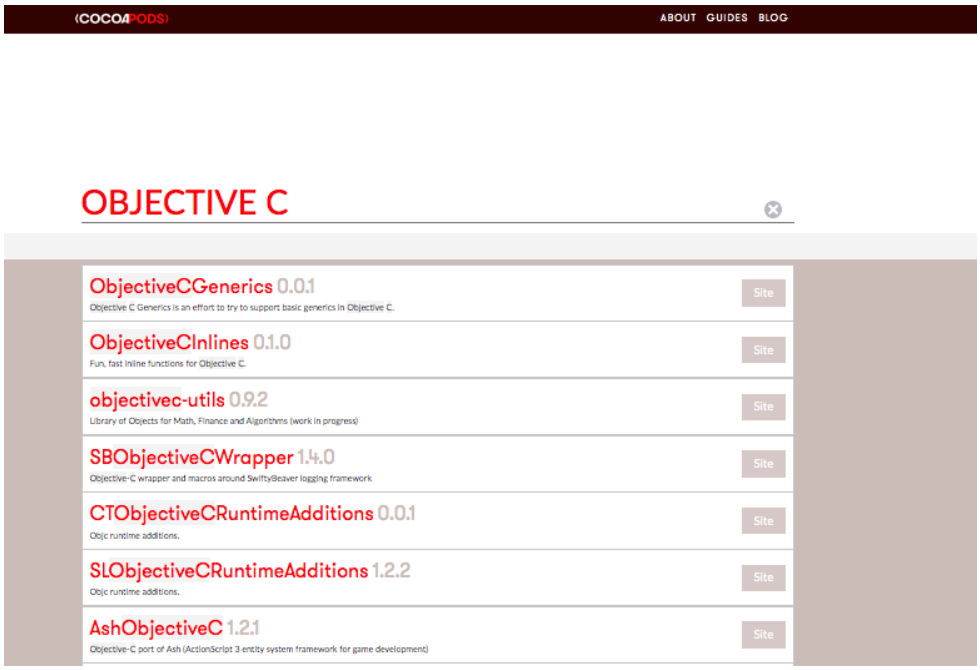

Рисунок 2.7 – Бібліотеки на сайті cocoapods.org

Після налаштування залежностей буде створено нову проектну заготовку. З цього моменту проект потрібно відкрити через згенерований файл із розширенням <workspace>. Цей файл містить у собі задоволення, або "pods", тому він є єдиним способом відкриття проекту.

2.4.2 Storyboard

Storyboard було представлено в iOS 5 як корисну функцію, яка економить час на створенні інтерфейсу користувача для ваших програм. Це дозволяє створювати кілька контролерів перегляду в одному файлі. До Storyboard вам потрібно було використовувати файли XIB, і ви могли використовувати лише один файл XIB на контролер (UITableViewCell, UITableView або інші підтримувані типи UIView).

За допомогою Storyboard ви можете створити кілька контролерів перегляду та налаштувати з'єднання та переходи (segues) між ними (рисунок 2.8). Такий підхід дуже зручний і візуально зрозумілий.

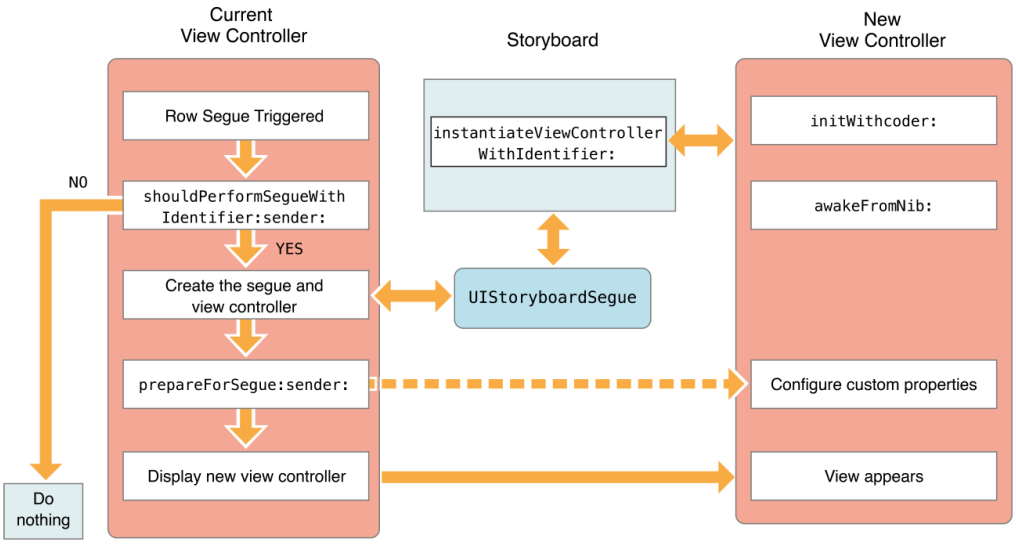

Рисунок 2.8 – Відображення контролера перегляду за допомогою переходу

Storyboard має ряд переваг:

- можливість візуально викласти всі потрібні контролери у сцени (scenes) та можливість описати зв'язки між ними;
- механізм переходу між сценами. Ця функція дозволяє зменшити кількість кодів, необхідних для роботи з інтерфейсом користувача, за допомогою зручного способу налаштування переходів між сценами.

- AutoLayout – це функція, яка дозволяє встановлювати математичні зв’язки між елементами і визначати їх положення та розміри. Це значно полегшує розробку інтерфейсу для різних розмірів екрану, що робить її більш простою і зрозумілою.

Існує безліч різних типів переходів, відомих як segues. Однак кожен тип переходу може підходити для певних ситуацій та не підходити для інших. Вибір переходу між двома контролерами залежить від структури інтерфейсу та бажаного ефекту. Деякі переходи можуть бути лише ефективними в деяких сценаріях. Ось деякі типи переходів, які доступні:

- перехід push segue дозволяє переміщатися в обидві сторони стеку контролерів. Використовуючи цей перехід, ви можете «вставити» новий контролер на вершину стеки поверх попереднього контролера або «викинути» контролер зі стеки, щоб відкрити нижній контролер у стеці.
- present modally – це один із найбільш поширених типів переходів в iOS, що дозволяє показати новий контролер перегляду поверхневого, проте цей контролер не повинен бути включений до будь-якої ієрархії або стека екранів, він просто розміщується на верхньому рівні попереднього контролера.
- Перехід типу popup показує новий контролер перегляду у спливаючому вікні, що обмежує його розмір. Цей вид переходів має сенс використовувати лише на iPad, тому що на пристроях з меншими екранами, наприклад на iPhone, він поводиться як звичайний модальний перехід.
- Show detail – це спеціальний тип переходу, який добре підходить для використання з контролерами розділення перегляду.

2.4.3 Simulator

Симулятор, що є частиною інструментів Xcode, дозволяє швидко створювати прототипи та тестувати збірки вашої додатки під час розробки. Він

емулює мобільний пристрій, на якому ведеться розробка і працює як стандартне застосування.

Симулятор є корисним інструментом для попереднього тестування, який слід використовувати до того, як тестувати програму на власному пристрої.

Рисунок 2.9 – Simulator у дії

Симулятор дозволяє відтворювати пристрої з великими розмірами екрану та версіями ОС (рисунок 2.9). Кожне модельоване середовище, що складається з компонування пристрою та версії програмного забезпечення, є самостійним та незалежним від інших, має власні налаштування та файли. Ці параметри та файли доступні на кожному пристрої, який може бути відтворений у середовищі моделювання.

Використання Simulator дозволяє знайти основні проблеми вашого застосування на етапі розробки та раннього тестування, а також перевірити його за допомогою інструментів розробника, які доступні лише в середовищі моделювання.

Хоча симулятор є корисним інструментом, він не є повністю достовірним тестом програми і не повинен бути єдиним засобом для її перевірки. Симулятор працює на комп’ютері і має доступ до його ресурсів,

таких як процесор, пам’ять і мережеве з’єднання, які можна привести до істотної різниці в продуктивності через мобільний пристрій.

Однак завжди потрібно перевіряти продуктивність та користувальницький інтерфейс програми на мобільному пристрої, так як вони можуть працювати вільніше та менш плавно, ніж у симуляторі. Для тестування програм на мобільному пристрої можна записати так само як «Sideloading», підключивши пристрій до комп’ютера та вибравши його в якості цільового пристрою для тестування у вікні вибору пристрою (рисунок 2.10).

Рисунок 2.10 – Sideloading

2.4.4 Interface Builder

Interface Builder є компонентом Xcode – системи розробки програмного забезпечення для розробників Apple Developer Connection, яка працює на операційній системі MacOS.

Він надає колекції об’єктів користувацького інтерфейсу для розробників, які задають Objective– C або Swift. Ці об’єкти користувальницького інтерфейсу містять такі елементи як:

- текстові поля (textField);
- таблиці даних (tableView);
- слайдери (Slider);
- спливаючі меню (popupMenu).

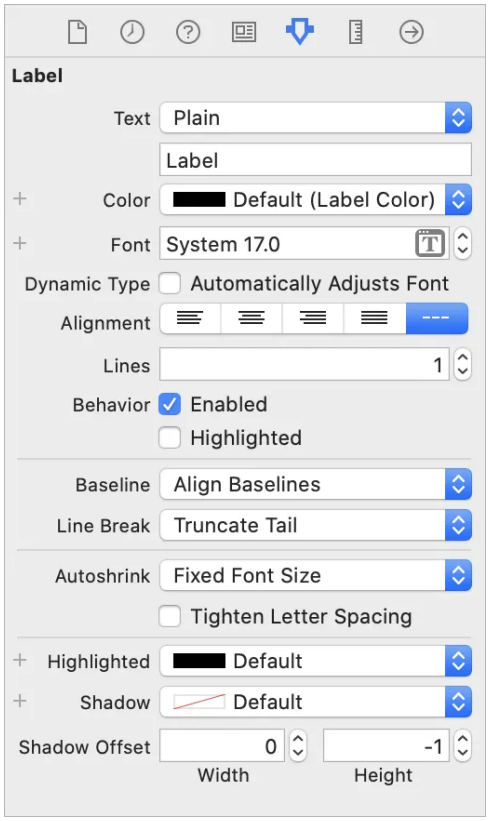

Рисунок 2.11 – Interface Builder in Xcode

Можна переформулювати наступним чином: Розробники можуть розширювати палітри в Interface Builder, додавати до них нові елементи інтерфейсу та використовувати їх для створення інтерфейсу своєї програми.

Цей процес є простим – розробник просто перетягує елементи з палітри у вікно або меню. У коді програми вказуються конкретні об’єкти, які підтримують повідомлення від цих елементів інтерфейсу.

Усі необхідні ініціалізації відбуваються до виконання, що сприяє підвищенню продуктивності та робить процес розробки більш структурованим.

2.4.5 Sourcetree

Sourcetree – це потужний і, що не менш важливо, абсолютно безкоштовний менеджер проектів, розміщених у сховищі GitHub. Він буде корисним тим, хто одночасно веде кілька розробок і хоче максимально швидко

завантажувати зміни до Git. Програма дозволяє забути про роботу з командним рядком та пропонує дуже простий інтерфейс з підтримкою незалежних вкладок для кожного проекту. Вона стане незамінним помічником “тимбілдам” та рядовим розробникам, які спільно працюють над одним проектом.

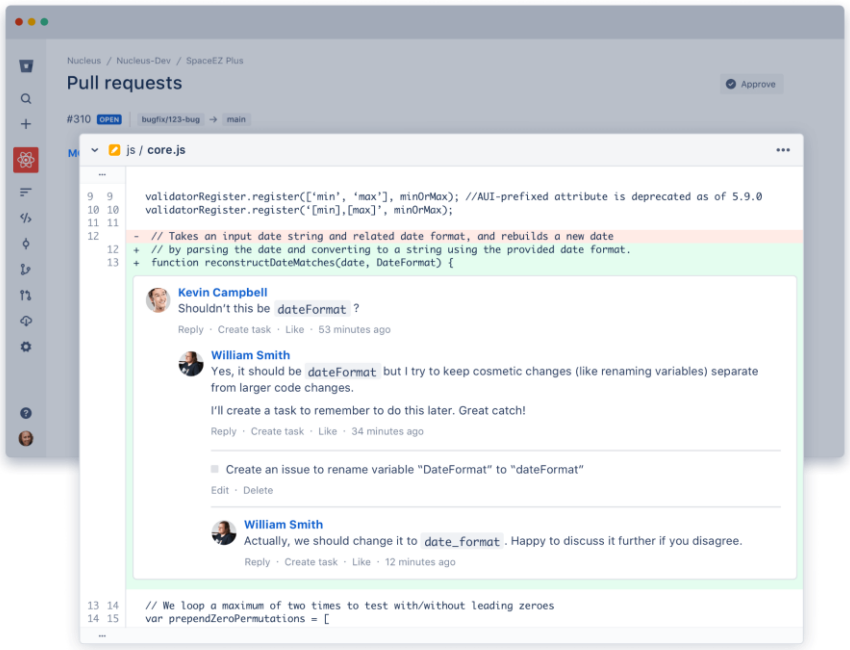

Рисунок 2.12 – програма Sourcetree

Завдяки Sourcetree можна створювати, переглядати та змінювати проекти GitHub, а також швидко публікувати нові розробки у своєму профілі.

Більшість операцій у програмі здійснюється з єдиної панелі інструментів. Там можливо знайти клавіші для відкриття репозиторіїв, впровадження патчів, переміщення та клонування окремих елементів проекту, відображення останніх змін, виконання коду, додавання тегів, запуску «ребейза» тощо. Менеджер інтуїтивно візуалізує локальні та глобальні репозиторії, а також допомагає швидко поєднувати гілки коду. Просунуті користувачі можуть взагалі будь-якої миті скористатися терміналом, зробивши буквально один клік.

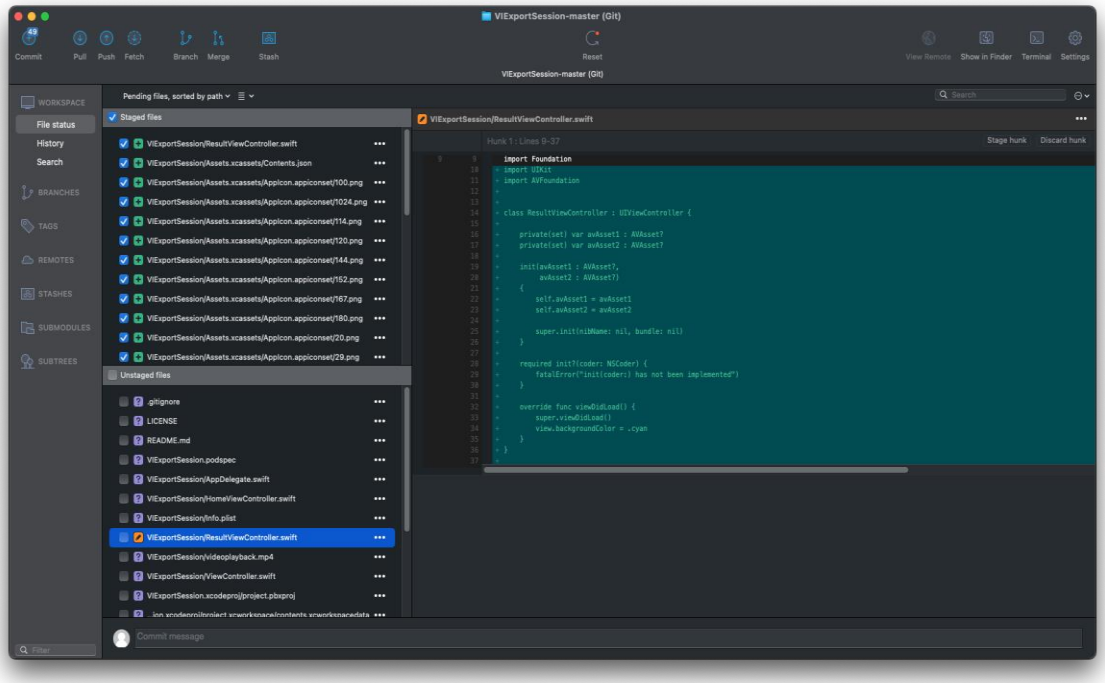

Рисунок 2.13 – проект конвертера

Крім роботи з GitHub, менеджер також можна підключати до Stach та Bitbucket. Не забули розробники Sourcetree і про можливість роботи з Mercurial. Надалі планується підтримка Git-Flow та інтеграція з JIRA. Загалом, програма буде корисна як розробникам-початківцям, так і «ветеранам», які прагнуть оптимізувати свою взаємодію з GitHub і суміжними сервісами.

2.5 Frameworks

В iOS є два типи фреймворків: статичний і динамічний.

Статичні фреймворки – це попередньо скомпільовані бібліотеки, які підключаються до вашого додатку в процесі збірки. Це означає, що код фреймворка включено в додатки кінцевого подвійного файлу та не може бути оновленим або зміненим під час виконання.

Одним з основних переваг статичних фреймворків є те, що компоновщик часто може підтримувати повну зачистку файлів, що виконується. Ви можете ввімкнути цю функцію в Xcode та дозволити

компоненту аналізувати використані символи, що призведе до потенційного економічного розміру.

Динамічні фреймворки, з іншого боку, представляють собою скомпільовані бібліотеки, які завантажуються під час виконання. Це означає, що код платформи не включено в подвійний файл додатків і може бути оновлений або змінений під час виконання.

Динамічні фреймворки вбудовані десь всередині кінцевого додатка бандла і збільшують загальний розмір додатка бандла.

Використання динамічних фреймворків має переваг у розмірі. У залежності від конфігурації вашого додатка кілька подвійних файлів всередині бандла вашого додатка можуть використовувати один і той же динамічний фреймворк. Зазвичай, це відбувається, коли розробник хоче розділити код між основними подвійним файлом додатків і розширенням додатків.

Таким чином, статичні та динамічні фреймворки – це два типу попередньо скомпільованих бібліотек, які розробники iOS можуть використовувати для створення своїх програм.

Основні відмінності між ними заключаються в їх впливі на розмір і продуктивність додатка. Статичні фреймворки збільшують розмір додатків, але забезпечують більш швидкий час запуску. У кінцевому підсумку рішення про використання статичного або динамічного фреймворка буде залежати від конкретних потреб додатка і цілей розробки.

2.5.1 CoreVideo

Core Video – модель обробки відео, яка використовується в MacOS. Core Video зв’язує процес декомпресії кадрів з джерела відео з іншими технологіями Quartz для рендерингу та композиції зображень.

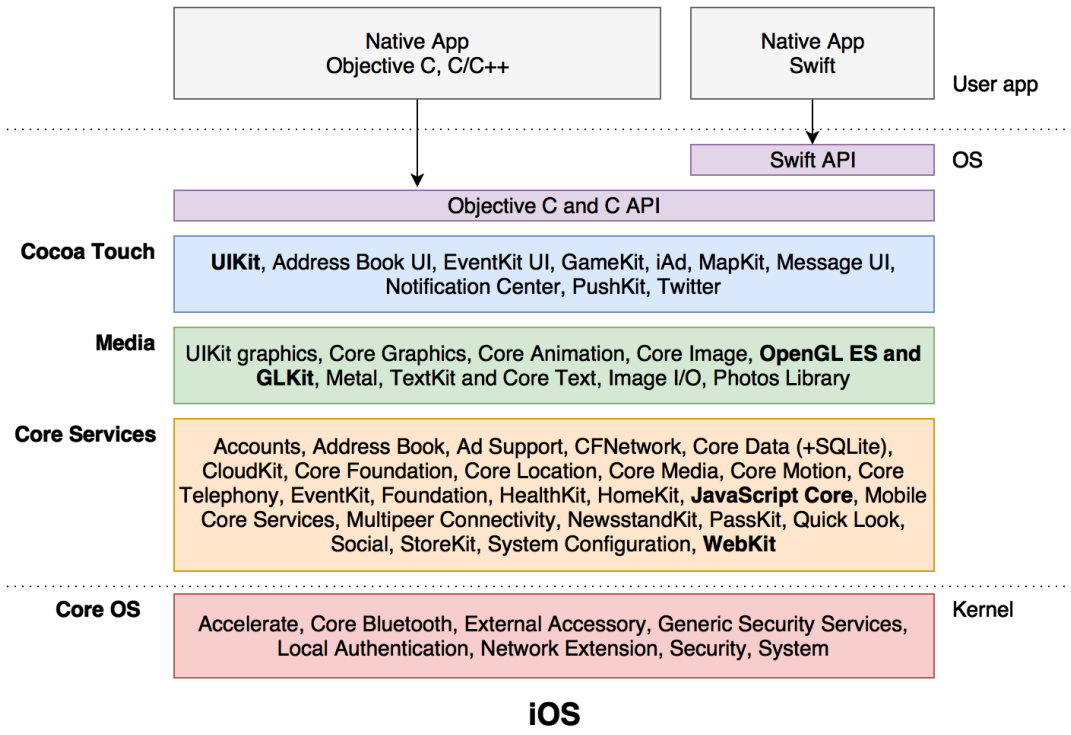

Рисунок 2.14 – Загальна архітектура iOS

Він забезпечує як модель буферизації, так і рішення для синхронізації відтворення в межах свого конвеєра обробки. Джерело відео надає розпакований потік даних для візуалізації як зображення у візуальному контексті в Quartz 2D. Основне відео можна розглядати як зв'язок між цим джерелом відео та контекстом його відображення. Візуалізовані зображення можуть бути додатково оброблені за допомогою Core Image перед створенням остаточної сцени за допомогою Quartz Compositor. Як частина процесу композиції, Core Video підтримує буфер кадрів,зберігаючи, пул відтворених, складених кадрів, готових до відтворення. Щоб забезпечити плавне відтворення, Core Video використовує високо пріоритетний потік для підтримки кадрового буфера. Це посилання на дисплей працює незалежно від програми, яка викликає відтворення відео, і воно компенсує різні частоти оновлення дисплея та затримку.

Core Video отримує переваги від рендерингу графічного процесора (GPU) і композиції, які забезпечуються Quartz GL, Core Image і Quartz Extreme, оскільки остаточне відтворення компонується на поверхні Opengl.

Core Video забезпечує конвеєрну модель для цифрового відео. Він спрощує роботу з відео, розділяючи, процес на окремі етапи. Це полегшує розробникам доступ до окремих кадрів і керування ними, не турбуючись, про переклад між типами даних або проблеми синхронізації відображення. Програмам, яким не потрібно маніпулювати окремими відеокадрами, не потрібно безпосередньо використовувати Core Video.

2.5.2 AVFoundation

AVFoundation – це фреймворк для роботи з аудіовізуальними медіафайлами на iOS, macOS, watchOS, tvOS. Завдяки AVFoundation можна відтворювати, створювати та редагувати файли QuickTime та файли MPEG– 4, відтворювати потоки HLS та вбудовувати потужну мультимедійну функціональність у свої програмні продукти.

Як стало зрозуміло з визначення, AVFoundation – це широкополосний фреймворк з великою кількістю можливостей, в які входять і створюються фото і відео, на яких сьогодні і можливо сконцентруватися. Фото та відеокамери – найчастіший ключ використання AVFoundation.

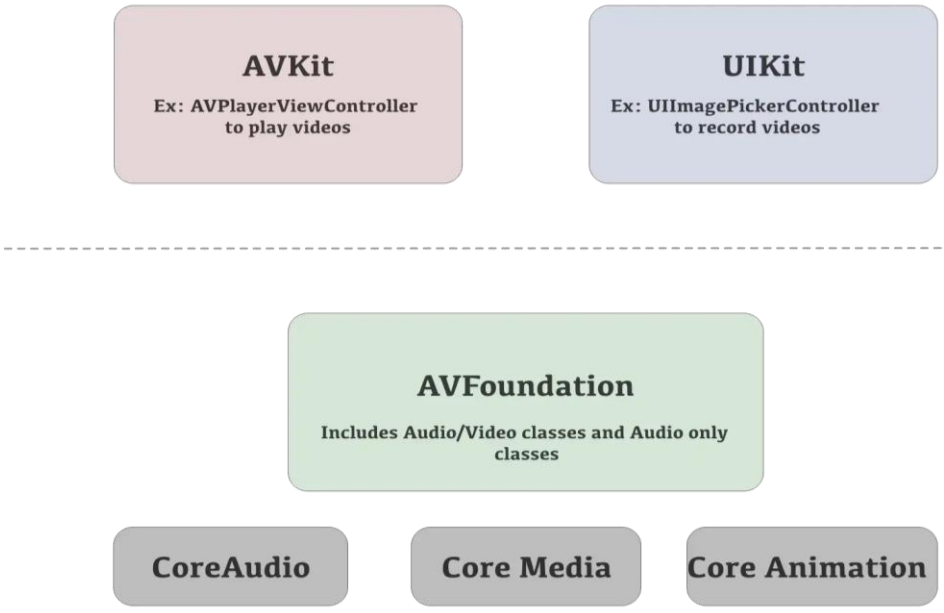

Рисунок 2.15 – AVFoundation framewotk

Під фреймворком AVFoundation лежать CoreVideo, CoreMedia – ці назви говорять самі за себе. А також CoreAnimation для представлення ієрархії відео та презентаційного шару.

Якщо заглибитися в тему фото. Основні класи AVFoundation, що використовуються для написання камери:

– AVCaptureDevice – клас, що є прямим API для пристроїв камери;

```
let session = AVCaptureSession()
session.beginConfiguration() session.sessionPreset = AVCaptureSession.Preset.photo
```

– AVCaptureDeviceInput – проводити дані від камери;

```
captureSession.beginConfiguration()
let videoDevice = AVCaptureDevice.default(.builtInWideAngleCamera,
for: .video, position: .unspecified)
guard
let videoDeviceInput = try? AVCaptureDeviceInput(device: videoDevice!),
captureSession.canAddInput(videoDeviceInput)
else { return }
captureSession.addInput(videoDeviceInput)
```

– AVCaptureOutput – абстрактний клас, що відповідає для виведення картинок на екран. Ми будемо використовувати його сабкласс – AVCapturePhotoOutput.

```
let photoOutput = AVCapturePhotoOutput()
guard captureSession.canAddOutput(photoOutput) else { return }
captureSession.sessionPreset = .photo
captureSession.addOutput(photoOutput)
captureSession.commitConfiguration()
```

– AVCaptureSession – забезпечує зв’язок між входом і аутпутом камери та відповідає за роботу камери в цілому.

```
var captureSession = AVCaptureSession() lazy var frontCameraDevice: AVCaptureDevice? = { let devices = AVCaptureDevice.devicesWithMediaType(AVMediaTypeVideo) as! [AVCaptureDevice] return devices.filter{$0.position == .Front}.first }()
```

– AVCaptureVideoPreviewLayer – сабклас CALayer, що виводить на екран зображення відео з нашого девайса.

```
class PreviewView: UIView { // Use AVCaptureVideoPreviewLayer as the view’s backing layer.
override class var layerClass: AnyClass {
AVCaptureVideoPreviewLayer.self }
var previewLayer: AVCaptureVideoPreviewLayer {
layer as! AVCaptureVideoPreviewLayer
}
```

Також сьогодні ми будемо використовувати такі допоміжні класи:

– AVCaptureDevice.DiscoverySession – дозволяє підключити специфічний CaptureDevice для даних пристроїв – двох і трьох камер, або тільки одну з них.

```
func cameraWithPosition(_ position: AVCaptureDevicePosition) - > AVCaptureDevice? {
 if let deviceDescoverySession = AVCaptureDeviceDiscoverySession.init(deviceTypes: [AVCaptureDeviceType.builtInWideAngleCamera], mediaType: AVMediaTypeVideo, position: AVCaptureDevicePosition.unspecified)
{ for device in deviceDescoverySession.devices
{ if device.position == position
{ return device }
 }
 }
return nil
}
```

– AVCapturePhotoSettings – налаштування камери для конкретного зображення.

```
func capturePhoto()
{
let settings = AVCapturePhotoSettings() let previewPixelType = settings.availablePreviewPhotoPixelFormatTypes.first!
let previewFormat = [kCVPixelBufferPixelFormatTypeKey as String: previewPixelType, kCVPixelBufferWidthKey as String: 160, kCVPixelBufferHeightKey as String: 160]
settings.previewPhotoFormat = previewFormat self.cameraOutput.capturePhoto(with: settings, delegate: self)
}
```

Більшість користувачів, які намагалися скористатися конвертерами, розуміють що недостатньо мати тільки встановлений програмний програвач, орієнтований на відтворення файлів того чи іншого формату, потрібно встановити ще й додаткові інструменти, звані кодеками. Кодеки – це спеціальні програми або набори програми, що дозволяють кодувати і декодувати файли мультимедіа (в основному це стосується аудіо і відео).

В програмному продукті у дипломному проекті використовується кодек H.264.

Кодек H.264 (Advanced Video Coding) – ліцензований стандарт стиснення відео, призначений для досягнення високого ступеня відеопотоку при збереженні високої якості.

Стандарт стиснення відео H.264 (повна назва MPEG– 4 Part 10 або AVC). Сам стандарт був прийнятий ще в середині 2003 року, але по–справжньому ефективні кодеки цього стандарту почали з’являтися зовсім нещодавно.

Для користувачів перехід до нового стандарту означає покращення ефективності кодування їх послідовностей. Тобто, при однаковій якості стиснутої послідовності фільм нового стандарту займатиме менше місця на

диску або меншу ширину каналу (розробники стандарту ставлять собі завдання зменшити розмір на 50 %).

Кодек AAC – пропрієтарний (патентований) формат аудіо файлу файлу із втратами. Також AAC – це багатоканальний алгоритм кодування аудіо, який підтримує потокову передачу.

AAC (Advanced Audio Coding) спочатку створювався як наступник .MP3 з покращеною якістю кодування. Формат AAC, офіційно відомий як ISO/IEC 13818-7, побачив світ у 1997 році як нова, сьома частина сімейства MPEG-2. Існує також формат AAC відомий як MPEG-4 частина 3.

У дипломній роботі саме ці кодеки будуть відігравати ключову роль у конвертації відео та аудіо.

2.5.3 UIKit

UIKit – надає різноманітні функції для створення програм, включаючи, компоненти, які можна використовувати для створення основної інфраструктури програм для iOS, iPadOS або tvOS.

Фреймворк забезпечує архітектуру вікон і переглядів для реалізації вашого інтерфейсу користувача, інфраструктуру обробки подій для поставки Multi-Touch та інших типів введення у вашу програму, а також основний цикл виконання для керування взаємодією між користувачем, системою та вашою програмою.

UIKit також включає підтримку анімації, документів, малювання та друку, керування текстом і відображення, пошук, розширення, програм, керування ресурсами та отримання інформації про поточний пристрій. Також можливо налаштувати підтримку спеціальних можливостей і локалізувати інтерфейс програми для різних мов, країн або культурних регіонів.

UIKit бездоганно працює з інтерфейсом SwiftUI, тож можно реалізувати частини свого додатка UIKit у SwiftUI або змішувати елементи інтерфейсу між

двома фреймворками. Наприклад, ви можете розміщувати представлення UIKit і контролери перегляду всередині представлень SwiftUI і навпаки.

Щоб створити програму для macOS, ви можете використовувати SwiftUI, щоб створити програму лише для Mac. Крім того, ви можете перенести свою програму, яка працюватиме на всіх платформах Apple або використовувати AppKit, щоб створити програму лише для Mac. Крім того, ви можете перенести свою програму UIKit для iPad на Mac за допомогою Mac Catalyst.

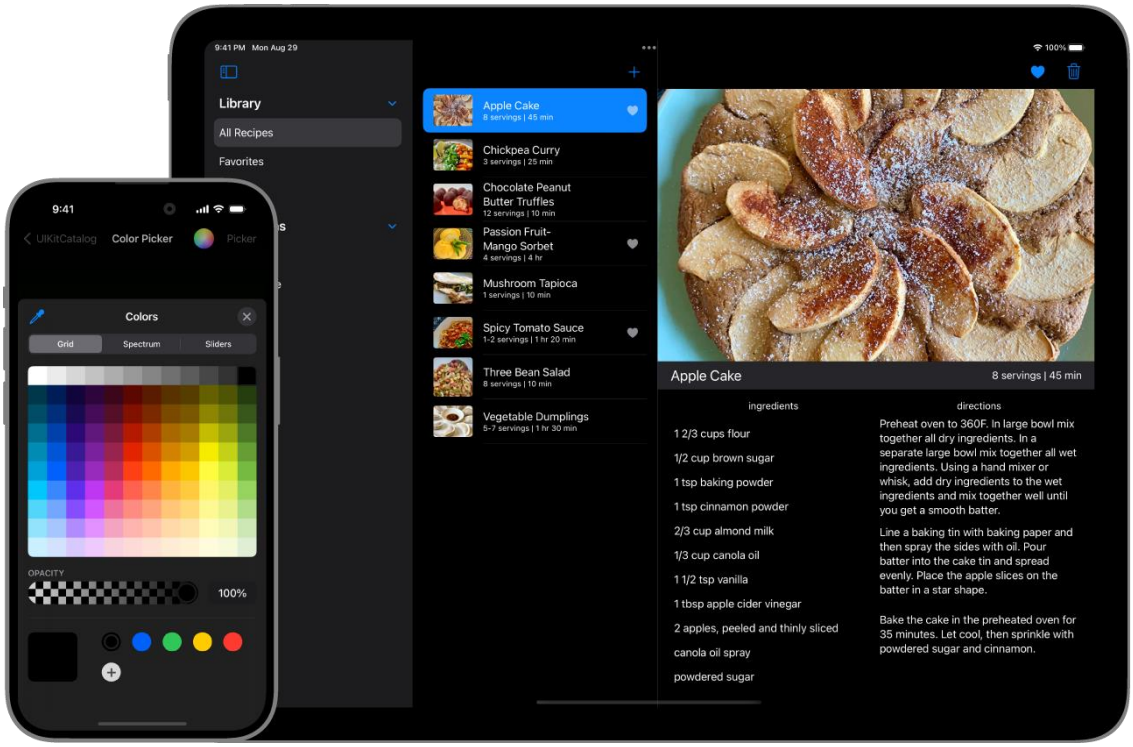

Рисунок 2.16 – Framework UIKit

Класи UIKit потрібно лише з головного потоку програми або головної черги відправлення, якщо інше не вказано в документації для цих класів. Це обмеження особливо стосується класів, які є похідними від UIResponder або які передбачають будь-яке маніпулювання інтерфейсом користувача вашої програми.

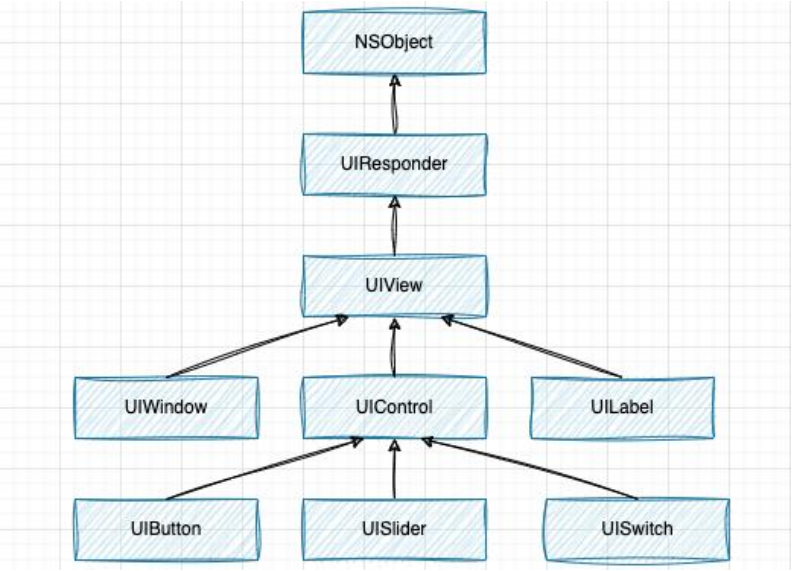

Рисунок 2.17 – UIKit Inheritance

2.5.4 Core Media

Фреймворк Core Media визначає меді-конвеєр, який використовується AVFoundation та іншими медіа-фреймворками високого рівня на платформах Apple. Потрібно використовувати низькорівневі типи даних та інтерфейси Core Media для ефективної обробки медіа-семплів і керування чергами медіа-даних.

2.5.5 Core Audio

Core Audio – це цифрова аудіо інфраструктура iOS і OS X. Вона включає в себе набір програмних інфраструктур, розроблених для задоволення потреб у аудіо у програмах, оптимізованих для обчислювальних ресурсів, доступних у мобільній платформі з живленням від акумулятора.

Core Audio використовує поняття проксі-об’єктів для представлення таких речей, як файли, потоки, аудіоплеєри тощо. Наприклад, якщо потрібно, щоб програмний продукт працював з аудіо файлами на диску, першим кроком є створення об’єкта аудіофайлу типу AudioFileID, який оголошено як непрозорі дані.

РОЗДІЛ 3 ПРОГРАМНИЙ МОДУЛЬ

3.1 Архітектура застосування

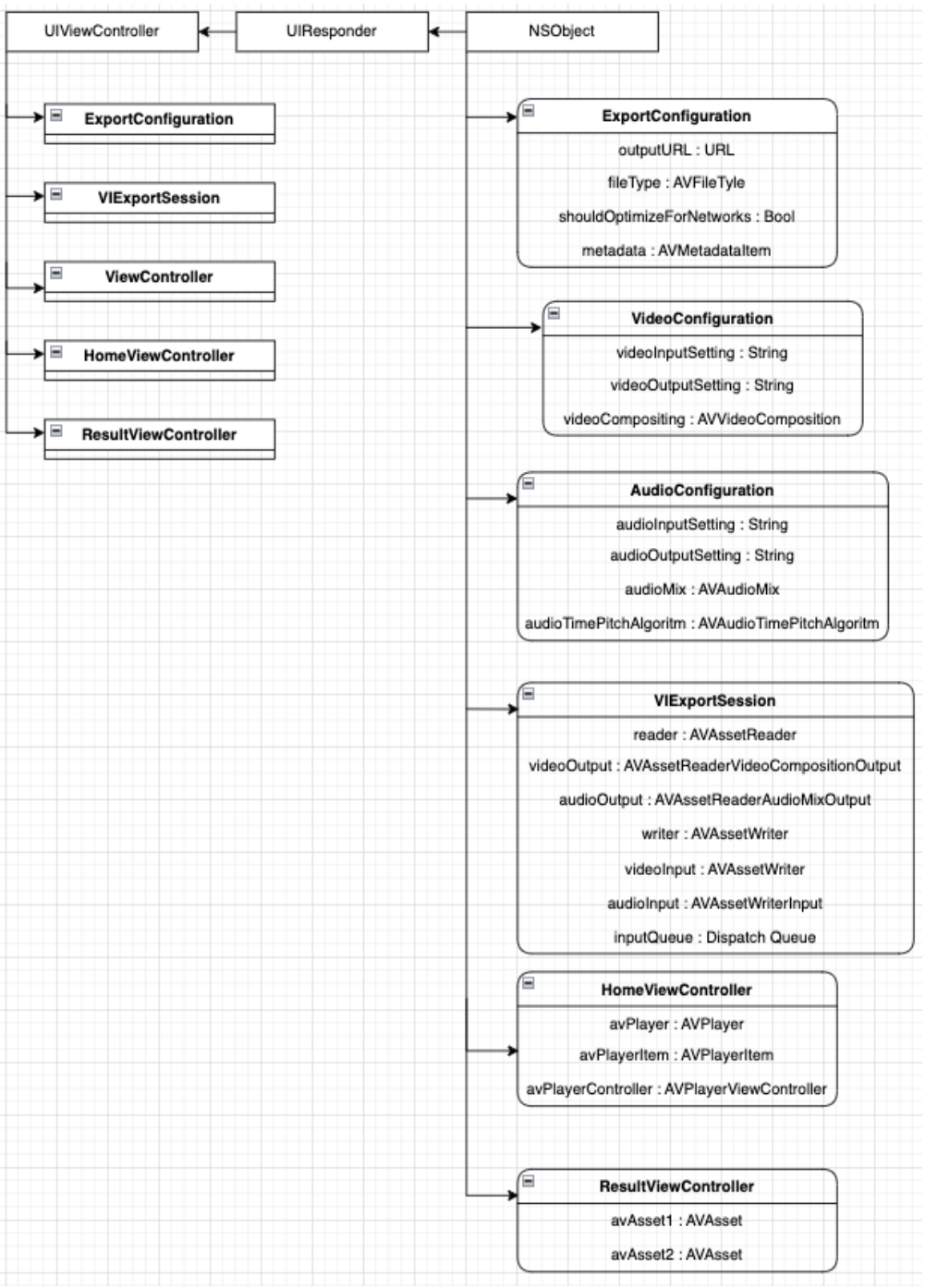

Рисунок 3.1 – Архітектура додатка

MVC є класичним патерном програмування, яке розділяє застосування на 3 шари абстракції. Це дозволяє полегшити внесення якихось змін до

застосування в майбутньому після чіткого розподілу відповідальності між його компонентами. Основна мета MVC досягає чіткого розподілу відповідальності між моделлю, представленням та контролером цього (рис. 3.2), щоб вирішити проблему.

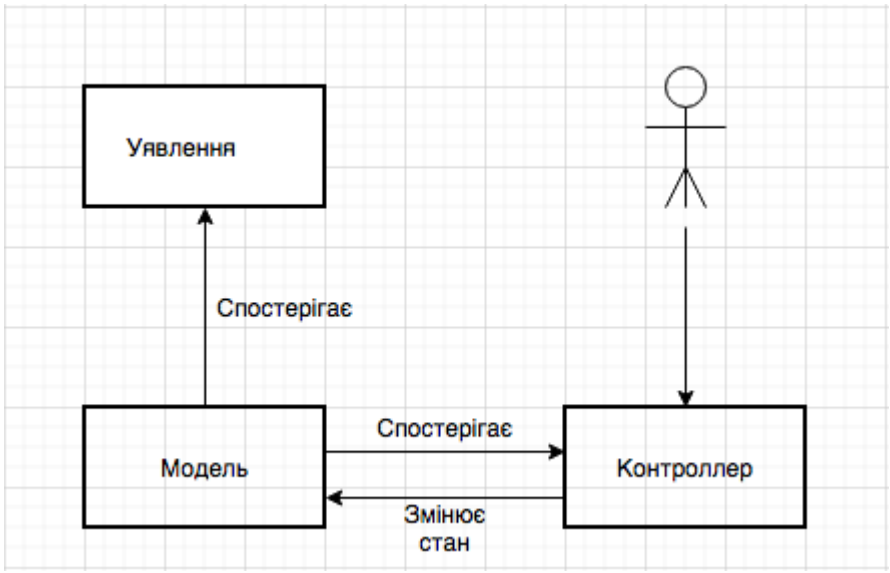

Рисунок 3.2 – класичне MVC

Модель (MVC) – це набір класів, який включає в себе дані та алгоритми перевірки, пов’язані з конкретною предметною областю. У класичному виконанні MVC модель також містить у собі бізнес– логіку. Моделі можуть бути двох типів: активні та пасивні. Модель може змінитися про зміни свого стану, традиційно, використовуючи шаблон Спостерігач. Реалізація MVC підтверджує активну модель. Модель не залежить від View та Controller, і може працювати незалежно. Це є важливою вимогою для тестування.

Уявлення – це клас, який відповідає для відображення даних у графічному (хоча не обов’язково) форматі. У виконанні класичного MVC View має доступ до моделі тільки для читання. View не повинні змінювати стан моделі напряму. Цю відповідальність несе контролер.

Контролер керує реакцією на зовнішні подразники, виконуючи певну логіку, що може включити зміну стану моделі. Однак контролер не має

прямого зв’язку з виявленням і не зберігає свій власний стан, а також не відповідає для передачі даних між моделлю та виявленням.

Існує підхід до адаптації класичного патерну MVC під назвою Apple MVC (рис. 3.3), який задає фреймворки Cocoa та CocoaTouch для розробки додатків.

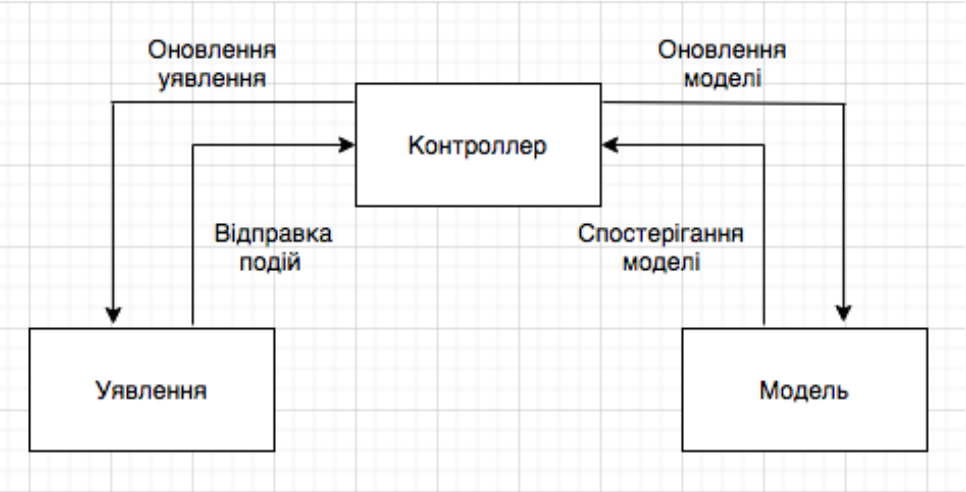

Рисунок 3.3 – Схема паттерну Apple MVC

Основними відмінностями Apple MVC від MVC є те, що контролер взаємодіє з виявленням, що посилає події на контролер, а той у своїй класичній версії оновлює оголошення. Контролер змінює стан моделі за потреби, а коли стан змінюється, контролер повідомляється і знову вирішує, як обробити ці зміни.

Для реалізації програми потрібно передавати та зберігати інформацію між користувачами, яка повинна бути передана в реальний час. Це можна зробити за допомогою хмарної бази даних. Однак на розробку такої системи потрібно багато часу та велика команда розробників для підтримки проекту.

Для кращого розуміння того, які компоненти повинні взаємодіяти, були створені різні діаграми UML, такі як діаграми прецедентів, діяльності та взаємодії. Вони представлені на рисунках 3.4-3.5.

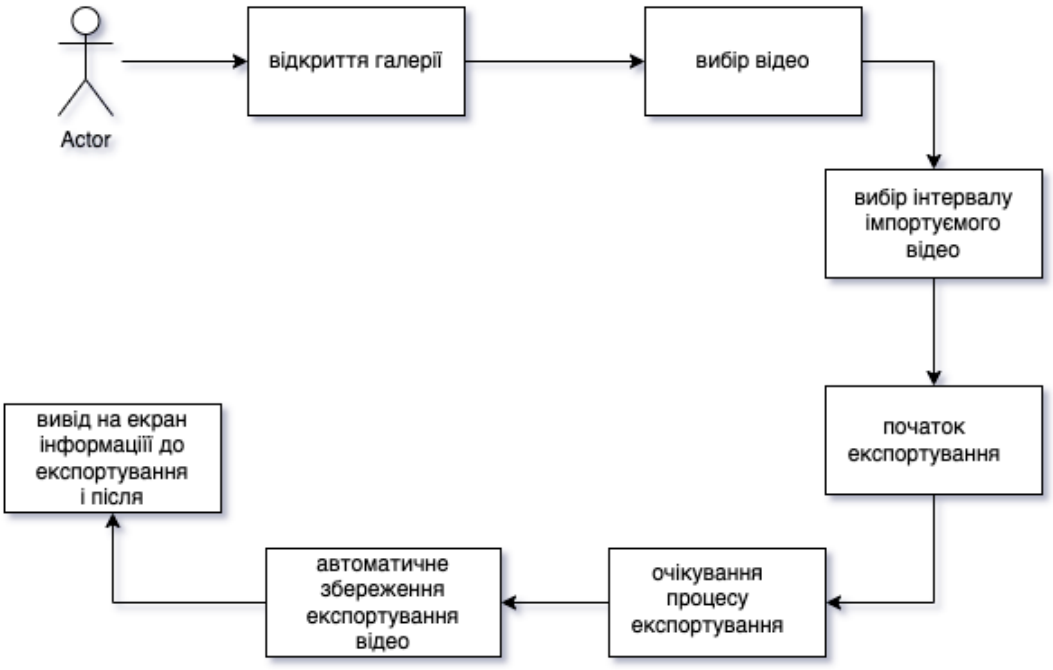

Рисунок 3.4 – Діаграма прецедентів

Існує лише один актор додатку – користувач. Він може вибрати буль– яке відео на його розсуд, наприклад, різної якості, з аудіо доріжкою чи без неї, великого розміру чи маленького. Після цього можливо вибрати інтервал імпортуємого відео. Потім запустити початок експортування. Відео після конвертації автоматично зберігається у галерею. Після конвертації з’являється інформація на екрані яка показує мета інформацію до початку конвертації та після неї для того, щоб побачити різницю.

3.2 Організація середовища для розробки додатку

Виходячи, з попереднього розділу було обрано у якості набору ПЗ наступні компоненти:

1. Операційна система MAC– OS Ventura 13.3;
2. Xcode Version 14.2 (14C18) – візуальна середа розробки;
3. CocoaPods Version 1.10.1 – бібліотека;
4. Sourcetree – 4.2.2 (250);

5. Terminal.

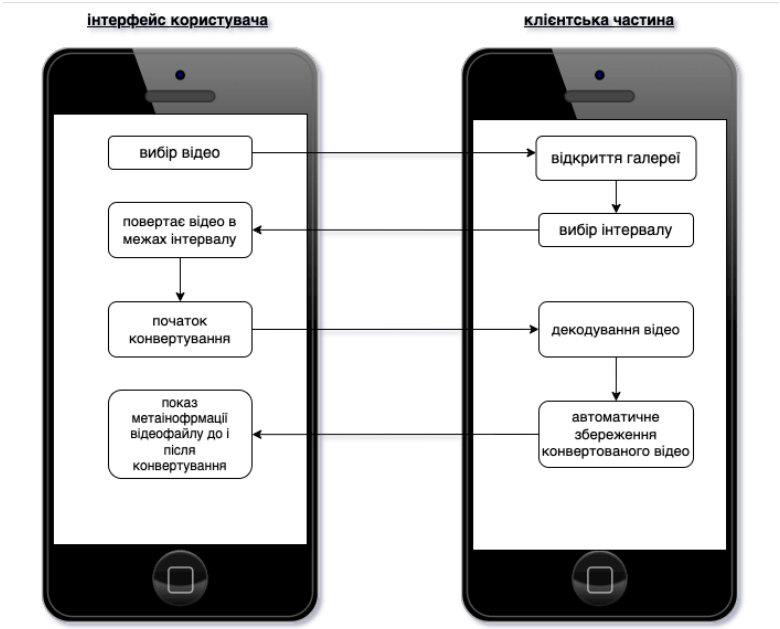

Рисунок 3.5 – Діаграма діяльності

3.3 Файлова структура додатку

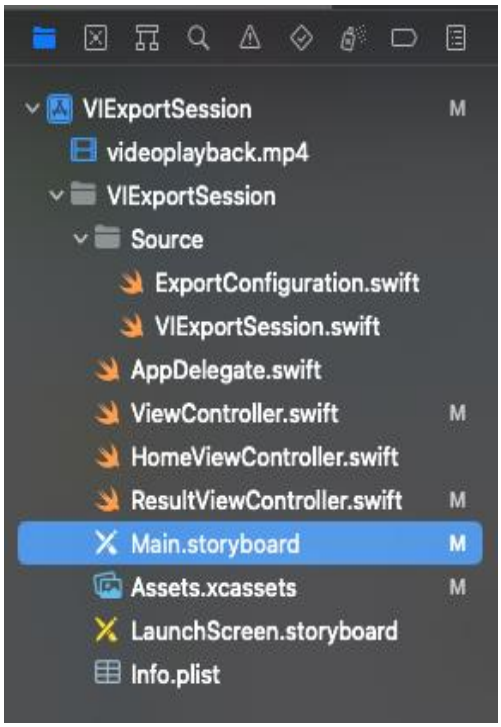

Рисунок 3.7 – Дерево вихідних файлів

3.4 Створення та опис інтерфейсної частини

Розглянемо інтерфейс додатку.

В ході над опрацюванням практичної частини курсової роботи були використані наступні компоненти:

- UILabel – помітка;
- UIButton – кнопка;
- UIImageView – зображення;
- UIViewController – вікно;
- UINavigationBar – панель навігації;
- ProgressView – вікно навігації;

Опис інтерфейсної частини:

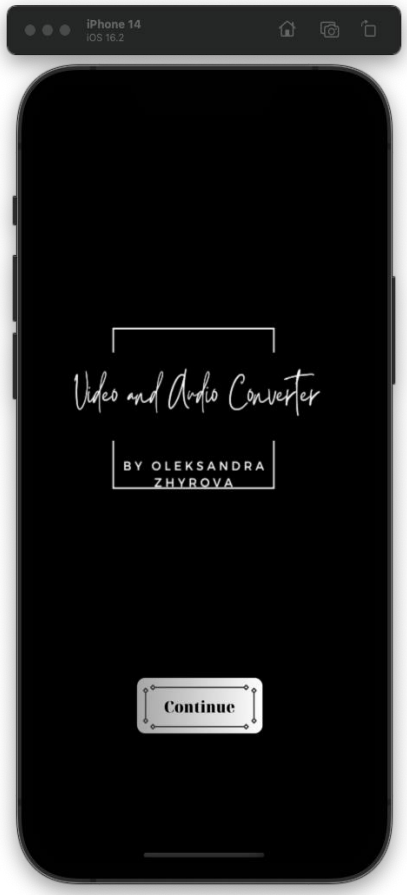

Рисунок 3.8 – Стартове вікно

При запуску програми відкривається стартове вікно (рис. 3.8). На цьому контролері зображений логотип та клавіша (Continue) для переходу до іншого контролера. Ця клавіша з анімацією, для того, щоб зробити таке потрібно додати відео до Xcode. Використані компоненти: UIButton, UILabel.

```
let filepath: String? = Bundle.main.path(forResource:
"videoplayback", ofType: "mp4")
let fileURL = URL.init(fileURLWithPath: filepath!)
```

Рисунок 3.9 – клавіша «Continue»

На другому контролері є системна клавіша  для вибору відео у галереї. Після вибору відео є можливість обрізки відео потрібного розміру, довжини відео.

```
let imagePicker = UIImagePickerController()
    imagePicker.allowsEditing = true
    imagePicker.sourceType = .photoLibrary
    imagePicker.mediaTypes = [kUTTypeMovie as String]
    imagePicker.delegate = self
    present(imagePicker,
              animated: true,
              completion: nil)
```

Рисунок 3.10 – Обрізка відео

Рисунок 3.11 – Другий контролер

Після вибору відео на екрані з’являється інформація про поточне не оброблене відео. А також дві клавіші: Export та Cancel. Використані компоненти: 2 UIButton, ProgressView, ImageView, UILabel, NavigationBar.

При натисканні клавіші «Export» виконується обробка відео в інший формат. Для відображення процесу обробки є індикатор (Status). Якщо під час обробки потрібно відмінити все, то для цього є клавіша «Cancel».

Є також і відображення відео зверху для того, щоб впевнитись, що саме це відео було попередньо, вибране у галереї. Під час конвертування відео відбувається декодування потоку даних відео фреймів, перекодування в інший формат, збереження у накопичувач.

```
var infoText = "duration: \(String(format: "%.2f", asset.duration.seconds))"
```

```
        let size = asset.tracks(withMediaType:
.video).first!.naturalSize
        infoText.append("\nresolution: \(size)")
        let framerate = asset.tracks(withMediaType:
.video).first!.nominalFrameRate
        infoText.append("\nframerate: \(String(format: "%.2f",
framerate))")
        let bitrate = asset.tracks(withMediaType:
.video).first!.estimatedDataRate
        infoText.append("\nbitrate: \(String(format: "%.2f", bitrate
/ 1000))kb")
        let transform = asset.tracks(withMediaType:
.video).first!.preferredTransform
        let angleDegress = atan2(transform.b, transform.a) * 180 /
CGFloat.pi
        infoText.append("\nangle degress: \(String(format: "%.0f",
angleDegress))")
```

Рисунок 3.12 – завершальний (третій) контролер

У третьому завершальному контролері (Total Information), можливо побачити показ мета інформації відеофайлу до та після конвертації. І для користувача одразу буде помітна різниця між відеофайлом до конвертації та після процесу конвертування. Використані компоненти: UILabel, ImageView, UIButton, NavigationBar.

```
    if let videoTrack = asset.tracks(withMediaType:
.video).first
    {
    infoText.append("\n Video info:")
    infoText.append("duration: \(String(format: "%.2f",
asset.duration.seconds))")

    let size = videoTrack.naturalSize
    infoText.append("\n resolution: \(size)")

    let framerate = videoTrack.nominalFrameRate
    infoText.append("\n framerate: \(String(format: "%.2f",
framerate))")

    let minFrameDuration = videoTrack.minFrameDuration.seconds
    infoText.append("\n minFrameDuration: \(String(format:
"%.2f", minFrameDuration))")

    let bitrate = videoTrack.estimatedDataRate
    infoText.append("\n bitrate: \(String(format: "%.2f",
bitrate / 1024/1024/8))Mb")

    let formatDescriptions = videoTrack.formatDescriptions.first
as! CMFormatDescription
    let subType =
CMFormatDescriptionGetMediaSubType(formatDescriptions)
    let codecString = stringWith(forCC: Int(subType))
    infoText.append("\n codec: \(codecString)")
```

```
     let naturalTimeScale = videoTrack.naturalTimeScale
     infoText.append("\n natural time scale: \(String(format:
"%.2f", naturalTimeScale))")
     print("")
       }
     if let audioTrack = asset.tracks(withMediaType:
.audio).first
             {
     infoText.append("\n Audio info")

     let bitrate = audioTrack.estimatedDataRate
     infoText.append("\nbitrate: \(String(format: "%.2f", bitrate
/ 1000))kb")

     let naturalTimeScale = audioTrack.naturalTimeScale
     infoText.append("\n natural time scale:
\(Int(naturalTimeScale))")

     let preferredVolume = audioTrack.preferredVolume
     infoText.append("\n preferred volume: \(preferredVolume)")

     let formatDescriptions = audioTrack.formatDescriptions.first
as! CMFormatDescription
     let subType =
CMFormatDescriptionGetMediaSubType(formatDescriptions)
     let codecString = stringWith(forCC: Int(subType))
     infoText.append("\n codec: \(codecString)")

     let asbd =
CMAudioFormatDescriptionGetStreamBasicDescription(formatDescript
ions)
     let sampleRate : Int = Int(asbd?.pointee.mSampleRate ?? 0)
     infoText.append("\n sampleRate: \(sampleRate)")
     }
      print(infoText)
```

```
return infoText
  }
```

3.5 Види тестування

Програм без помилок не існує. Деякі розробники самі є винуватцями помилок у програмах.

Програміст повинен не тільки писати ефективні програми, але і знаходити в них помилки. Обробка помилок є невід'ємною частиною розробки будь-якого програмного забезпечення. Обробка помилок в теорії звучить просто: згенеруйте декілька помилок і перехопіть їх, і ваша програма продовжить роботу. Але насправді зробити це правильно може бути досить складно. Помилки можуть бути різних форм та розмірів. Якщо запитати на приклад трьох розробників, що таке помилка, то можливо отримаєте три різні відповіді і, швидше за все, почуєте різні точки зору щодо винятків, помилок, проблем під час виконання або навіть звинувачення на адресу користувача через те, що він зіштовхнувся з проблемою.

Помилки діляться на три категорії:

– помилки програмування – до них відносяться, наприклад, вищі за межі масиви, поділ на нуль та цілочисельне переповнення. По суті це проблеми, які можна виправити на рівні коду. Саме тут модульні тести та забезпечення якості можуть врятувати ситуацію, коли розробник припускається помилки. Зазвичай у Swift можна використовувати на кшталт assert, щоб переконатися, що код працює, як і було задумано, і precondition, щоб інші знали, що API викликається правильно.

– Помилки користувача – це коли користувач взаємодіє з системою і не може правильно виконувати завдання. Помилки користувача можуть бути викликані неповним розумінням системи, коли людина відволіклася, або незграбним інтерфейсом користувача. Можливо запобігти цим проблемам за допомогою гарного дизайну, чіткої взаємодії та формування програмного

забезпечення таким чином, щоб це допомогло користувачам досягти того, чого вони хочуть.

– Помилки, виявлені під час виконання – ці помилки можуть бути пов'язані з неможливістю створення файлу додатком через переповнення жорсткого диска, збоєм мережного запиту, сертифікатами з терміном дії, що закінчився, JSON– парсерами, які висловлюють невдоволення після того, як їм надали невірні дані, і багатьом іншим, що може піти не так, коли програма працює. Остання категорія помилок відноситься до категорії виправних.

Існують декілька типів програмних помилок:

– синтаксичні помилки – помилки, що виникають через порушення правил мови програмування. Помилки виникають під час компіляції. Можуть бути виключені порівняно легко;

– семантичні помилки – ті помилки, які призводять до не дуже коректних обчислень або помилок під час виконання (run– time error).

За об'єктом тестування виділяють 12 видів тестування ПЗ:

– Функціональне тестування (functional testing) – процес перевірки функціональності програмного забезпечення відповідно до вимог та очікувань користувача, що дозволяє перевірити правильність роботи програми та можливість помилки та дефекти.

– Дослідницьке тестування (exploratory testing) – метод тестування програмного забезпечення, що забезпечує одночасну процедуру тестування та дослідження функціональності продукту, щоб знайти помилки та проблеми, які можуть залишитися непоміченими під час інших видів тестування. У ході дослідження тестер може вільно експериментувати з програмою та тестувати різні її аспекти, щоб знайти проблему та відхилитися від звичайного шаблону тестування.

– Тестування продуктивності (тестування продуктивності) – це вид тестування програмного забезпечення, що забезпечує оцінку швидкості, масштабованості, стійкості та інших аспектів продуктивності програмного забезпечення під ефективними навантаженнями та умовами використання.

Цей вид тестування дозволяє вирішити проблеми, пов’язані з продуктивністю програмного забезпечення, такі як вільна відповідь, збої, системи перезавантаження тощо, та забезпечує можливість оптимізувати продукт перед його випуском.

– Навантажувальне тестування (load testing) – це вид тестування програмного забезпечення, який дозволяє використовувати, як програма працює за умов великого навантаження на систему. Цей вид тестування з використанням оцінки максимального навантаження, така система може витримати без збоїв, вибраних проблем, пов’язаних з продуктивністю, що виконується за умов великого навантаження. Навантажувальне тестування дозволяє застосувати, як програмне забезпечення працює в реальних умовах використання, та забезпечити стабільну роботу системи під час максимального завантаження.

– Димне тестування (smoke testing) – вид тестування програмного забезпечення, який передбачає проведення невеликого набору тестів для перевірки основної функціональності програми після її змін. Цей вид тестування протягом тривалого часу перед запуском більш повного набору тестів, щоб переконатися, що в правильній роботі програми виявлені критичні проблеми. Індивідуум тестування дозволяє швидко виявити серйозні проблеми та дефекти, які можуть призвести до збою програми під час її роботи, та запобігти підвищеним витратам часу та ресурсів на тестування.

– Стрес– тестування (стрес– тестування) – вид тестування програмного забезпечення, що створює навантаження на систему за межами її нормального режиму роботи з призначенням визначення її меж витривалості та стійкості. Вид тестування дозволяє перевірити, як програмне забезпечення працює за умов високого навантаження та стресових ситуацій, які можуть виникати в реальних умовах використання.

– Тестування стабільності (testing testing) – вид тестування програмного забезпечення, що перевіряє його здатність працювати стійко протягом тривалого періоду часу в різних умовах та при різних навантаженнях.

Тестування спрямоване на створення можливих цих дефектів, які можуть призвести до збоїв системи під час тривалої роботи, а також на визначення того, як швидко система може відновитися після збою.

– Тестування зручності використання (usability testing) – вид тестування програмного забезпечення, що спрямований на визначення того, наскільки легко та зручно користувачі можуть взаємодіяти з програмою. Тестування оцінює, які і наскільки інтуїтивно зрозумілі та зручні для користувачів інтерфейс та функціональні програми.

– Тестування інтерфейсу користувача (UI testing) – процес перевірки функціональності та коректності роботи інтерфейсу програмного забезпечення. У тестуванні перевіряються такі аспекти, як взаємодія користувача з графічним інтерфейсом, правильність розміщення елементів інтерфейсу, відповідність кольорів та шрифтів вимогам дизайну.

– Тестування безпеки (security testing) – процес перевірки програмного забезпечення на вразливість до виявлених атак та зловмисних дій. У тестуванні перевіряються програми щодо захисту від різноманітних загроз, наприклад, включення в систему, крадіжка конфіденційної інформації, злам паролів та ін.

– Тестування локалізації (localization testing) – процес програмного забезпечення щодо його можливості коректно працювати в різних перевірках локалізації та мов. У тестуванні перевіряється правильність відтворення та роботи ПЗ з урахуванням мовних особливостей локалізації.

– Тестування сумісності (compatibility testing) – процес програмного забезпечення на його здатність працює правильно в різних середовищах перевірки. Під час тестування перевіряється сумісність, як програмне забезпечення працює з різними системами, усілякими браузерами, апаратним забезпеченням, з якими можливо взаємодіяти.

У виладенні вищезазначеного можна сказати, що тестування має дві мети:

1. Показати розробнику і клієнту, що ПО відповідає заявленим вимогам. З точки зору клієнта це означає, що для кожної функціональності, бажаної з його боку і записаної в документі вимог, проведено як мінімум в один тест (як правило, звичайно, більше). У разі загальнодоступного програмного забезпечення то, що у програмному забезпеченні протестовані всі задані основні властивості. Відповідний тест називається – валідація (перевірка достовірності). Успішна валідація вказує, що система працює як треба.

2. Знайти ситуації, програмне забезпечення поводиться не так, небажано або не відповідає специфікації. Отже, пошук помилок в цьому сенсі призначений для того, щоб ліквідувати небажану поведінку системи, як наприклад, крах системи, небажана взаємодія з іншими системами, неправильні розрахунки, пошкоджені дані. Виконує це завдання тестування іменується тестуванням дефектів (detect testing).

Тут є успішним тест, який показує дію помилки системи, або ж іншими словами, знаходить у системі помилку (до подальших виправлень якої, і приступають).

Наприклад один специфічний аспект обробки помилок у Swift полягає у тому, що функції не показують які помилки вони можуть генерувати. Функція, позначена як `throws`, теоретично не може генерувати помилки або декілька мільйонів помилок і це не можна дізнатися, подивившись на сигнатуру функції.

Відсутність явного перерахування та обробки кожної помилки дає розробнику гнучкість, але суттєвим недоліком є те, що не можна швидко дізнатися, які помилки може викликати або розповсюджувати функцію.

Функції не показують своїх помилок, тому рекомендується надавати якусь інформацію там, де це можливо. Але за допомогою Quick Help на ОС MacOS може допомогти надати додаткову інформацію про помилки, які розробник може генерувати.

3.6 Тестування додатку

Я надаю особливу увагу якості продукту та проведення тестування на наявність різних видів помилок, таких як технічні, візуальні, логічні та інші. Це є необхідною та відповідною частиною процесу розробки програмного продукту.

Перед розробкою додатків для iOS аналітики досліджують бізнес–ринку та конкуренції, цільової аудиторії та актуальних трендів. Далі повідомляються основні цілі та завдання програми, що формує стратегію розробки проекту. Уважна та якісна розробка на кожному етапі є запорукою успішного результату.

Розробка програми для iOS вимагає дотримання вимог Apple та особливостей цієї платформи. Важливо, щоб все, що було заплановано на етапі проектування та дизайну, працювало стабільно.

Процес тестування має величезне значення, відсутність неналежної роботи програми може призвести до втрати користувачів. Тому ми приділяємо цьому процесу особливу увагу та виділяємо необхідний ресурс. Фахівець з тестування залучається до роботи з самого початку, починаючи з передачі розробнику схем навігації та дизайну макетів екранів.

На етапі запуску додатка ми проводимо тестування на наявність технічних та візуальних помилок, вносимо додаткові правки та доводимо додаток до повної готовності. Після цього ми опублікуємо готовий продукт в App Store і правильною публікацією та складанням якісних описів для кращого розміщення в магазинах.

3.7 Висновки до третього розділу

Розробка програми – це комплексний процес включає в себе багато різних стадій і вимагає певної кількості умінь і навичок в області проектування, програмування і тестування.

РОЗДІЛ 4 ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ПРОЕКТУ

4.1 Бізнес-план розробки програмного забезпечення

У стадії розробки бізнес– плану на розробку програмного забезпечення – додатку для iOS з назвою VIA Converter.

Даний додаток, призначений для звичайних користувачів і компаній, із можливістю розширення функціоналу та отримання прибутку від його використання.

VIA Converter – це відео/аудіо конвертер, який дозволяє конвертувати файли відео та аудіо в різні формати, а також містить інші корисні функції.

4.1.1 Резюме

– Послуги, які надаються, полягають у створених відповідних додатках для обробки відео та інформатизації та відповідають певним КВЕДам, а саме: КВЕД 62.01 (Комп’ютерне програмування), КВЕД 62.02 (Консультації з питань інформатизації) та КВЕД 58.29 (Видання іншого програмного забезпечення) .

– Цільова аудиторія продукту включає користувачів, які регулярно працюють з мультимедіа– файлами, а також компанії. Визначення цільової аудиторії допоможе у просуванні продукту та розробці ефективної рекламної кампанії.

– Для реалізації проекту необхідно залучити інвестиції розміром 1 500 000 гривень. Гроші можуть бути залучені від інвесторів, які зацікавлені в проектах. Перші продажі не плануються протягом першого року, вони стануть можливими лише на другому році після повної реалізації продукту.

– Для створення продукту необхідна фронтенд– команда з двох людей, менеджер проекту та команда промоутерів з двох людей. Інвестор має

замовити продукт та узгодити всі деталі з менеджером проекту, після чого надати підтримку проекту.

– Загальна вартість додаткової реалізації становить близько 1 000 000 гривень, а підтримка та рефакторинг оцінюється в 100 000 гривень на місяць.

При реалізації проекту середня ціна становить 1 000 000 гривень, а час реалізації – від 6 до 10 місяців. Загальний прибуток без податків становить близько 100 000 гривень.

4.1.2 Маркетинг

Для того, щоб задовольнити користувача потрібно зробити зручним інтерфейс. Багато часу буде присвячено розробці дизайну. Швидкодія та надійність є дуже важливими пунктами. Цим повинні займатися команда бекенд, не менш важливим є також, читабельність коду. Тому потрібні фахівці, а саме (senior). Головним недоліком є велика конкурентність на ринку.

Просуванням додатку буде займатись команда із промоутерів. Після закінчення проекту будуть створені реклами на різних сайтах, пізніше буде створений власний сайт для перегляду інформації про застосунок та його скачування.

4.1.3 Обгрунтування необхідних фінансових вкладень

Через війну в країні багато спеціалістів працюють дистанційно. Тому оренда офісу не відіграє ніякої ролі.

Режим роботи – повна або не повна зайнятість. Перші гроші, які поступлять на рахунок, потрібно буде витратити на оплату фахівців та пошук кадрів.

4.1.4 Нормативно – правові нюанси

Юридичний статус – ФОП (фізичний підприємець третьої групи).

Переваги ФОП:

- низьке держмито;
- розмір штрафів;
- спрощена ліквідація;
- робота по схемі єдиного податку.

Недоліки ФОП:

- можливий арешт майна;
- обмеження по видам діяльності;
- ліміти доходів ФОП.

Враховуючи всі недоліки і переваги ведення ФОП описані вище, хоч і приватна підприємницька форма ведення бізнесу є прекрасним вибором для більшості фізичних осіб, які здійснюють, наприклад, не ризиковану підприємницьку діяльність (наприклад, у сфері IT технологій), ця структура є хорошим вибором для компаній і стартапів, які планують рости чи мають високі бізнес– ризики і потенційні зобов'язання.

4.1.5 Складання фінансового бюджету

Для забезпечення фінансування проекту розміром 1500000 грн передбачається можливість залучення інвесторів або кредитних коштів.

Витрати на проект оцінюється розміром 1000000 грн, а перший прибуток очікується через рік.

Особа, яка займається проектом, є фізичною особою– підприємцем третьої групи оподаткування.

Податок на прибуток становить 3% або 5%, залежно від суми доходу, і становить 1054,66 грн на місяць.

Термін сплати податків становить 10 календарних днів згідно з положеннями П.295.3 КПУ.

4.1.6 Оцінка можливих ризиків проекту

Можливість провалу проекту через сильну конкуренцію існує. На мою думку, одним із способів зменшення цього ризику можна витратити додаткові кошти на рекламу продукту. Якщо виникнути збитки, їх можна покрити шляхом підвищення ціни на внутрішню добавку продукту.

ВИСНОВКИ

Під час написання дипломної роботи вивчені сучасні технології конвертації відео, які значно спрощують робочі процеси та підвищують їх ефективність.

У висновку можна зробити висновок, що доцільніше підібрати певний тип видеоконвертера відповідно до поставлених завдань.

Для написання дипломної роботи було використано операційну систему Mac OS, інтегровану мову програмування Swift та середовище розробки Xcode.

Було створено клієнтську частину додатку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стивен Кочин, Програмування на Swift, [ориг. мова Англійська] видавництво 2004 р. Вашингтон – 201 ст., ст. 90–91. Дата перегляду 17.04.2023 р.
2. Скотт Кнастер, Марк Далримпл, Swift 2.0 і програмування для MAC OS [переклад з англ.] – видавництво у New York 2010 р. – 291ст., ст. 182–183. Дата перегляду 17.04.2023 р.
3. Garry Bennet, Brod Less, Swift for Absolute Beginners [переклад з англ.], видавництво USA, 2014 р. – 303 ст., с. 23–27. Дата перегляду 22.04.2023 р.
4. Jonathon Manning, Tim Nugent, Learning Cocoa with Swift and Objective-C [ориг. мова англійська], видавництво USA, 2005 р. – 321 ст., ст. 116. Дата перегляду 11.04.2023 р.
5. Amit Chaudhary, Swift Create Sheet [перев. з індійськ.] – видавництво India 2001 р. – 271 ст., ст. 133– 135. Дата перегляду 11.04.2023 року.
6. Beta Testing Made Simple with TestFlight [deweloper.apple.com] // External testers and groups. Режим доступу: www, http://deweloper.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjective
7. Waqar Malik, Mark Dalrymple, Learn Swift on the Mac [перекл. з англ.]– видавництво USA 2007 р. – 311 ст., ст. 67– 69 Дата перегляду 14.04.2023 р.
8. Platform Support [swift.org] // Deployment and Development. Режим доступу www, https://www.swift.org/platform– support/
9. AVVideoCodecType [developer.apple.com] // TypeALias. Режим доступу www, https://developer.apple.com/documentation/avfoundation/avvideocodectype?language=objc Дата перегляду 14.04.2023 р.

10. Custom Encoder/Decoder supporting [forums.swift.org] // RawRepresentable. Режим доступу www, https://forums.swift.org/t/custom– encoder– decoder– supporting– rawrepresentable/60344
11. Swift – Initialize model object with init [stack overflow] // (from decoder:). Режим доступу www, https://stackoverflow.com/questions/68213705/swift– initialise– model– object– with– initfrom– decoder
12. The Swift Programming Language [перев. з англ.], видавництво 2011 року, 422 ст., ст. 301– 312. Дата перегляду 19.04.23–20.04.23 р.

ДОДАТОК А

ExportConfiguration.swift

```
//
//  ExportConfiguration.swift
//  AVIConverter
//
//  Copyright © 2023 oleksandrazhyrova. All rights reserved.
//

import AVFoundation

public class ExportConfiguration {
    public var outputURL = URL.temporaryExportURL()
    public var fileType: AVFileType = .mp4
    public var shouldOptimizeForNetworkUse = false
    public var metadata: [AVMetadataItem] = []
}

public class VideoConfiguration {
    // Video settings see AVVideoSettings.h
    public var videoInputSetting: [String: Any]?
    public var videoOutputSetting: [String: Any]?
    public var videoComposition: AVVideoComposition?
}

public class AudioConfiguration {
    // Audio settings see AVAudioSettings.h
    public var audioInputSetting: [String: Any]?
    public var audioOutputSetting: [String: Any]?
    public var audioMix: AVAudioMix?
 public var audioTimePitchAlgorithm:
AVAudioTimePitchAlgorithm?
}
```

```
// MARK: – Helper

fileprivate extension URL {
    static func temporaryExportURL() – > URL {
        let documentDirectory = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).last!
    let filename = ProcessInfo.processInfo.globallyUniqueString + ".mp4"
  return documentDirectory.appendingPathComponent(filename)
    }
}
```

ExportSession.swift

```
//
//  ExportSession.swift
//  ExportSession
//
//  Copyright © 2023 oleksandrazhyrova. All rights reserved.
//

import AVFoundation

public class VIExportSession {

public private(set) var asset: AVAsset
public var exportConfiguration = ExportConfiguration()
public var videoConfiguration = VideoConfiguration()
public var audioConfiguration = AudioConfiguration()

fileprivate var reader: AVAssetReader!
fileprivate var videoOutput: AVAssetReaderVideoCompositionOutput?
fileprivate var audioOutput: AVAssetReaderAudioMixOutput?
fileprivate var writer: AVAssetWriter!
```

```
        fileprivate var videoInput: AVAssetWriterInput?
        fileprivate var audioInput: AVAssetWriterInput?
        fileprivate var inputQueue = DispatchQueue(label:
"VideoEncoderQueue")

            // MARK: – Exporting properties
            public var progress: Float = 0 {
                didSet {
                    progressHandler?(progress)
                }
            }
            public var videoProgress: Float = 0 {
                didSet {
                    if audioInput != nil {
          progress = 0.95 * videoProgress + 0.05 * audioProgress
                    }
        else
         {
                      progress = videoProgress
                    }
                }
              }
            public var audioProgress: Float = 0 {
                didSet {
                    if videoInput != nil {
            progress = 0.95 * videoProgress + 0.05 * audioProgress
                    }
        else
        {
                        progress = audioProgress
                    }
                }
            }

            public var progressHandler: ((Float) - > Void)?
            public var completionHandler: ((Error?) - > Void)?
```

```
            fileprivate var videoCompleted = false
            fileprivate var audioCompleted = false

            public init(asset: AVAsset) {
                self.asset = asset
            }


        // MARK: – Main

            public func cancelExport() {
                if let writer = writer, let reader = reader {
                    inputQueue.async {
                        writer.cancelWriting()
                        reader.cancelReading()
                    }
                }
            }
            public func export() {
                cancelExport()
                reset()
                do {
                    reader = try AVAssetReader(asset: asset)
  writer = try AVAssetWriter(url:
exportConfiguration.outputURL,fileType:
exportConfiguration.fileType)
      writer.shouldOptimizeForNetworkUse=
exportConfiguration.shouldOptimizeForNetworkUse
             writer.metadata = exportConfiguration.metadata

                    // Video output

let videoTracks = asset.tracks(withMediaType: .video)
                    if videoTracks.count > 0 {
                    if videoConfiguration.videoOutputSetting == nil
```

```
      {
      videoConfiguration.videoOutputSetting=
buildDefaultVideoOutputSetting(videoTrack: videoTracks.first!)
             }

      let videoOutput=
AVAssetReaderVideoCompositionOutput(videoTracks: videoTracks,
videoSettings: videoConfiguration.videoInputSetting)
videoOutput.alwaysCopiesSampleData = false
videoOutput.videoComposition =
videoConfiguration.videoComposition
                if videoOutput.videoComposition == nil
      {
        videoOutput.videoComposition =
buildDefaultVideoComposition(with: asset)
                     }

         guard reader.canAdd(videoOutput)
      else {
          throw NSError(domain: "com.exportsession", code: 0,
userInfo: [NSLocalizedDescriptionKey: NSLocalizedString("Can't
add video output", comment: "")])
                     }
                     reader.add(videoOutput)
                     self.videoOutput = videoOutput

                     // Video input

 let videoInput = AVAssetWriterInput(mediaType: .video,
outputSettings: videoConfiguration.videoOutputSetting)
                videoInput.expectsMediaDataInRealTime = false
                     guard writer.canAdd(videoInput)
      else
       {
 throw
NSError(domain: "com.exportsession",
```

```
            code: 0,
        userInfo: [NSLocalizedDescriptionKey:
      NSLocalizedString("Can’t add video input", comment: "")])
                        }
                    writer.add(videoInput)
                    self.videoInput = videoInput
                    }

                    // Audio output

         let audioTracks = asset.tracks(withMediaType: .audio)
                  if audioTracks.count > 0 {
                  if audioConfiguration.audioOutputSetting == nil
        {
audioConfiguration.audioOutputSetting =
buildDefaultAudioOutputSetting()
                        }
let audioOutput = AVAssetReaderAudioMixOutput(audioTracks:
audioTracks, audioSettings:
audioConfiguration.audioInputSetting)
              audioOutput.alwaysCopiesSampleData = false
              audioOutput.audioMix = audioConfiguration.audioMix

if let audioTimePitchAlgorithm =
audioConfiguration.audioTimePitchAlgorithm
         {
audioOutput.audioTimePitchAlgorithm = audioTimePitchAlgorithm
                        }
           if reader.canAdd(audioOutput)
         {
                      reader.add(audioOutput)
                    self.audioOutput = audioOutput
                        }

           if self.audioOutput != nil
        {
```

```
                        // Audio input

let audioInput = AVAssetWriterInput(mediaType: .audio,
outputSettings: audioConfiguration.audioOutputSetting)
audioInput.expectsMediaDataInRealTime = false
                        if writer.canAdd(audioInput)
       {
                            writer.add(audioInput)
                            self.audioInput = audioInput
                        }
                    }
                }

      writer.startWriting()
      reader.startReading()
      writer.startSession(atSourceTime: kCMTimeZero)

       encodeVideoData()
       encodeAudioData()
              }
      catch
       {
              self.completionHandler?(error)
              }
          }
      fileprivate func encodeVideoData() {
if let videoInput = videoInput {
videoInput.requestMediaDataWhenReady(on: inputQueue,
using: { [weak self] in
          guard let strongSelf = self else { return }
          guard let videoOutput = strongSelf.videoOutput,
      let videoInput = strongSelf.videoInput
      else
      { return }
      strongSelf.encodeReadySamplesFrom(output: videoOutput, to:
videoInput, completion: {
```

```
                strongSelf.videoCompleted = true
                strongSelf.tryFinish()
                     })
                 })
            }
      else
       {
         videoCompleted = true
         tryFinish()
              }
          }
fileprivate func encodeAudioData() {
if let audioInput = audioInput {
             audioInput.requestMediaDataWhenReady(on: inputQueue,
using: { [weak self] in
      guard let strongSelf = self
      else
      { return }
      guard  let  audioOutput  =  strongSelf.audioOutput,  let
audioInput = strongSelf.audioInput
      else
      { return }
      strongSelf.encodeReadySamplesFrom(output: audioOutput, to:
audioInput, completion: {
       strongSelf.audioCompleted = true
       strongSelf.tryFinish()
                     })
                })
            }
      else
       {
         audioCompleted = true
         tryFinish()
              }
          }
```

```
private var lastVideoSamplePresentationTime = kCMTimeZero
private var lastAudioSamplePresentationTime = kCMTimeZero

fileprivate func encodeReadySamplesFrom(output:
AVAssetReaderOutput, to input: AVAssetWriterInput,
completion: @escaping () - > Void)
 {
      while input.isReadyForMoreMediaData {
let complete = autoreleasepool(invoking: { [weak self] () - >
Bool in
      guard let strongSelf = self
      else
      { return true }
           if let sampleBuffer = output.copyNextSampleBuffer()
       {
      guard strongSelf.reader.status == .reading &&
strongSelf.writer.status == .writing
      else {
         return true
             }

guard input.append(sampleBuffer)
 else {
      return true
                }
if let videoOutput = strongSelf.videoOutput, videoOutput ==
output
 {
     lastVideoSamplePresentationTime =
CMSampleBufferGetPresentationTimeStamp(sampleBuffer)
      if strongSelf.asset.duration.seconds > 0 {
        strongSelf.videoProgress =
Float(lastVideoSamplePresentationTime.seconds /
strongSelf.asset.duration.seconds)
            }
      else
```

```
        {
             strongSelf.videoProgress = 1
             }
         }
       else
       {
        if let audioOutput = strongSelf.audioOutput, audioOutput
== output {
          lastAudioSamplePresentationTime =
CMSampleBufferGetPresentationTimeStamp(sampleBuffer)
          if strongSelf.asset.duration.seconds > 0 {
strongSelf.audioProgress =
Float(lastAudioSamplePresentationTime.seconds /
strongSelf.asset.duration.seconds)
           }
       else
        {
            strongSelf.audioProgress = 1
                  }
           }
        } else
       {
             input.markAsFinished()
             return true
              }
            return false
           })
             if complete {
             completion()
             break
          }
         }
       }
        fileprivate func tryFinish() {
        objc_sync_enter(self)
       defer { objc_sync_exit(self) }
```

```
            if audioCompleted && videoCompleted {
            if reader.status == .cancelled ||
               writer.status == .cancelled {
           finish()
                    }
       else
        if writer.status == .failed {
               finish()
       }
       else
       if reader.status == .failed {
                       writer.cancelWriting()
                 finish()
            } else
       {
             writer.finishWriting { [weak self] in
          guard let strongSelf = self else { return }
                  strongSelf.finish()
                       }
                   }
               }
           }

       fileprivate func finish() {
       if writer.status == .failed || reader.status == .failed {
         try? FileManager.default.removeItem(at:
exportConfiguration.outputURL)
               }
       let error = writer.error ?? reader.error
       completionHandler?(error)
           reset()
           }
       fileprivate func reset() {
          videoCompleted = false
          videoCompleted = false
          videoProgress  = 0
```

```
          audioProgress  = 0
          progress       = 0

          reader      = nil
          videoOutput = nil
          audioInput  = nil
          writer      = nil
          videoInput  = nil
          audioInput  = nil
           }
       }
extension VIExportSession {
fileprivate   func   buildDefaultVideoComposition(with   asset:
AVAsset) - > AVVideoComposition {
     let videoComposition = AVMutableVideoComposition()
       if   let   videoTrack   =   asset.tracks(withMediaType:
.video).first {
      let trackFrameRate = videoTrack.nominalFrameRate
      videoComposition.frameDuration = CMTime(value: 1, timescale:
CMTimeScale(trackFrameRate))

            var naturalSize = videoTrack.naturalSize
            var transform = videoTrack.preferredTransform
            let angle = atan2(transform.b, transform.a)
            let videoAngleInDegree = angle * 180 / CGFloat.pi
          if videoAngleInDegree == 90 || videoAngleInDegree == - 90
{
             let width = naturalSize.width
            naturalSize.width = naturalSize.height
            naturalSize.height = width
           }
         videoComposition.renderSize = naturalSize
         var targetSize = naturalSize
      if let width =
videoConfiguration.videoOutputSetting?[AVVideoWidthKey] as?
NSNumber {
```

```
        targetSize.width = CGFloat(width.floatValue)
   }
         if let height =
videoConfiguration.videoOutputSetting?[AVVideoHeightKey] as?
NSNumber {
        targetSize.height = CGFloat(height.floatValue)
                }
// Center
 if naturalSize.width > 0 && naturalSize.height > 0 {
       let xratio = targetSize.width / naturalSize.width
       let yratio = targetSize.height / naturalSize.height
       let ratio = min(xratio, yratio)
       let postWidth = naturalSize.width * ratio
       let postHeight = naturalSize.height * ratio
       let transx = (targetSize.width -  postWidth) * 0.5
       let transy = (targetSize.height -  postHeight) * 0.5
       var matrix = CGAffineTransform(translationX:
      transx/xratio, y: transy / yratio)
       matrix = matrix.scaledBy(x: ratio / xratio, y: ratio /
yratio)
       transform = transform.concatenating(matrix)
               }
       let passThroughInstruction =
AVMutableVideoCompositionInstruction()
       passThroughInstruction.timeRange =
CMTimeRangeMake(kCMTimeZero, asset.duration)
       let passThroughLayer =
AVMutableVideoCompositionLayerInstruction(assetTrack:
videoTrack)
       passThroughLayer.setTransform(transform, at: kCMTimeZero)
       passThroughInstruction.layerInstructions                    =
[passThroughLayer]
       videoComposition.instructions = [passThroughInstruction]
           }
        return videoComposition
 }
```

```
      fileprivate func buildDefaultVideoOutputSetting(videoTrack:
AVAssetTrack) - > [String: Any] {
      let trackDimensions = { () - > CGSize in
      var trackDimensions = videoTrack.naturalSize
      let videoAngleInDegree =
atan2(videoTrack.preferredTransform.b,
videoTrack.preferredTransform.a) * 180.0 / CGFloat(Double.pi)
      if abs(videoAngleInDegree) == 90 {
let width = trackDimensions.width
trackDimensions.width = trackDimensions.height
trackDimensions.height = width
       }
     return trackDimensions
}()
     var videoSettings: [String : Any] = [
      AVVideoWidthKey: trackDimensions.width,
      AVVideoHeightKey: trackDimensions.height,]
          if #available(iOS 11.0, *) {
      videoSettings[AVVideoCodecKey] = AVVideoCodecType.h264}
      else
      {
          videoSettings[AVVideoCodecKey] = AVVideoCodecH264
              }
         return videoSettings
          }

fileprivate func buildDefaultAudioOutputSetting() - > [String:
Any] {
      var stereo_ChannelLayoutVideo = AudioChannelLayout()
      memset(&stereo_ChannelLayoutVideo, 0,
MemoryLayout<AudioChannelLayout>.size)
      stereo_ChannelLayoutVideo.mChannelLayoutTag =
kAudioChannelLayoutTag_Stereo
      let channelLayoutAsData = Data(bytes:
&stereo_ChannelLayoutVideo, count:
MemoryLayout<AudioChannelLayout>.size)
```

```
        let compressionAudioSettings: [String: Any] = [
          AVFormatIDKey: kAudioFormatMPEG4AAC,
          AVEncoderVideoBitRateKey: 128000,
          AVSampleVideoRateKey: 44100,
          AVChannelLayoutKey: channelLayoutAsData,
          AVNumberOfChannelsKey: 2]
        return compressionAudioSettings
    }}
```

ViewController.swift

```
        //
        //  ViewController.swift
        //  ExportSession
        //
        //  Created by oleksandrazhyrova on 30/04/2023.
        //  Copyright © 2023 oleksandrazhyrova. All rights
reserved.
        //

        import UIKit
        import AVFoundation
        import Photos
        import MobileCoreServices

        class ViewController: UIViewController {

            @IBOutlet weak var _progressView: UIProgressView!
            @IBOutlet weak var _statusLabel: UILabel!
            @IBOutlet weak var _exportButton: UIButton!
            @IBOutlet weak var _cancelButton: UIButton!

            @IBOutlet weak var coverImageView: UIImageView!
            @IBOutlet weak var fileInfoLabel: UILabel!
```

```
        private var exportSession: VIExportSession!

        fileprivate var pickedAsset: AVAsset?

        override func viewDidLoad() {
        super.viewDidLoad()
        }

    override func viewDidAppear(_ animated: Bool) {
      super.viewDidAppear(animated)
       PHPhotoLibrary.requestAuthorization { status in
          print(status)
            }
        }

    @IBAction func addAction(_ sender: Any) {
      let imagePicker = UIImagePickerController()
          imagePicker.allowsEditing = true
          imagePicker.sourceType = .photoLibrary
          imagePicker.mediaTypes = [kUTTypeMovie as String]
          imagePicker.delegate = self
      present(imagePicker, animated: true, completion: nil)
        }

     @IBAction func exportAction(_ sender: UIButton) {
         guard let asset = pickedAsset
    else {
             return
          }
      exportSession = VIExportSession.init(asset: asset)
      if let track = asset.tracks(withMediaType: .video).first
{
            configureExportConfiguration(videoTrack: track)
            }
       exportButton.isEnabled = false
```

```
         exportSession.progressHandler = { [weak self] (progress)
in
         guard let strongSelf = self
      else
      { return }
      DispatchQueue.main.async {
strongSelf.progressView.progress = progress
strongSelf.statusLabel.text = "Exporting \(Int(progress * 100))%"
                }
             }
exportSession.completionHandler = { [weak self] (error) in
       guard let strongSelf = self
      else
      { return }
       DispatchQueue.main.async {
        if let error = error {
       strongSelf.statusLabel.text = error.localizedDescription
          }
      else
       {
          strongSelf.statusLabel.text = "Finished"
          strongSelf.saveFileToPhotos(fileURL:
          strongSelf.exportSession.exportConfiguration.outputURL
          )
              }
          strongSelf.exportButton.isEnabled = true
         }
     }
        exportSession.export()
       cancelButton.isHidden = false;
          }

      @IBAction func cancelAction(_ sender: UIButton) {
        exportSession.cancelExport()
        cancelButton.isHidden = true;
}
```

```
 private func saveFileToPhotos(fileURL: URL) {
     PHPhotoLibrary.shared().performChanges({
     PHAssetChangeRequest.creationRequestForAssetFromVideo(atFil
     eURL: fileURL)
          })
      { [weak self] (saved, error) in

DispatchQueue.main.async {
         guard let self = self else { return }
          if saved
            {
      ResultViewController.init(avAsset1: self.pickedAsset,
 avAsset2: AVAsset(url:
self.exportSession.exportConfiguration.outputURL))

let storyboard = UIStoryboard(name: "Main", bundle: nil)
if let vc = storyboard.instantiateViewController(withIdentifier:
"ResultViewController") as? ResultViewController
     {
         vc.avAsset1 = self.pickedAsset
         vc.avAsset2 = AVAsset(url:
     self.exportSession.exportConfiguration.outputURL)
     self.navigationController?.show(vc, sender: nil)
             }
      }
else
      {
      let errorMessage = error?.localizedDescription ?? ""
      let alertController = UIAlertController(title: " Video
can’t save to Photos.app, error: \(errorMessage)",
message: nil,  preferredStyle: .alert)
      let defaultAction = UIAlertAction(title: "OK", style:
.default, andler: nil)
            alertController.addAction(defaultAction)
             self.present(alertController,
                                animated: true,
```

```
                                completion: nil)
            }
          }
        }
      }

          // MARK: – Helper

func configureExportConfiguration(videoTrack: AVAssetTrack) {
     exportSession.videoConfiguration.videoOutputSetting =
      {
          let frameRate_video = 30;
          let bitrate_video = min(200000,
      videoTrack.estimatedDataRate)
          let trackDimensions_video = videoTrack.naturalSize
          let compressionSettings: [String: Any] = [
               AVVideoAverageBitRateKey: bitrate,
               AVVideoMaxKeyFrameIntervalKey: 30,
     AVVideoProfileLevelKey:
     AVVideoProfileLevelH264HighAutoLevel]
      var videoSettings: [String : Any] = [
         AVVideoWidthKey: trackDimensions.width,
         AVVideoHeightKey: trackDimensions.height,
         AVVideoCompressionPropertiesKey: compressionSettings]
             if #available(iOS 11.0, *) {
       videoSettings[AVVideoCodecKey] =  AVVideoCodecType.h264
         }
      else
       {
      videoSettings[AVVideoCodecKey] =  AVVideoCodecH264
           }
       return videoSettings
  }()
exportSession.audioConfiguration.audioOutputSetting = {
      var stereoChannelLayout = AudioChannelLayout()
```

```
memset(&stereoChannelLayout, 0,
MemoryLayout<AudioChannelLayout>.size)
stereoChannelLayout.mChannelLayoutTag =
kAudioChannelLayoutTag_Stereo

 let channelLayoutAsData = Data(bytes: &stereoChannelLayout,
count: MemoryLayout<AudioChannelLayout>.size)
 let compressionAudioSettings: [String: Any] = [
            AVFormatIDKey: kAudioFormatMPEG4AAC,
            AVEncoderBitRateKey: 128000,
            AVSampleRateKey: 44100,
            AVChannelLayoutKey: channelLayoutAsData,
            AVNumberOfChannelsKey: 2]
        return compressionAudioSettings
       }
      ()
}

 fileprivate func updatePickedAsset(_ asset: AVAsset) {
     pickedAsset = asset
 let _imageGenerator = AVAssetImageGenerator(asset: asset)
    imageGenerator.appliesPreferredTrackTransform = true
 let _width = UIScreen.main.bounds.width
imageGenerator.maximumSize = CGSize(width: width, height: width)

imageGenerator.generateCGImagesAsynchronously(forTimes:
[NSValue(time: kCMTimeZero)])
      {
      [weak self] (time,
                   image,
                   actualTime,
                   result,
                   error)
      in guard let strongSelf = self
         else
      {
```

```
        return
}
      if let image = image
        {
          DispatchQueue.main.async
            {
      strongSelf.coverImageView.backgroundColor = UIColor.clear
      strongSelf.coverImageView.image = UIImage(cgImage: image)
                }
            }
           else
        {
              print("load thumb image failed")
            DispatchQueue.main.async
      {
         strongSelf.coverImageView.backgroundColor =
UIColor.red.withAlphaComponent(0.7)
         strongSelf.coverImageView.image = nil
              }
            }
         }
var infoText = "duration: \(String(format: "%.2f",
asset.duration.seconds))"
let size = asset.tracks(withMediaType: .video).first!.naturalSize
          infoText.append("\nresolution: \(size)")

let framerate = asset.tracks(withMediaType:
.video).first!.nominalFrameRate
     infoText.append("\nframerate: \(String(format: "%.2f",
framerate))")
let bitrate = asset.tracks(withMediaType:
.video).first!.estimatedDataRate
      infoText.append("\nbitrate: \(String(format: "%.2f",
bitrate / 1000))kb")
let transform = asset.tracks(withMediaType:
.video).first!.preferredTransform
```

```
let angleDegress = atan2(transform.b, transform.a) * 180 /
CGFloat.pi
       infoText.append("\nangle degress: \(String(format: "%.0f",
angleDegress))")
      fileInfoLabel.text = infoText
          }
      }
extension ViewController: UIImagePickerControllerDelegate,
                          UINavigationControllerDelegate
{
      func imagePickerControllerDidCancel(_ picker:
UIImagePickerController)
{
          picker.dismiss(animated: true,
          completion: nil)
          }

      func imagePickerController(_ picker:
UIImagePickerController,
                 didFinishPickingMediaWithInfo info: [String :
Any])
          {
              picker.dismiss(animated: true,
                             completion: nil)
 if let videoURL = info[UIImagePickerControllerReferenceURL] as?
URL
{
    let asset = AVURLAsset(url: videoURL)
       updatePickedAsset(asset)
      }
 }}
```

HomeViewController.swift

```
 //
```

```
//  HomeViewController.swift
//  ExportSession
//
//  Created by oleksandrazhyrova on 10.04.2023.
//

import Foundation
import UIKit
import AVKit

class HomeViewController: UIViewController
 {

    var avPlayer: AVPlayer?
    var avPlayerItem: AVPlayerItem?
    let avPlayerController = AVPlayerViewController()


     override func viewDidLoad()
     {
              super.viewDidLoad()

// create views
       _createPlayer()
     let tap = UITapGestureRecognizer(target: self, action:
#selector(self.handleTap(_:)))
     avPlayerController.view.addGestureRecognizer(tap)
         }

override func viewDidAppear(_ animated: Bool) {
       super.viewDidAppear(animated)
       avPlayerController.player?.play()
         }

override func viewDidLayoutSubviews()
{
```

```
        super.viewDidLayoutSubviews()
      // set frames
avPlayerController.view.frame = _playerViewFrame()
avPlayerController.view.center = CGPoint(x: view.frame.width/2,
                                         y: view.frame.height/2 + 250)
      }
private func _createPlayer()
{
let filepath: String? = Bundle.main.path(forResource:
"videoplayback", ofType: "mp4")
let fileURL = URL.init(fileURLWithPath: filepath!)
          avPlayerItem = AVPlayerItem(url: fileURL)
          avPlayer = AVPlayer(playerItem: avPlayerItem)
          avPlayerController.player = avPlayer
          avPlayerController.view.frame = _playerViewFrame()

// Turn on video controlls
avPlayerController.showsPlaybackControls = false

self.view.addSubview(avPlayerController.view)
self.addChildViewController(avPlayerController)
      }

@objc func handleTap(_ sender: UITapGestureRecognizer? = nil) {
      if let rate = avPlayerController.player?.rate,
          rate == 0
{
performSegue(withIdentifier: "HomeToConverterSequeIdentifier",
sender: nil)
        }}
private func _playerViewFrame() - > CGRect {return CGRect(x: 0,
                                                          y: 0,
                                                      width: 200,
                                                     height: 200)
          }
      }
```

ResultViewController.swift

```
//
//  ResultViewController.swift
//  ExportSession
//
//  Created by Oleksandra Zhyrova on 11.04.2023.
//

import Foundation
import UIKit
import AVFoundation


class ResultViewController : UIViewController {

     @IBOutlet var label1: UILabel!
     @IBOutlet var label2: UILabel!
     @IBOutlet var imageView: UIImageView!
     @IBAction func backHomeButton(_ sender: Any) {    }
     var avAsset1 : AVAsset?
     var avAsset2 : AVAsset?

     required init?(coder: NSCoder) {
        super.init(coder: coder)
         }

override func viewDidLoad() {
        super.viewDidLoad()
          assert(avAsset1 != nil)
          assert(avAsset2 != nil)
```

```
if let asset1 = self.avAsset1
            {
     let text = _assetDescription(asset: asset1)
     label1?.text = text
}
if let asset2 = self.avAsset2
     {
        label2?.text = _assetDescription(asset: asset2)
     }
}
private func _assetDescription(asset : AVAsset) - > String
     {
        var infoText = ""
     if let videoTrack = asset.tracks(withMediaType:
     .video).first
 {
       infoText.append("\n Video info:")
       infoText.append("duration: \(String(format: "%.2f",
     asset.duration.seconds))")

       let size = videoTrack.naturalSize
       infoText.append("\n resolution: \(size)")
       let framerate = videoTrack.nominalFrameRate
         infoText.append("\n framerate: \(String(format: "%.2f",
     framerate))")

      let minFrameDuration = videoTrack.minFrameDuration.seconds
                infoText.append("\n minFrameDuration:
     \(String(format: "%.2f", minFrameDuration))")

      let bitrate = videoTrack.estimatedDataRate
       infoText.append("\n bitrate: \(String(format: "%.2f",
                                      bitrate /
                                  1024/1024/8))Mb")
```

```
 let formatDescriptions =
videoTrack.formatDescriptions.first     as!
CMFormatDescription
 let subType =
CMFormatDescriptionGetMediaSubType(formatDescriptions)
 let codecString = stringWith(forCC: Int(subType))
       infoText.append("\n codec: \(codecString)")

 let naturalTimeScale = videoTrack.naturalTimeScale
       infoText.append("\n natural time scale:
\(String(format: "%.2f", naturalTimeScale))")
            print("")
        }

  if let audioTrack = asset.tracks(withMediaType:
.audio).first
        {
            infoText.append("\n Audio info")

     let bitrate = audioTrack.estimatedDataRate
            infoText.append("\nbitrate: \(String(format:
"%.2f", bitrate / 1000))kb")

     let naturalTimeScale = audioTrack.naturalTimeScale
     infoText.append("\n natural time scale:
\(Int(naturalTimeScale))")

     let preferredVolume = audioTrack.preferredVolume
     infoText.append("\n preferred volume:
\(preferredVolume)"
     let formatDescriptions =
audioTrack.formatDescriptions.first as! CMFormatDescription
     let subType =
CMFormatDescriptionGetMediaSubType(formatDescriptions)
     let codecString = stringWith(forCC: Int(subType))
     infoText.append("\n codec: \(codecString)")
```

```
          let asbd =
CMAudioFormatDescriptionGetStreamBasicDescription(formatDescript
ions)
          let sampleRate : Int = Int(asbd?.pointee.mSampleRate
     ?? 0)
                infoText.append("\n sampleRate: \(sampleRate)")
        }
        print(infoText)
       return infoText
         }

func stringWith(forCC : Int) - > String
{
     var s: String = String(UnicodeScalar((forCC >> 24) & 255)!)
         s = s + String(UnicodeScalar((forCC >> 16) & 255)!)
         s = s + String(UnicodeScalar((forCC >> 8) & 255)!)
          s = s + String(UnicodeScalar(forCC & 255)!)
             return (s)
         }
     }
```