МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«АУДІОТЕКСТ: РОЗРОБКА ПРОГРАМИ ТРАНСКРИПЦІЇ З РЕДАГУВАННЯМ»**

Виконала: студентка 2 курсу, групи 8.1213-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
О.В. Жирова
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Жировій Олександрі Володимирівні
(прізвище, ім’я та по-батькові)

1. Тема роботи АудіоТекст: Розробка програми транскрипції з редагуванням

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 384-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Комплект нормативних документів.
2. Технічне завдання до роботи.
3. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області.
2. Огляд програмного рішення.
3. Програмний модуль.
4. Економічне обгрунтування проєкту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.03.2024	
2.	Формулювання основної задачі дипломної	15.03.2024	
	роботи та узгодження її з науковим керівником.		
3.	Збір вихідних даних.	19.04.2024	
4.	Аналіз існуючих методів рішення.	22.05.1024	
5.	Обробка методичних та теоретичних	21.06.2024	
	джерел.		
6.	Розробка першого та другого розділу.	26.08.2024	
7.	Реалізація застосунку.	04.10.2024	
8.	Розробка третього розділу.	25.10.2024	
9.	Усунення додаткових проблем та «багів».	08.11.2024	
10.	Розробка четвертого розділу.	15.11.2024	
11.	Оформлення та нормоконтроль	22.11.2024	
	кваліфікаційної роботи магістра.		
12.	Захист кваліфікаційної роботи.	14.12.2024	

Студент _____ О.В. Жирова
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «АудіоТекст: Розробка програми транскрипції з редагуванням»: 130 с., 41 рис., 2 табл., 10 джерел, 1 додаток.

АУДІО, АЛГОРИТМИ, МАКОС, ОБРОБКА, РЕДАГУВАННЯ, РОЗРОБКА, ТЕХНОЛОГІЇ, ТЕКСТ, ТРАНСКРИПЦІЯ.

Об’єктом дослідження – є створення сучасного додатку для транскрипції та розпізнавання аудіо, з акцентом на розробку настільного додатку для операційної системи macOS для полегшення цих процесів.

Мета дослідження – оцінити ефективність і зручність програми, а також розробити рекомендації щодо підвищення точності розпізнавання мовлення та спрощення процесу редагування транскрибованих матеріалів.

Методика дослідження включає детальний огляд сучасних аналогів, таких як Otter.ai, Notta, Speechnotes і AutoMemo. Крім того, проєкт передбачає реалізацію зручного додатку для macOS, розробленого за допомогою Objective-C та Swift, що містить сторонні бібліотеки.

Цей підхід гарантує, що програмний продукт є зручним для користувача, має широкі можливості налаштування та здатний задовольняти сучасні потреби користувачів, відкриваючи нові можливості для автоматизації процесів транскрипції та розпізнавання звуку.

SUMMARY

Master’s qualifying paper “AudioText: Transcription Software Development with Editing”: 130 pages, 41 figures, 2 tables, 10 references, 1 supplement.

ALGORITHMS, AUDIO, DEVELOPMENT, EDITING, MACOS, PROCESSING, TECHNOLOGIES, TEXT, TRANSCRIPT.

The object of the study is the creation of a modern application for audio transcription and recognition, with a focus on developing a desktop application for the macOS operating system to facilitate these processes.

The aim of the study is to assess the application’s efficiency and user-friendliness, as well as to develop recommendations to improve speech recognition accuracy and streamline the editing process for transcribed materials.

The method of research includes a detailed review of modern counterparts such as Otter.ai, Notta, Speechnotes, and AutoMemo. Additionally, the project involves the implementation of a convenient macOS application developed using Objective-C and Swift, incorporating third-party libraries.

This approach ensures the software product is user-friendly, highly customizable, and capable of meeting contemporary user needs while opening new possibilities for automating audio transcription and recognition processes.

ЗМІСТ

Завдання на кваліфікаційну роботу ……..2
Реферат 4
Summary..... 5
Вступ..... 8
Розділ 1 Аналіз предметної області..... 9
1.1 Дослідження актуальності проблеми..... 9
1.2 Актуальність застосунку 10
1.3 Аналогічні додатки 11
1.4 Мета та задачі 14
1.5 Висновки до розділу 15
Розділ 2 Огляд програмного рішення 16
2.1 Операційна система MacOS 16
2.2 Основи програмування 18
2.3 Мова програмування Objective-C і Swift 20
2.4 Візуальна середа розробки 23
2.4.1 CocoaPods..... 25
2.4.2 Storyboard..... 26
2.4.3 Simulator..... 29
2.4.4 Interface Builder 30
2.4.5 Sourcetree..... 31
2.5 Frameworks..... 32
2.5.1 CoreVideo 34
2.5.2 AVFoundation..... 34
2.5.3 Dispatch..... 37
2.5.4 Core Media..... 37
2.5.5 Core Audio..... 38
2.5.6 AppKit..... 38

2.5.7 Accelerate 39
2.5.8 Core Data 39
Розділ 3 Програмний модуль 41
3.1 Архітектура застосування 41
3.1.1 MVC та розподіл відповідальностей 42
3.2 Файлова структура додатку 46
3.3 Організація середовища для розробки додатку 50
3.4 Створення інтерфейсної частини 50
3.5 Види тестування 60
3.6 Тестування додатку 65
3.7 Висновки до третього розділу 65
Розділ 4 Економічне обгрунтування проєкту 66
4.1 Бізнес-план розробки програмного забезпечення 66
4.2 Резюме 66
4.2.1 Маркетинг 67
4.2.2 Нормативно-правові нюанси 67
4.2.3 Складання фінансового бюджету 68
4.2.4 Оцінка можливих ризиків проєкту 68
Висновки 69
Перелік посилань 70
Додаток А Реалізація програми MovieTranscript 71

ВСТУП

На сьогоднішній день робота з комп’ютерами стала невід’ємною частиною нашого життя, а обробка аудіо файлів набуває все більшої актуальності. У мережі існує велика кількість схожих застосунків для транскрибування аудіо, але не всі вони забезпечують достатню точність розпізнавання та зручність редагування матеріалів.

В наш час досить часто виникають ситуації, коли потрібно швидко перетворити відеозапис або аудіозапис на текст, але транскрибування вручну займає багато часу. Програма для MacOS, яка розроблена в цьому проєкті, вирішує цю проблему, поєднуючи точність автоматичного розпізнавання мовлення із зручним інтерфейсом для редагування тексту.

Основні етапи створення додатку включає в себе такі аспекти:

– проєктування та створення архітектури додатку для обробки аудіо файлів;
– інтеграцію алгоритмів розпізнавання мовлення;
– функціональне та модульне тестування застосунку.

При проєктуванні цього проєкту було враховано потреби користувачів у таких функціях, як:

– можливість швидкого редагування тексту після транскрибування;
– підтримка різних форматів аудіо файлів;
– гнучке налаштування параметрів розпізнавання мовлення.

Завдяки простому і зрозумілому інтерфейсу додаток забезпечує ефективну роботу з аудіофайлами і задовольняє сучасні потреби користувачів. Він стане незамінним помічником для журналістів, дослідників, студентів і всіх, хто регулярно працює з аудіо чи відео і потребує швидкого й точного інструменту для транскрибування.

Застосунок не лише спрощує процес обробки інформації, але й підвищує продуктивність, зменшуючи час, необхідний на рутинні завдання.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

У сучасному світі, де комунікація стає ключовою для особистого та професійного життя, програми для транскрипції аудіо в текст набувають особливого значення, особливо в умовах війни.

Транскрибування – це процес перетворення усного мовлення або звукових записів у текстовий формат, який може стосуватися записів інтерв'ю, лекцій, нарад або будь-якого іншого матеріалу. Розробка програми, яка дозволяє легко перетворювати звукові записи на текст з можливістю редагування, є важливим кроком у напрямку полегшення комунікації для багатьох користувачів.

Цей інноваційний додаток не лише спростить процес перетворення мовлення в текст, але й надасть можливість вносити корективи та уточнення, що забезпечує високу якість та точність кінцевого результату. Зручний та інтуїтивно зрозумілий інтерфейс дозволить користувачам з будь-яким рівнем технічної підготовки швидко освоїти програму і використовувати її у повсякденному житті.

Особливо варто підкреслити, що цей додаток може стати життєво важливим інструментом для ветеранів війни, які мають поранення і як наслідок проблеми зі слухом. Багато з них можуть стикатися з труднощами в спілкуванні через втрату слуху внаслідок поранень або стресових ситуацій під час проходження служби. Моя програма допоможе цим людям долати бар'єри у спілкуванні, відновлюючи їхню можливість бути активними учасниками суспільного життя.

Завдяки функціоналу редагування користувачі зможуть адаптувати текст до своїх потреб, що особливо важливо для створення документів, заяв або просто для ведення нотаток. Це сприятиме підвищенню їхньої самостійності та впевненості у спілкуванні.

Таким чином, розробка програми транскрибування аудіо з його подальшим редагуванням стане не лише технологічним досягненням, а й важливим соціальним проєктом, покликаним полегшити життя тих, хто цього найбільше потребує у складні часи війни. Ця ініціатива сприятиме реабілітації та соціальній інтеграції ветеранів, надаючи їм нові можливості для спілкування і вираження своїх думок.

Є мінімум 3 причини для того, чому варто використовувати новий додаток під MacOS для транскрибування аудіо в текст.

По-перше це висока точність розпізнавання мовлення. Додаток використовує алгоритми розпізнавання мовлення, що дозволяє отримувати максимально точні текстові транскрипції. Це знижує потребу в тривалому редагуванні, економлячи часі зусилля користувачів.

По-друге це інтуїтивний інтерфейс для швидкого редагування. Він простий у використанні і додаток дозволяє легко редагувати отриманий текст. Це особливо корисно для журналістів, дослідників та студентів, які працюють з великими обсягами аудіоданих і потребують інструмент, що спрощує обробку матеріалів.

По-третє підтримка різних форматів аудіофайлів і налаштування параметрів. Додаток підтримує широкий спектр аудіоформатів і дозволяє користувачам шукати параметри розпізнавання для досягнення найкращих результатів у конкретних умовах. Це робить його універсальним рішенням для транскрибування як звичайних розмов, так і спеціальних аудіозаписів з унікальними параметрами звуку.

Ці переваги роблять додаток незамінним помічником для тих, кому важлива швидкість, точність та зручність роботи з аудіофайлами.

1.2 Актуальність застосунку

Проблема перетворення аудіо в текст є актуальною для багатьох користувачів сучасних технологій. Зі збільшенням обсягу споживання медіа-

контенту, зокрема під час війни, стає важливою необхідність швидко та точно перетворювати звукові записи на текст.

Транскрибування дозволяє не лише зберегти інформацію, але й робить її більш доступною для для аналізу, редагування та обміну.

Існує безліч рішень для транскрибування, але багато з них мають свої обмеження. Безкоштовні програми часто накладають обмеження на кількість трансформованого контенту або вимагають підписки на платний план для доступу до всіх функцій. Тому важливо мати на ринку універсальне рішення, яке б задовольнило потреби широкої аудиторії без якихось суттєвих обмежень.

Особливо важливою є можливість швидкої адаптації програм до потреб користувачів. Впровадження механізмів машинного навчання та штучного інтелекту може суттєво підвищити точність транскрибування, а також забезпечити автоматичне коригування помилок, що часто виникають при ручному введенні. І це дозволить значно зекономити час користувачів і підвищить загальну якість кінцевого продукту.

Таким чином, розробка програми транскрибування з редагуванням є важливим кроком у розвитку технологій, що підтримують комунікацію всіх верств населення.

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

У 2024 році я можу точно сказати, що цей додаток є актуальним.

1.3 Аналогічні додатки

Даний розділ присвячений аналізу основних аналогів на ринку, щоб визначити їхні ключові можливості та обмеження, а також виділити унікальні переваги запропонованого додатку для macOS. Це дозволить не лише зрозуміти позицію нового продукту в контексті існуючих рішень, а й підкреслити ті інноваційні функції, які роблять його конкурентоспроможним і зручним для користувачів.

Декілька популярних додатків для транскрибування аудіо в текст, які працюють під MacOS, наведено нижче.

Otter.ai – це один із найбільш популярних додатків для автоматичного транскрибування. Він використовує технології штучного інтелекту для перетворення аудіо в текст у реальному часі. Додаток підтримує редагування та має можливість синхронізації з календарем для автмоматичного запису зустрічей (див. рис. 1.1).

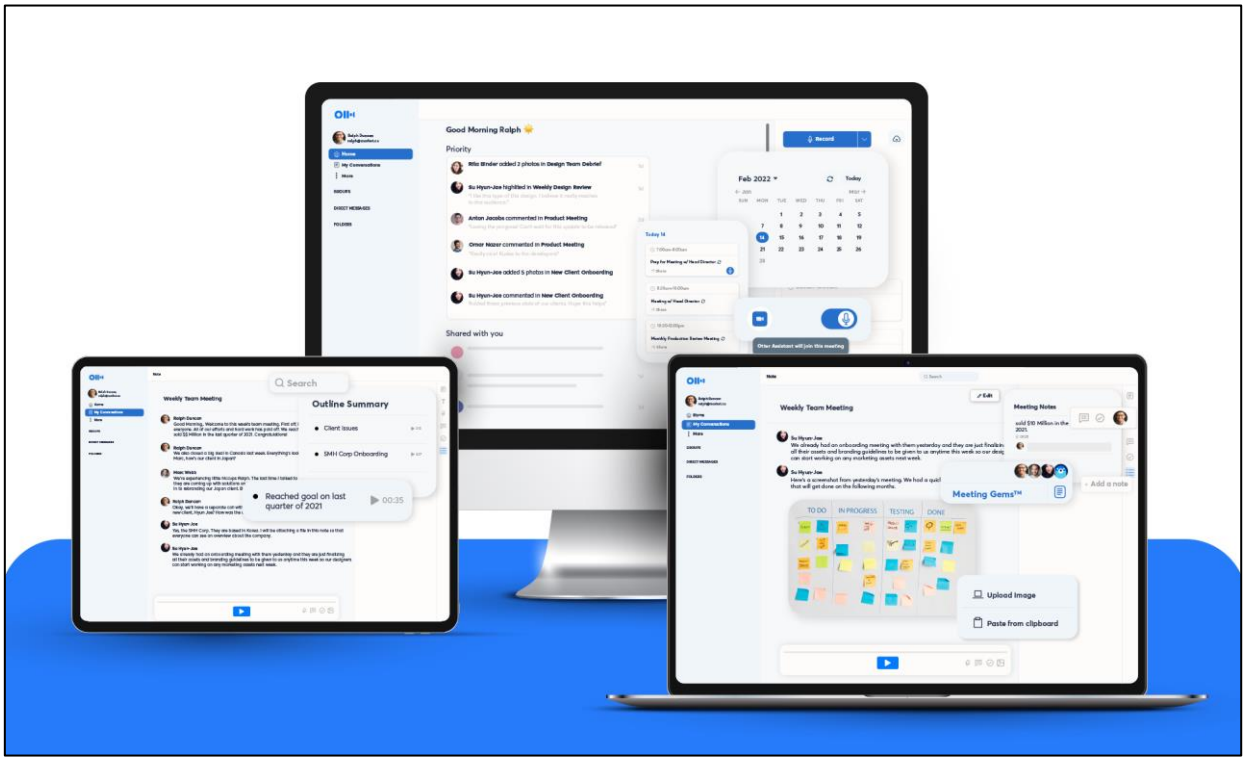


Рисунок 1.1 – Застосунок Otter.ai

Happy Scribe – це програма, яка надає послуги транскрибування як автоматично так і вручну. Користувачі можуть завантажувати аудіо та отримувати точні транскрипти, а також редагувати їх у зручному інтерфейсі. Попри широкий функціонал Happy Scribe, його недоліки можуть бути потрібні в Інтернет-з’єднанні для обробки даних і залежності від платформи. Саме ці аспекти враховано при створенні нової додатки для macOS, що дозволяє зосередитися на автономності та максимальної зручності для користувача (див. рис. 1.2).



Рисунок 1.2 – Додаток Happy Scribe

Express Scribe – це професійний плеєр для транскрибування, який підтримує різні формати аудіо та відео. Додаток дозволяє вручну транскрибувати аудіо, маючи зручні функції управління відтворенням, такими як повільне відтворення. Основною особливістю Express Scribe є орієнтованість на ручне транскрибування, що забезпечує контроль над текстом, проте вимагає більше часу та використання від користувача. На користь від нього пропонується додаток для macOS, який сприяє автоматичному розпізнаванню мовлення з можливістю швидкого редагування, що дозволяє користувачам значно швидше отримувати текст і при цьому зберегти точність і зручність коригування (див. рис. 1.3).



Рисунок 1.3 – Застосунок Express Scribe

1.4 Мета та задачі

Основною метою розробки програми транскрипції з редагуванням – є створення функціонального інструменту, який дозволить користувачам швидко та точно перетворювати аудіо в текст. Додаток має на меті покращити доступ до інформації, забезпечити зручність редагування та адаптацію отриманих текстів під конкретні потреби користувачів.

Задачі цього додатку:

- автоматичне транскрибування: розроблення алгоритму, який забезпечить точне перетворення аудіозаписів у текст з максимальною швидкістю та ефективністю;
- функціонал редагування: надання користувачам можливості редагувати транскрипти, включаючи корекцію помилок, додавання коментарів та зміни структури тексту;
- зручний інтерфейс: розробка користувальницького інтерфейсу, який дозволить легко освоїти додаток незалежно від рівня технічної підготовки користувачів;
- підтримка різних форматів аудіо: забезпечення можливості імпорту аудіо файлів з різних форматів, таких як MP3, WAV, AAC та ін.;
- інтеграція з іншими сервісами: реалізація можливості інтеграції з популярними сервісами, такими як Google Drive, DropBox та ін., для зручного зберігання та обміну транскрипциями;
- персоналізація: надання можливості налаштування додатка відповідно до індивідуальних потреб користувачів, таких як вибір мови, шрифтів, кольорів та інших параметрів;
- інтелектуальні функції для покращення якості транскрипції: додаток може включати функції, як-от адаптацію до індивідуальних голосів, автоматичне виправлення найбільш поширених помилок і пропозицій для стилістичної корекції, що покращує якість кінцевого тексту;

– покращення даних безпеки: забезпечення конфіденційності та безпеки даних користувачів, особливо при інтеграції з хмарними сервісами – включення налаштувань для зберігання транскрипцій лише локально або в хмарному середовищі, відповідно до користувача.

На мою думку, завдяки досягненню цих задач програма транскрипції з редагуванням стане корисним інструментом для широкого кола користувачів, полегшуючи процес спілкування та забезпечуючи більшу доступність інформації.

1.5 Висновки до розділу

Загалом, впровадження інформаційних технологій у сферу транскрибування забезпечує не лише ефективність, але й зручність, що робить цей процес більш доступним для широкого кола користувачів. Розробка нових додатків, що враховують потреби сучасного ринку, є кроком у вдосконаленні комунікаційних технологій та підвищенні загальної якості роботи зі звітом.

РОЗДІЛ 2 ОГЛЯД ПРОГРАМНОГО РІШЕННЯ

2.1 Операційна система MacOS

MacOS – це операційна система, яку розробила компанія Apple спеціально для своїх комп'ютерів Mac (див. рис. 2.1). Вона працює на базі ядра Unix, що гарантує високу надійність, продуктивність і захист даних. MacOS поєднує простий і зручний інтерфейс із розширеними можливостями для професійного використання, що зручно як для звичайних користувачів, так і для фахівців у різних галузях.



Рисунок 2.1 – Продукти компанії Apple

Операційна система, спочатку відома як OS X, вперше з'явилася в 2001 році та відзначила значні зміни за час своєї створення. Кожне оновлення враховувало розвиток апаратного забезпечення Mac і зростання взаємодії між широкими пристроями Apple, включно з iPhone, iPad і Apple Watch. Кожне

оновлення додавало нові можливості, покращуючі роботу системи та користувацький досвід.

У порівнянні з операційною системою Windows MacOS має кілька вагомих переваг. По-перше, MacOS вирізняється своїм підходом до безпеки, оскільки система має суворий контроль над інсталяцією додатків і використанням програмного забезпечення, ризик зараження значно менший, ніж у Windows. Також MacOS надає користувачам інструмент для зберігання даних та захисту інформації, такі як FileVault для шифрування даних та Gatekeeper, який блокує встановлення неперевіреного ПЗ [1, с. 111–114].

MacOS також відзначається стабільною роботою навіть при великому навантаженні завдяки оптимізації пам’яті та процесії. Це особливо важливо для професіоналів, які використовують ресурсоємні програми для обробки графіки, відео чи звуку. У таких задачах MacOS часто показує вищу ефективність у порівнянні з Windows, завдяки чому користувачі можуть розраховувати на стабільну продуктивність протягом усього робочого дня.

Крім того, MacOS забезпечує чудову інтеграцію з екосистемою Apple, зокрема з пристроями iPhone, iPad, Apple Watch та iCloud сервісами. Це дозволяє користувачам зручно синхронізувати дані, миттєво обмінюватися файлами через AirDrop і працювати з іншими гаджетами в єдиній екосистемі. Так, функції Handoff, Universal Clipboard і Continuity дозволяють користувачам легко передавати робочий процес між різними пристроями, що значно підвищує зручність роботи.

Загалом, MacOS є ідеальною платформою для тих, хто шукає надійність, безпеку та зручність у роботі, а також високу продуктивність і довершену інтеграцію з іншими продуктами Apple. Це робить MacOS однією з найкращих операційних систем, особливо для користувачів, які працюють у сфері дизайну, програмування та творчих індустрій, де стабільність та продуктивність є ключовими вимогами.

У таблиці 2.1 наведені порівняльні переваги та недоліки MacOS.

Таблиця 2.1 – Порівняльні переваги та недоліки MacOS

Переваги MacOS	**Недоліки MacOS**
1. *Висока стабільність та надійність*: система, оптимізована під апаратне забезпечення Apple, забезпечує стабільну роботу навіть під великим навантаженням.	1. *Обмеженість ПЗ*: деякі спеціалізовані програми доступні лише для Windows і MacOS не має такої ж широкої підтримки ігор.
2. *Підвищена безпека*: менша схильність від вірусів і шкідливого ПЗ завдяки надійному захисту та обмежені можливості встановлення неперевірених додатків.	2. *Вища вартість пристроїв Apple*: комп'ютери Mac коштують дорожче ніж більшість пк Windows, що робить MacOS менш доступним.
3. *Інтуїтивний інтерфейс*: MacOS відома своїм зручним та простим інтерфейсом, що знижує навантаження на користувачів і спрощує робочий процес.	3. *Обмежені можливості налаштування*: MacOS менш гнучка в налаштуванні системних параметрів і апаратного забеспечення в порівнянні з Windows.
4. *Глибока інтеграція*: MacOS дозволяє легко об'єднувати всі пристрої Apple в єдину екосистему, що підвищує зручність роботи.	4. *Складність сумісності з іншими ОС та апаратним забезпеченням*: проблеми з підтримкою деяких периферійних пристроїв і обмежена інтеграція з пристроями, які не належать до екосистеми Apple.
5. *Висока продуктивність*: оптимізація для професійних ПЗ (Adobe, Final Cut Pro, Logic Pro) робить MacOS ідеальною для творчих професій.	5. *Складність в обслуговуванні та ремонті*: більшість пристроїв Apple є складними для самостійного ремонту та обслуговування, а деталі для них коштують дорожче.

2.2 Основи програмування

Для створення власних додатків для операційної системи MacOS потрібні деякі інструменти та навички, розглянемо їх.

Xcode. Інтегроване середовище розробки (IDE) для MacOS, яке включає редактор коду, інструменти компіляції, відлагодження, профілювання, а також Interface Builder для створення інтерфейсів. Xcode також містить симулятор для тестування додатків на різних версіях. Xcode підтримує мови програмування, як Swift, Objective-C, C, та C++, і надає потужні інструменти

для побудови сучасних додатків, такі як інструменти для управління версіями, аналізу коду, оптимізації продуктивності й роботи з інтерфейсами користувача.

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 MacOS складається з чотирьох основних рівнів абстракції (див. рис. 2.2), які забезпечують потужний інструментарій для розробників.

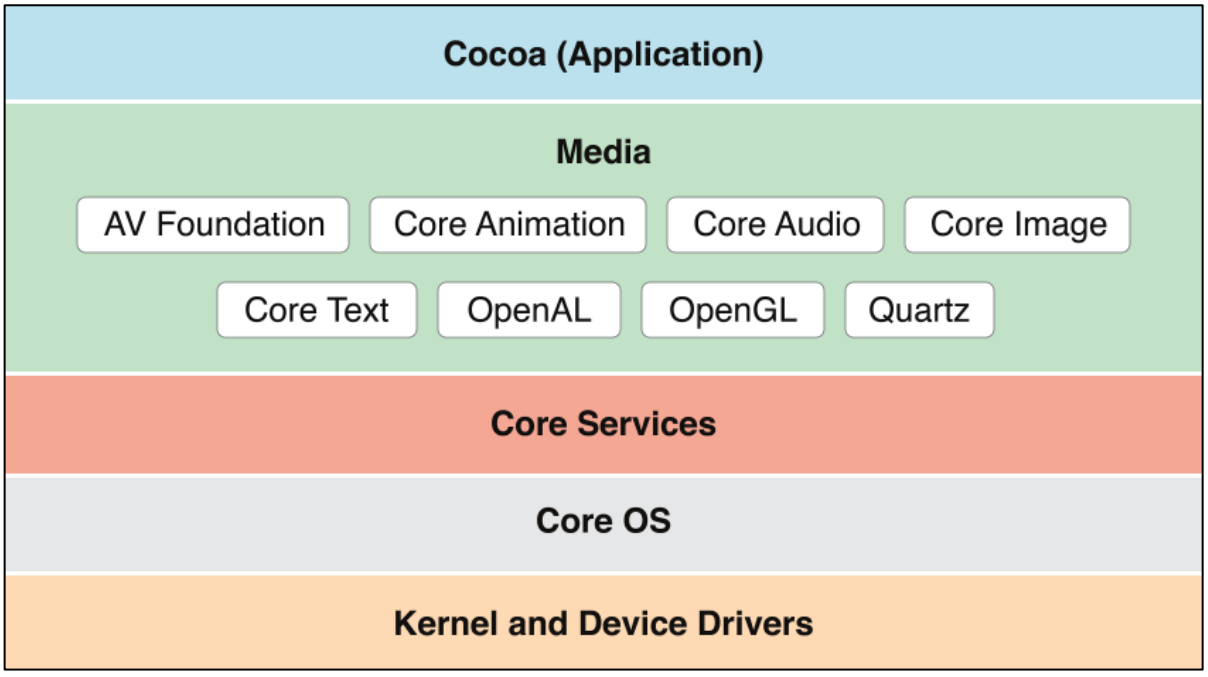


Рисунок 2.2 – Cocoa в архітектурі MacOS

Перший рівень. Cocoa – це основний рівень для MacOS-додатків, які містить основні рамки для інтерфейсу користувача, обробки подій, управління ресурсами та іншої базової функціональності. Cocoa включає фреймворки AppKit, Foundation, які є ключовими для розробки додатків [2, с. 91–98].

Другий рівень. Мультимедіа – цей рівень включає інструменти для роботи з аудіо, відео, графікою та анімацією, що дозволяє створювати мультимедійні елементи, редагувати зображення та відео, а також інтегрувати функціонал.

Третій рівень. Основні сервіси – це рівень, який забезпечує основні функціональні можливості як-от управління базами даних, мережеві сервіси, обробка тексту, геолокація, підтримка iCloud та багато іншого. Ці фреймворки, на які спирається більшість програм для доступу до функцій системи.

Четвертий рівень. CoreOS – базовий рівень ОС, що забезпечує низькорівневий доступ до системних функцій, таких як безпека, управління процесами, підтримка апаратного забезпечення та мережі.

Apple рекомендує розробникам використовувати високорівневі фреймворки, щоб забезпечити більшу стабільність та сумісність із системою. Фреймворки низького рівня слід використовувати лише при необхідності, коли потрібно отримати доступ до специфічного функціоналу.

Для створення успішного MacOS додатка важливо враховувати дизайн інтерфейсу користувача (UI) та досвід користувача (UX) на кожному етапі. Інтерфейс повинен бути зручним, з чіткою навігацією, відповідати стандартам MacOS, забезпечувати легке читання читання тексту та доступність функцій. Інтерактивні елементи мають бути інтуїтивно зрозумілими, а завдання користувача виконуватись з мінімальними зусиллями.

2.3 Мова програмування Objective-C і Swift

Objective-C – об’єктно-орієнтована мова програмування, розроблена компанією Apple на основі мови C і парадигми Smalltalk. На відміну від C++,

Objective-C повністю поєднується з C, а код C компілюється. Об’ктна 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. Мову 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. Тобто це означає, що виклики методів інтерпретуються як надсилання повідомлень з іменами та аргументами до об’єкта, подібно до того, як це працює у 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 (див. рис. 2.3). 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.



Рисунок 2.3 – Мови програмування Swift (наступник Objective-C)

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.

У таблиці 2.2 наведено основні відмінності між мовами програмування Swift та Objective-C, включаючи синтаксис, продуктивність, безпеку, простоту

використання, а також підтримку сучасних функцій, які впливають на розробку додатків для екосистеми Apple.

Таблиця 2.2 – Порівняння Swift та Objective-C

Feature	Swift	Objective-C
Syntax	Clean, modern, concise	Verbose, with a unique messaging syntax
Performance	Optimized for speed	Slower due to dynamic runtime
Memory Management	Automatic with ARC	Limited ARC features with Manual retain/release
Tooling	Strong support in Xcode, Playgrounds	Mature, but less modern features
Community	Growing rapidly	Established, vast resources
Execution Time	Generally faster	Slower due to runtime overhead
Safety	Strong type system, optionals	More prone to runtime errors
Learning Curve	Easier for beginners	Steeper due to complex syntax

2.4 Візуальна середа 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.

Основні вікна робочої області в Xcode (рис. 2.4) та розташовані зліва в напрямку годинникової стрілки:

– область навігації;

– область редактору;

– область інструментів;

– область (debug).

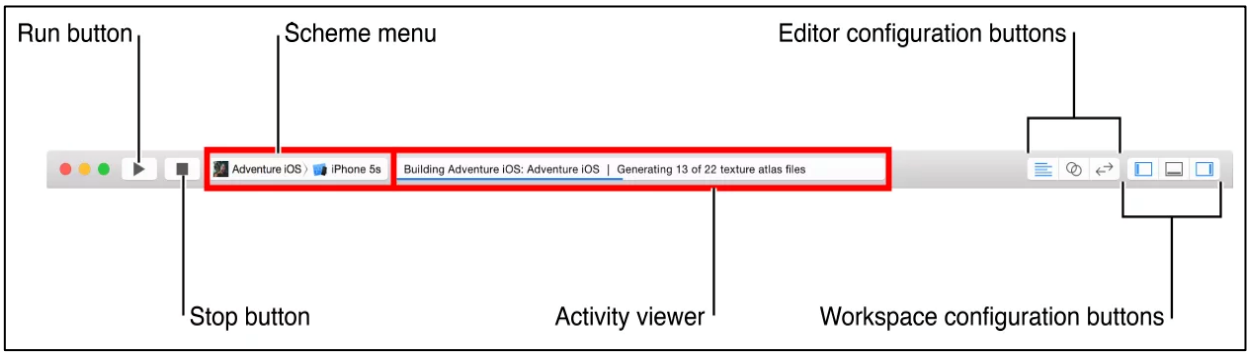


Рисунок 2.4 – Вікно робочої області Xcode

У Xcode, зокрема, є редактор коду, компілятор, відладчик, конструктор інтерфейсів (див. рис. 2.5) та симулятор (будь-який технічно є окремою програмою, запущеною з Xcode).

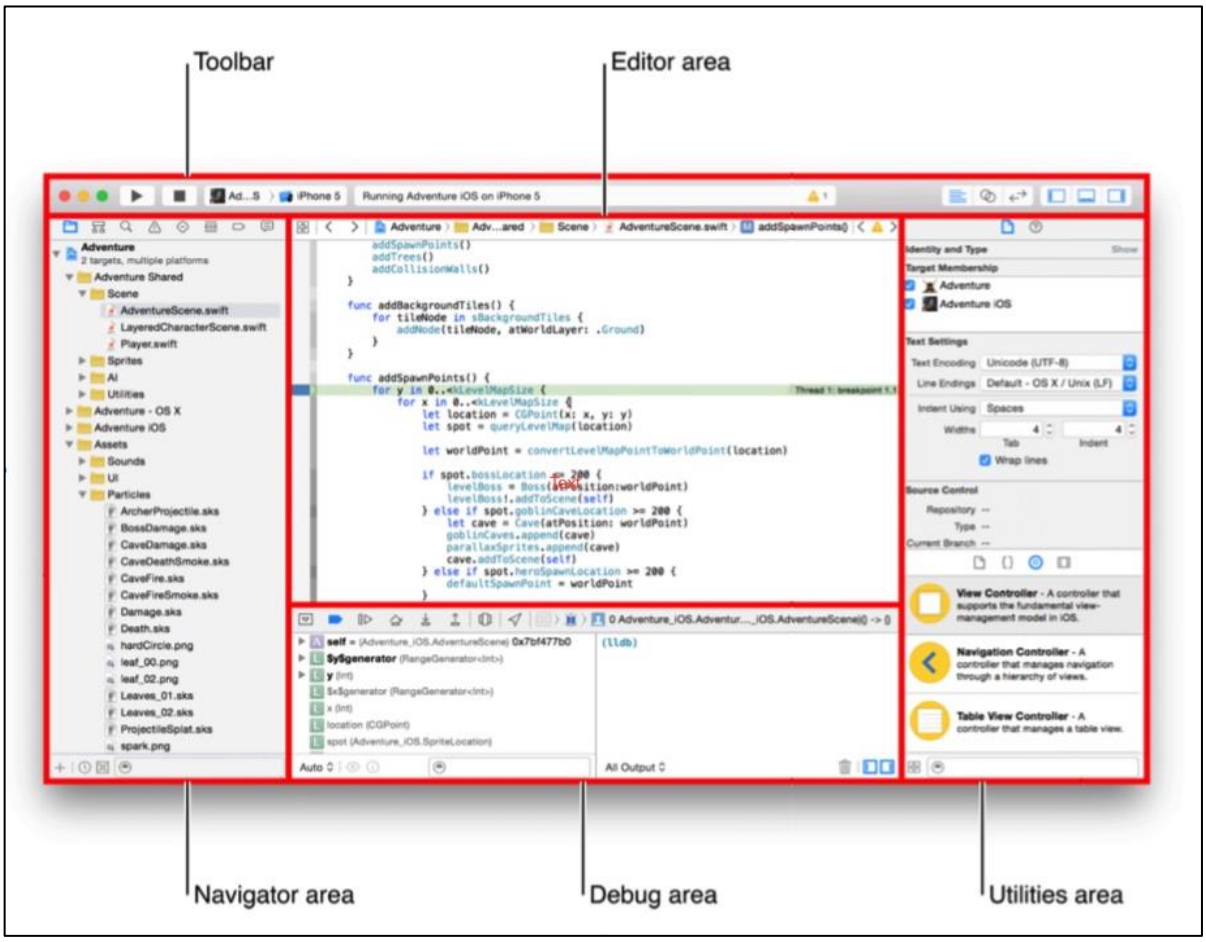


Рисунок 2.5 – Вікно робочої області детально у Xcode

2.4.1 CocoaPods

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

При створенні додатку не доцільно писати кожну частину коду самостійно. Виходячи з цього, попередній вихідний код інших розробників надає в корінь проєкту. 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 проєкт потрібно зберегти в репозиторії без зайвих файлів. бібліотек, і той, хто завантажує проєкт для роботи з ним, повинен самостійно додати ці файли до проєкту.

Управління залежностями за допомогою CocoaPods (рис. 2.6) є корисним інструментом для розробників macOS, після чого він допоможе вирішити проблеми, пов'язані з управлінням залежностями. CocoaPods дозволяє контролювати взаємодію між іншими бібліотеками, які використовують у проєкті, забезпечує структурованість проєкту та забезпечує підтримку проєкту в належному вигляді [7, с. 60–65].

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

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.

Після налаштування залежностей буде створено новий шаблон проєкту. З цього моменту проєкт слід відкрити через згенерований файл із

розширенням <workspace>. Цей файл містить усю повноцінну залежність або “pods”, тому єдиний правильний спосіб відкритого проєкту – через нього.

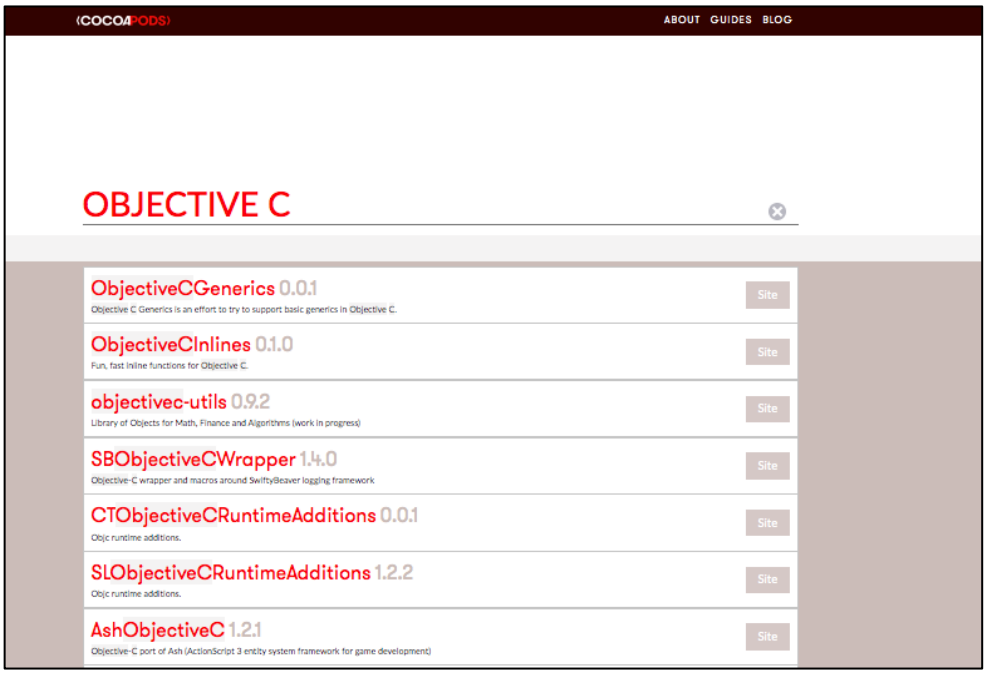


Рисунок 2.6 – Бібліотеки на сайті cocoapods.org

Це налаштування покращує підтримку проєкту шляхом централізації керування залежностями та спрощення оновлень. Однак важливо пам’ятати, що якщо в Podfile вносяться будь-які оновлення (наприклад, додавання або видалення бібліотек), потрібно знову запустити інсталяцію pod, щоб відповідним чином оновити файл “.xcworkspace”.

2.4.2 Storyboard

Storyboard спочатку було розроблено для iOS, але в macOS також є можливість використовувати його для створення інтерфейсів. З допомогою нього можна об’єднати кілька його вікон або представити в одному файлі, знайти зв’язки між ними та створити переходи (segue) між абсолютними контролерами. Це зручно для візуалізації структури доповнення та спрощує процес створення інтерфейсу, особливо коли він включає багато елементів.

Раніше в macOS для цих цілей використовуються окремі файли XIB для кожного контролера або компонента інтерфейсу.

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 – 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.

У MacOS також існує декілька типів переходів між вікнами або представленнями (view controllers), які дозволяють створити плавний та інтуїтивний інтерфейс (рис. 2.7).

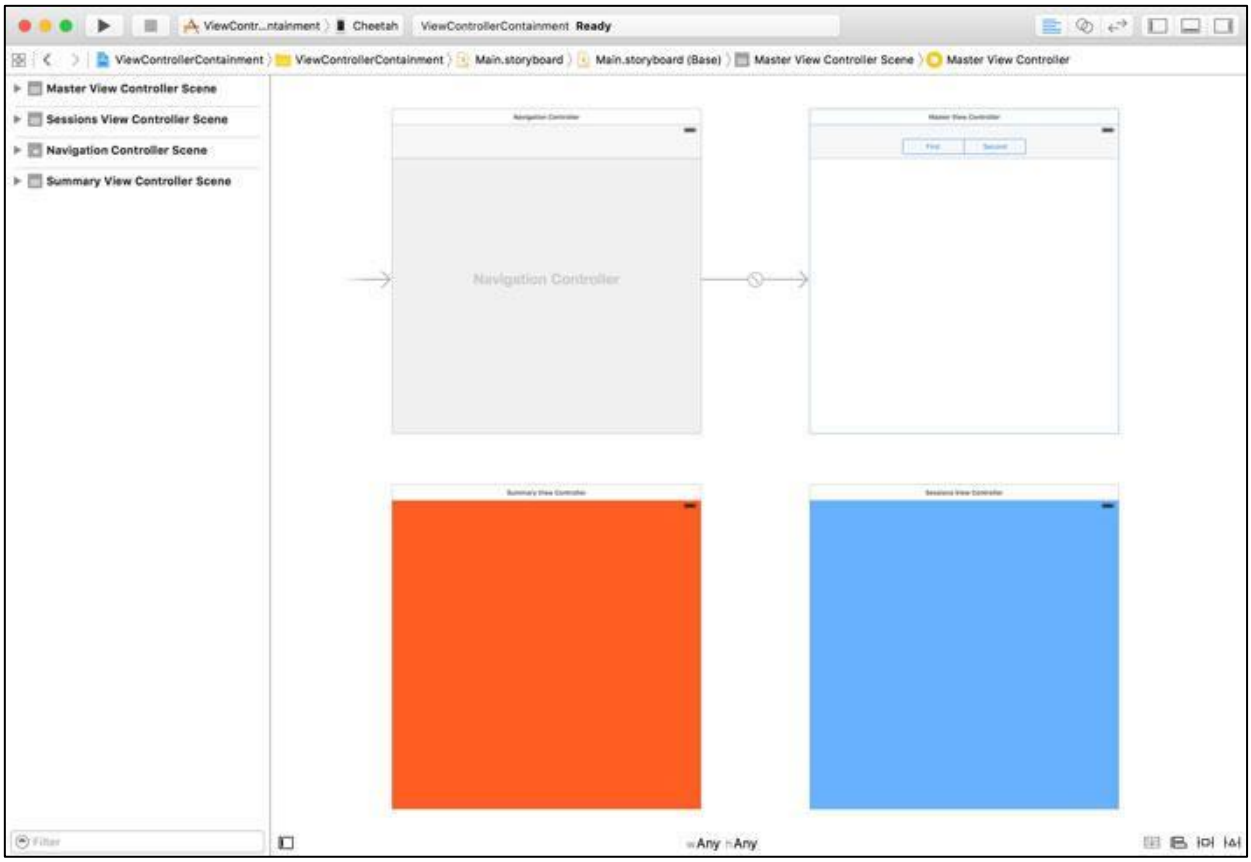


Рисунок 2.7 – Відображення контролерів

Однак, кожен тип переходу може підходити для певних ситуацій і бути менш ефективним у інших. Вибір переходу залежить від структури інтерфейсу та бажаного ефекту. Деякі переходи можуть працювати лише у визначених сценаріях. Розглянемо деякі з переходів, які часто використовуються у MacOS.

Перехід push. Використовуються для навігації в обидва боки стеку контролерів. Цей перехід дозволяє додавати новий контролер на вершину стека або, навпаки, прибирати його, щоб повернутися до попереднього. Однак у MacOS цей тип переходу частіше використовується в межах навігаційного контроллера в обмежених випадках.

Present as Modal Window. Один із найпоширеніших типів переходів у MacOS. Використовуючи цей перехід, можна показати новий контролер поверх інших, але без включення його в ієрархію або стек екранів. Контролер відображається у новому вікні, що надає йому окремий рівень взаємодії.

Перехід типу Popover. Відображає новий контролер у вигляді спливаючого вікна, яке зазвичай має обмежений розмір. Цей вид переходу є особливо корисним на великах екранах, наприклад, у разі надання швидкого доступу до додаткової інформації чи налаштувань, і добре інтегрується з контекстними меню або панелями інструментів.

Show as Sheet. Цей тип переходу показує новий контролер у вигляді листка, що закріплюється зверху на вікні попереднього контролера. Листкові переходи зазвичай використовуються для надання опцій або збору інформації від користувача, як наприклад, у випадку форм або діалогових вікон підтвердження.

Show Detail. Цей тип переходу добре підходить для роботи з поділеними контролерами (Split View Controllers). Наприклад, він може бути використаний для відображення деталей вибраного елемента в основному контролері.

Кожен із цих переходів має свої переваги та обмеження і може покращити користувацький досвід, якщо він обраний відповідно до дизайну програми та потреб користувача.

2.4.3 Simulator

Симулятор, який входить до складу інструментів Xcode, дозволяє швидко створювати прототипи й тестувати збірки вашої додатки під час розробки. Він імітує мобільний пристрій, забезпечуючи середовище для запуску програм, як на реальних пристроях. Симулятор є цінним інструментом для попереднього тестування і його доцільно використовувати перед перевіркою додатка на фізичному пристрої (рис. 2.8).

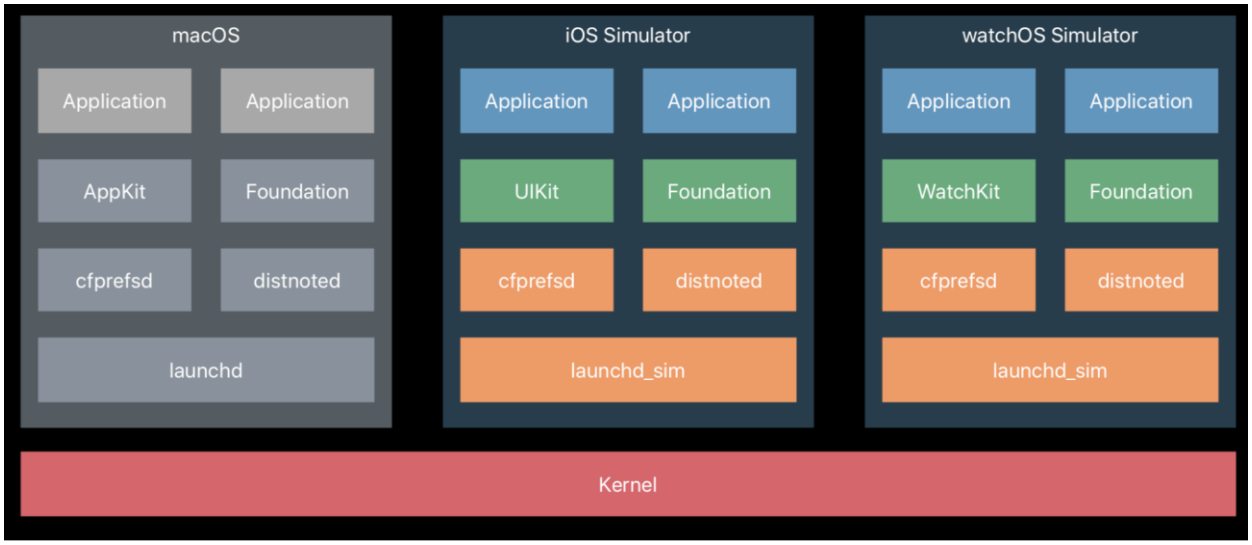


Рисунок 2.8 – The structure of the simulator

Симулятор дає змогу відтворювати пристрої з розширеними розмірами екранів та версіями операційної системи. Кожне змодельоване середовище, що складається з певної моделі пристрою та версії ПЗ, є незалежним, зі своїми унікальними налаштуваннями й файлами, доступними лише на даному пристрої в симуляторі.

Завдяки Simulator можна вирішувати основні проблеми вашої задачі ще на етапі розробки та початкового тестування, а також одним інструментом розробника, доступним тільки в середовищі моделювання. Втім, хоча симулятор є зручним для нього засобом для тестування, він не забезпечує повного відображення роботи програми і не має бути єдиним методом перевірки. Якщо симулятор працює на комп’ютері, він використовує його

ресурси – такі як процесор, пам’ять і мережеве підключення, що може суттєво відрізнятися від реальних умов на мобільному пристрої.

Тому важливо перевіряти продуктивність і користувальницький інтерфейс, оскільки в симуляторі вони можуть виглядати краще, ніж на реальному девайсі. Для цього можна скористатися одним процесом «Sideloading» – підключивши пристрій до комп’ютера й вибравши його як цільовий для тестування у вікні вибору пристрою.

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 (рис. 2.9) користувальницького інтерфейсу містять такі елементи як:

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

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.

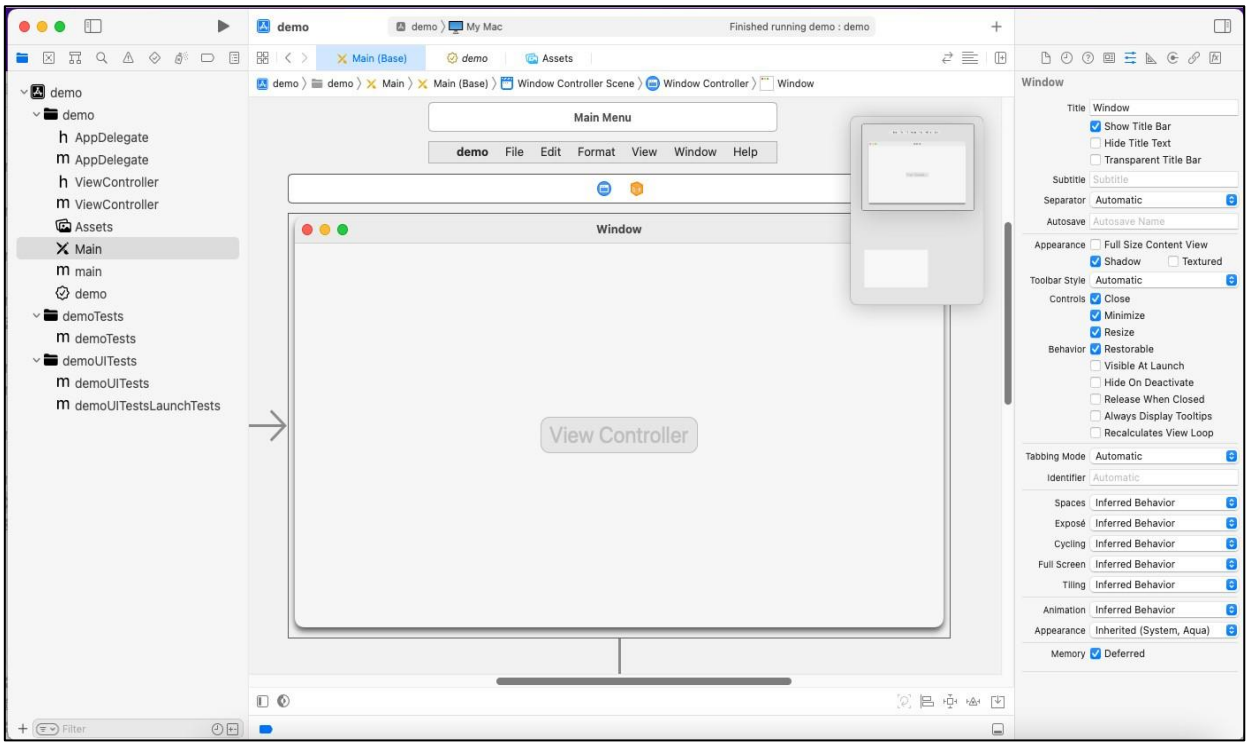


Рисунок 2.9 – Interface Builder in Xcode

2.4.5 Sourcetree

Sourcetree – це потужний і, що важливо, абсолютно безкоштовний інструмент для керування проєктами, які розміщені в GitHub-репозиторіях. Він стане в пригоді тим, хто одночасно веде кілька розробок і прагне швидко завантажити зміни в Git. Програма запускає потреби працювати з командним рядком, пропонуючи зручний інтерфейс з підтримкою окремих вкладок для кожного проєкту. Sourcetree – чудовий помічник для команд розробників та окремих фахівців, які спільно працюють над одним проєктом (рис. 2.10).

Завдяки Sourcetree для macOS можна легко створювати, переглядати та змінювати проєкти Git, а також швидко публікувати нові розробки у власному профілі GitHub. Більшість дій у програмі створюється через зручну панель інструментів, де доступні кнопки для відкриття репозиторіїв, застосування патчів, переміщення та клонування окремих елементів проєкту, перегляду останніх змін, виконання коду, додавання тегів, запуску «ребейза» тощо.

Sourcetree інтуїтивно візуалізує як локальні, так і віддалені репозиторії, а також дає можливість швидко підключати гілки коду. Просунуті користувачі завжди можуть зменшити термінал, просто натиснувши одну кнопку.

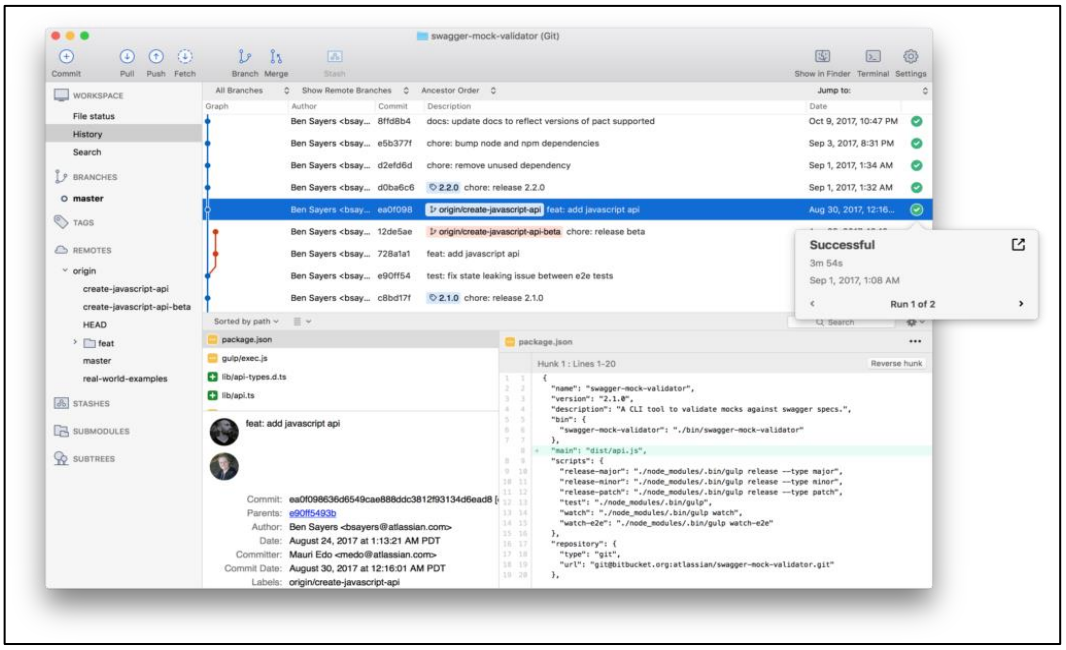


Рисунок 2.10 – Програма Sourcetree

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

У macOS також є два типи фреймворків: статичні та динамічні.

Статичні фреймворки – це вже скомпільовані бібліотеки, які приєднуються до додаток на етапі збірки. Це означає, що код фреймворка, вбудований у бінарний файл програми, і його неможливо змінити під час виконання. Основною перевагою статичних фреймворків є те, що

компоновщик часто може виконати оптимізацію та видалити невикористаний код. Цю функцію можна ввімкнути в Xcode, щоб аналізувати використані символи та зменшити розмір програми.

Динамічні фреймворки в macOS працюють трохи інакше. Це скомпільовані бібліотеки, які завантажуються під час виконання. Це означає, що код фреймворка не входить до бінарного файлу програми [6], а зберігається окремо, що дозволяє оновлювати або змінювати його під час роботи програми. Динамічні фреймворки можуть збільшити розмір бандла програми, після чого вони зберігаються в її структурі (рис. 2.11). Однак вони не можуть використовувати один і той самий динамічний фреймворк із додатковими компонентами в межах однієї додатки, наприклад, основним бінарним файлом та додатковими розширеннями.

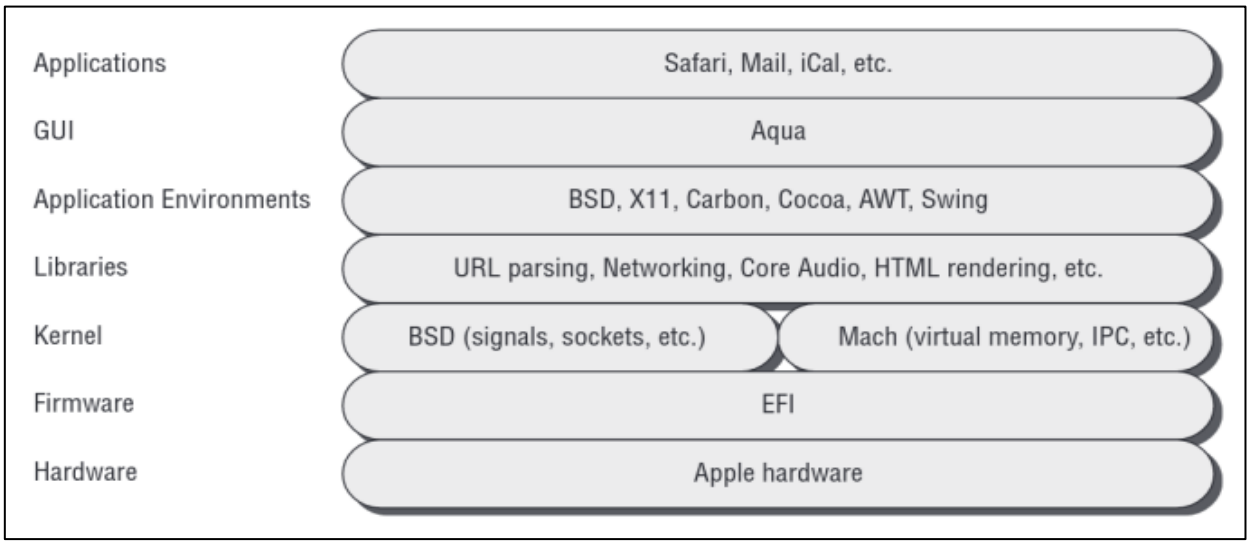


Рисунок 2.11 – Базова архітектура macOS

Таким чином, у macOS статичні та динамічні фреймворки мають подібні переваги та обмеження, як і в iOS. Статичні фреймворки можуть забезпечити менший час запуску, але збільшують розмір програми, тоді як динамічні дають більше гнучкості для оновлення та повторного використання, хоча і трохи впливають на продуктивність.

Вибір між статичними та динамічними фреймворками в macOS залежить від спеціально потрібних програм та цілих розробок.

2.5.1 CoreVideo

Core Video – це технологія обробки відео, яка використовується в macOS. Вона забезпечує з’єднання процесу декомпресії кадрів відео з іншими технологіями рендерингу та композиції, зокрема Quartz для обробки та відображення зображень.

Core Video надає модель буферизації та синхронізації для відтворення в рамках свого конвеєра обробки відео. Відео Джерело надає розпакований потік даних, які можуть бути перетворені на зображення та відображені в контексті Quartz 2D. Core Video служить посередником між цим джерелом і середовищем відтворення. Отримані зображення можуть додатково оброблятися через Core Image, перш ніж формувати фінальну сцену за допомогою Quartz Compositor. Для забезпечення плавності відтворення Core Video підтримує кадровий буфер, де зберігаються попередньо оброблені та скомпоновані кадри. Для підтримки буфера використовується високо пріоритетний потік, який працює незалежно від програми, що ініціює відтворення, і компенсує різні частоти оновлення дисплея та затримки.

Core Video використовує графічний процесор (GPU) для рендеринга та композиції, забезпеченого Quartz GL, Core Image та Quartz Extreme, а остаточне відтворення відбувається на поверхні OpenGL.

Конвеєрна модель Core Video спрощує роботу з відео, розділяючи процес на окремі етапи. Це дозволяє розробникам легко отримувати доступ до кадрів і керувати ними без потреби в конвертації даних чи синхронізації дисплея. Програми, які не вимагають маніпуляцій з окремими кадрами, можуть не використовувати Core Video відразу.

2.5.2 AVFoundation

AVFoundation – це фреймворк для роботи з аудіо- та відео контентом на macOS, iOS, watchOS та tvOS. Завдяки AVFoundation можна відтворювати,

створювати та редагувати файли QuickTime та MPEG-4, транслювати потоки HLS, а також вбудовувати потужний мультимедійний функціонал у свої додатки.

AVFoundation надає можливості великих наборів, які охоплюють створення, редагування та відтворення фото й відео, що його особливо зручно робити з камерами та медіафайлами (див. рис. 2.12). Фреймворк macOS часто використовується для запису й обробки відео та аудіо, забезпечуючи розробникам високий рівень контролю над медіаконтентом у їхніх додатках [8].

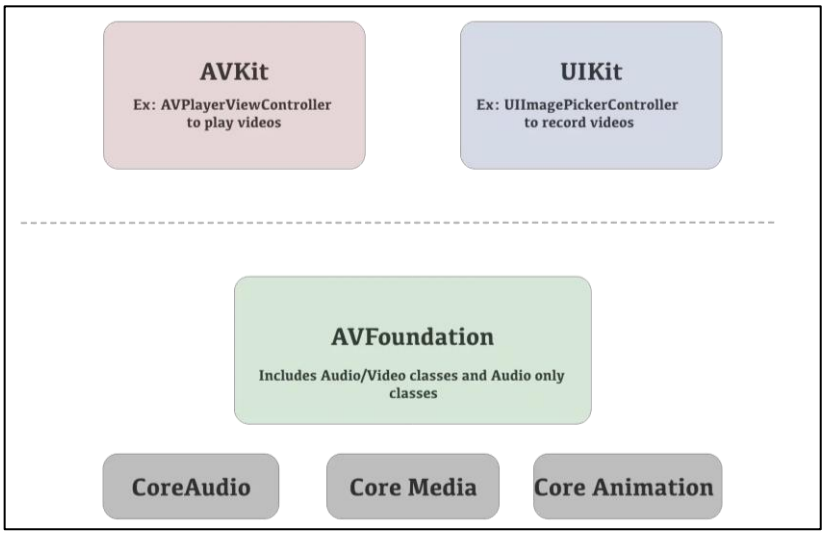


Рисунок 2.12 – AVFoundation framewotk

AVFoundation в macOS забезпечує все необхідне для створення додатка для транскрибування з можливістю редагування аудіо. Цей фреймворк надає інструменти для обробки аудіо контенту, які слід записувати, відтворювати та зберігати аудіо, а також отримувати доступ до даних для подальшої обробки та транскрибування.

Розглянемо **основні етапи створення програми з AVFoundation** для транскрибування та редагування аудіо.

Запис аудіо. Використання `AVAudioRecorder`, щоб налаштувати запис аудіо. Цей клас дозволяє налаштовувати параметри якості звуку, кодування, збереження аудіо файлів та контролювати процес запису.

Завантаження та відтворення аудіо. Для відтворення аудіо файлів потрібно використовувати `AVAudioPlayer` або `AVPlayer`. Це дозволяє керувати аудіо потоком, наприклад, починати, перемотувати, зупиняти, змінювати швидкість відтворення та ін., що корисно при транскрибуванні.

Отримання даних для транскрибування. Після запису або завантаження аудіо файлу можна отримати його буферні дані за допомогою `AVAudioPCMBuffer` і `AVAudioFile`. Крім того, для обробки аудіо і його перетворення в текст можна використовувати технології машинного навчання, такі як Speech Recognition API від Apple, або інтегрувати сторонні сервіси транскрибування, якщо потрібно точніше або швидше розпізнавання.

Редагування аудіо. AVFoundation дозволяє програмно обрізати, з'єднувати та змінювати аудіодані. За допомогою `AVMutableComposition` можна створити аудіо композицію з кількох файлів, виконати їх обрізку та з'єднання. Також можна додати ефекти за допомогою `AVAudioEngine`, які дозволяють підключати різні аудіоефекти, такі як реверберація, еквалізація та інші.

Зберігання та експорт аудіо. Збереження відредагованих аудіофайлів можна виконати, використовуючи `AVAssetExportSession`. Це дозволяє експортувати фінальні аудіофайли у різних форматах, наприклад, .m4a, .wav тощо.

Інтеграція з інтерфейсом користувача (UI). MacOS пропонує інструменти для створення зручного інтерфейсу для роботи з аудіо. Можна додати елементи управління, такі як таймлайн для аудіо, кнопки відтворення та зупинки, а також індикатори для відображення прогресу транскрипції.

Ця архітектура допоможе ефективно обробляти, редагувати та транскрибувати аудіофайли в додатку на macOS, використовуючи можливості AVFoundation.

2.5.3 Dispatch

Dispatch (або Grand Central Dispatch, GCD) – це фреймворк, який надає потужний та простий у використанні механізм керування багатопоточністю в macOS та iOS. З допомогою його розробники можуть виконувати операції асинхронно, використовуючи черги завдань (черги), що дозволяє оптимізувати продуктивність додатка, уникати блокування основного потоку та забезпечити швидкий перегляд використовуваного інтерфейсу [3, с. 29–44].

У Dispatch всі завдання виконуються в чергах. Черга – це послідовність операцій, що виконуються у визначеному порядку.

Існує два основні типи черг:

– серійні (serial) черги виконують завдання по одному, зберігаючи порядок виконання – це корисно для роботи з ресурсами, до яких потрібно отримувати доступ по черзі, уникати конфліктів або змін, наприклад, запис у файл;
– паралельні (concurrent) черги дозволяють виконувати кілька завдань одночасно – це підвищує ефективність і дозволяє швидше виконувати завдання, які не залежать один від одного.

Grand Central Dispatch є ефективним рішенням для побудови продуктивних та плавних додатків у macOS, спрощуючи керування асинхронними операціями і забезпечуючи гнучкість у плануванні виконання завдань.

2.5.4 Core Media

Фреймворк Core Media визначає медіа-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.

2.5.6 AppKit

AppKit – це фреймворк для створення графічного інтерфейсу користувача (GUI) в macOS (рис 2.13). Він надає інструменти та компоненти для розробки візуально насичених та функціональних додатків, що включають вікна, панелі, кнопки, текстові поля, таблиці, малювання графіки, обробку подій та багато іншого.



Рисунок 2.13 – Framework AppKit

AppKit є основною бібліотекою для побудови додатків на macOS і є еквівалентом UIKit на iOS.

AppKit залишається основою розробки додатків на macOS і забезпечує розробникам усі необхідні засоби для створення професійного інтерфейсу з нативним виглядом.

2.5.7 Accelerate

Accelerate – це потужний фреймворк для macOS та iOS, розроблений для високопродуктивних обчислень. Він оптимізує виконання математичних, фізичних та графічних обчислень на центральному та графічному процесорах (CPU і GPU) Apple, забезпечуючи швидкий і ефективний доступ до таких обчислень як лінійна алгебра, обробка сигналів, векторні та матричні операції, обробка зображень та статистичні функції.

Accelerate часто використовується для створення наукових додатків, машинного навчання, графічної візуалізації та інших додатків, де потрібна висока продуктивність.

2.5.8 Core Data

Core Data – це потужний фреймворк від Apple для управління модельними даними в додатках iOS, macOS, watchOS і tvOS. Він дозволяє розробникам працювати з даними високого рівня, не заглиблюючись у деталі збереження, пошуку та управління інформацією. Основне призначення Core Data – зберігання об’єктів у базі даних, їх витягування, синхронізація і управління ними [10, с. 408; с. 411; с. 437].

Core Data не є простою заміною для `UserDefaults` або файлового збереження. Це інструмент для складних завдань управління даними, який забезпечує високу продуктивність і зручність (рис. 2.14).

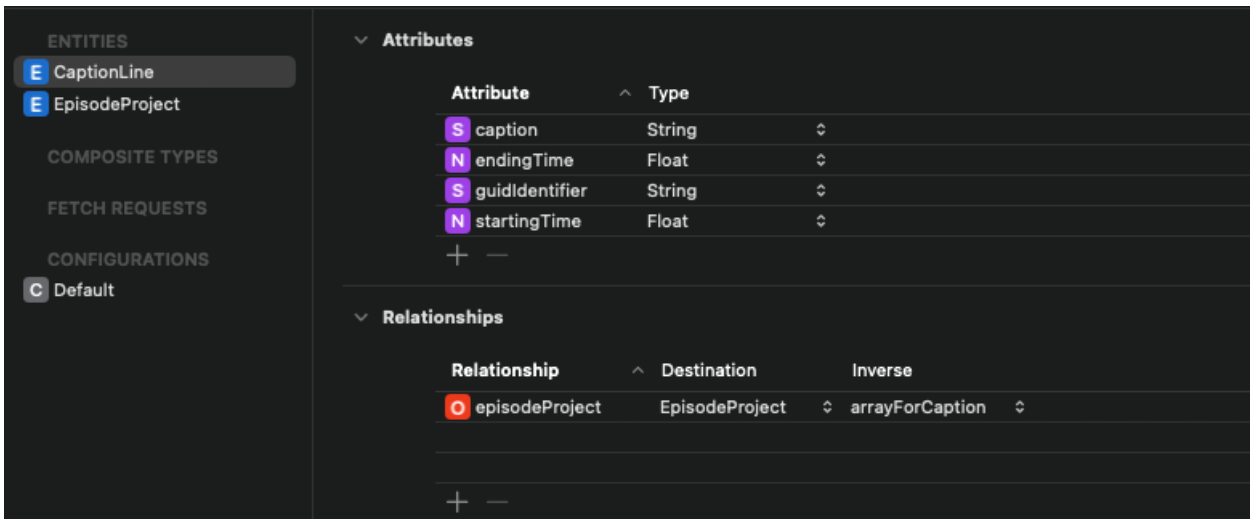


Рисунок 2.14 – Core Data у додатку

РОЗДІЛ 3 ПРОГРАМНИЙ МОДУЛЬ

3.1 Архітектура застосування

Архітектура застосування – це структура, яка визначає організацію та компоненти програмного забезпечення, їх взаємодію, а також принципи проєктування та реалізації. Вона описує, як різні модулі, сервіси або шари системи співпрацюють для досягнення цілей застосунку.

Побудова ефективної архітектури враховує такі аспекти, як продуктивність масштабованість, підтримка, безпека і економічність (див. рис. 3.1).

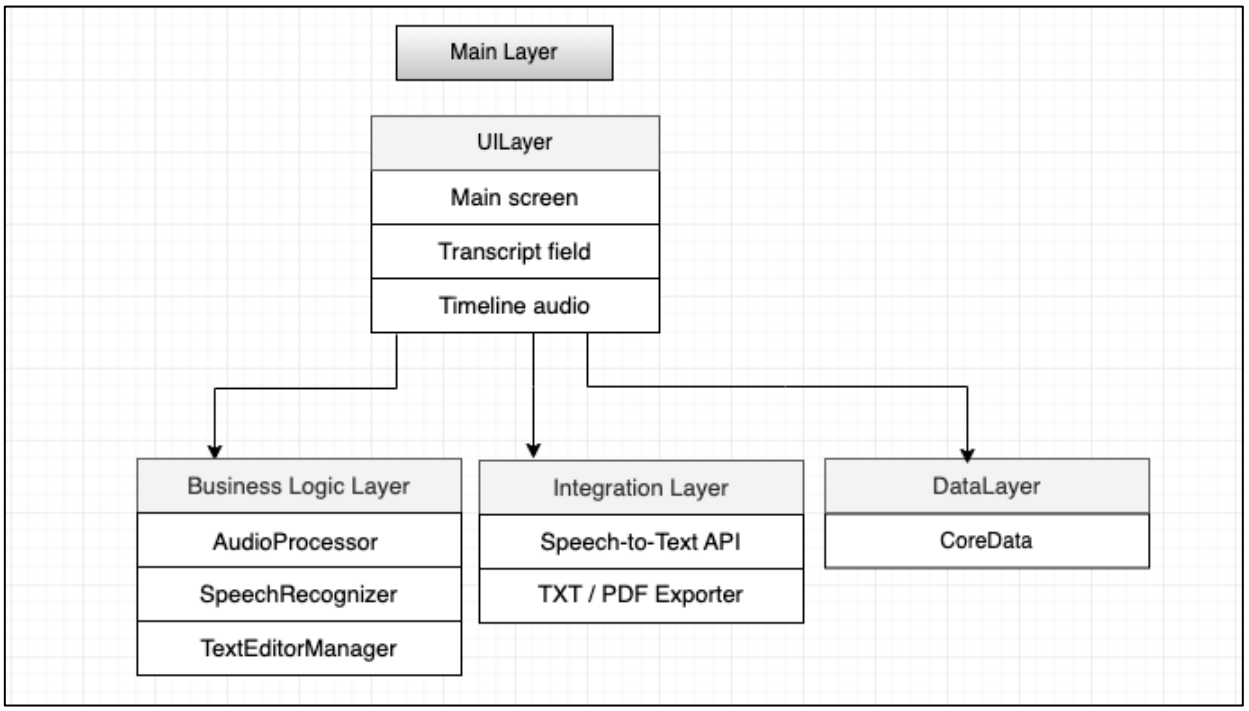


Рисунок 3.1 – Архітектура додатку

Main screen має поле для редагування транскрипту (NSTextView/Text Editor). Таймлайн аудіо із синхронізованими фрагментами тексту.

Business Logic Layer: Audio Processor – обробка аудіо файлів (шумозаглушення, конвертація). SpeechRecognizer – обробляє аудіо за

допомогою вибраного API (локальний або хмарний). TexEditorManager – обробка тексту (пошук, автозбереження, автозаміна).

Integration Layer: підключення до API для розпізнавання мовлення: Apple Speech Framework для локального розпізнавання. Модуль експорту: PDFKit для створення PDF-документів і також у текстовий формат (txt).

DataLayer: цей інструмент може зменшувати кількість коду, необхідного для роботи з даними і дозволяє зосередитися на логіці. Наприклад, завдяки CoreCata можна зберігати дані додатка, управляти об’єктами та їх змінами, оптимізувати доступ до даних, а також спрощувати синхронізацію та управління даними.

3.1.1 MVC та розподіл відповідальностей

Цей заголовок підкреслює використання патерну MVC (Model-View-Controller) у контексті описаної архітектури, а також позначає важливість розподілу відповідностей між ефективними компонентами системи.

MVC є класичним патерном програмування, яке розділяє застосування на три шари абстракції. І це дозволяє дуже полегшити внесення якихось змін до застосування в майбутньому після чіткого розподілу відповідальності між його компонентами [4, с. 109–111].

Основна мета MVC досягає чіткого розподілу відповідальності між моделлю, представленням та контролером цього (див. рис. 3.2), щоб вирішити проблему.

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;
– пасивні.

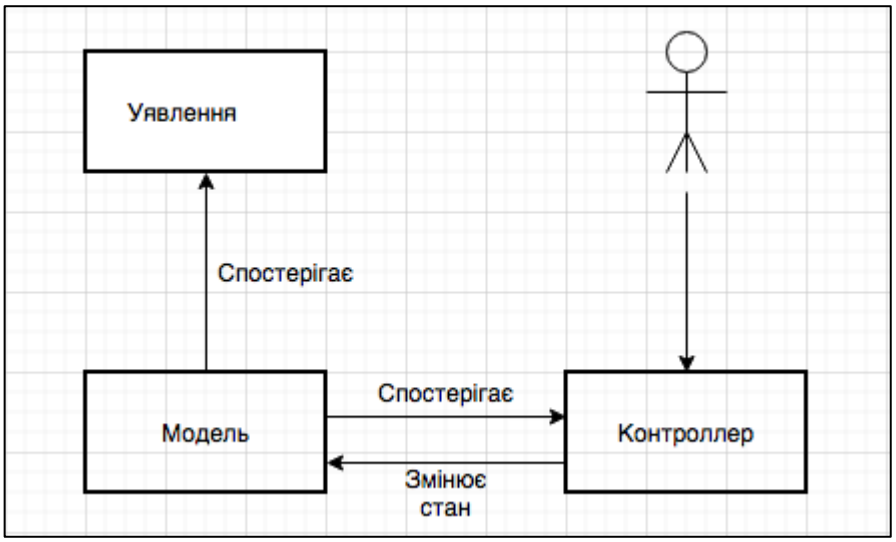


Рисунок 3.2 – класичний паттерн MVC

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 MVC View має доступ 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 (див. рис. 3.3), який задає фреймворки Cocoa та CocoaTouch для розробки додатків.

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.

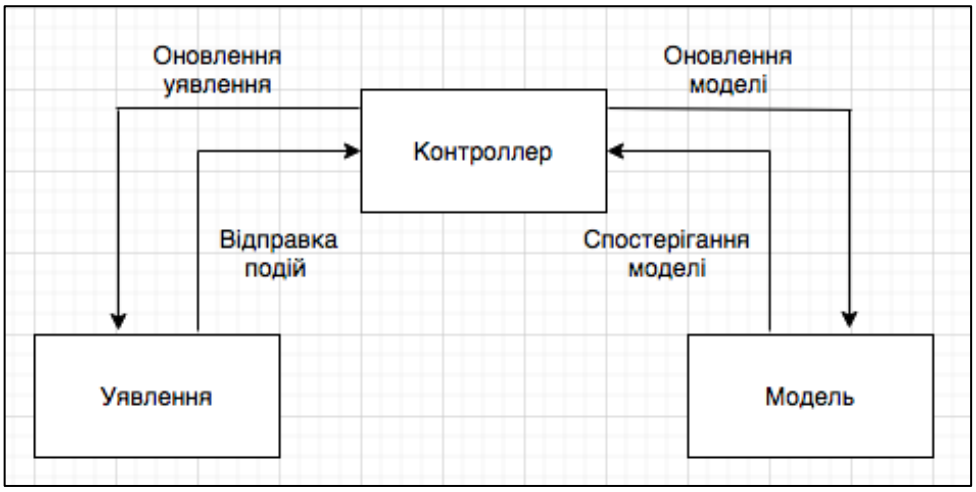


Рисунок 3.3 – Схема паттерну Apple MVC

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 проєкту.

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.

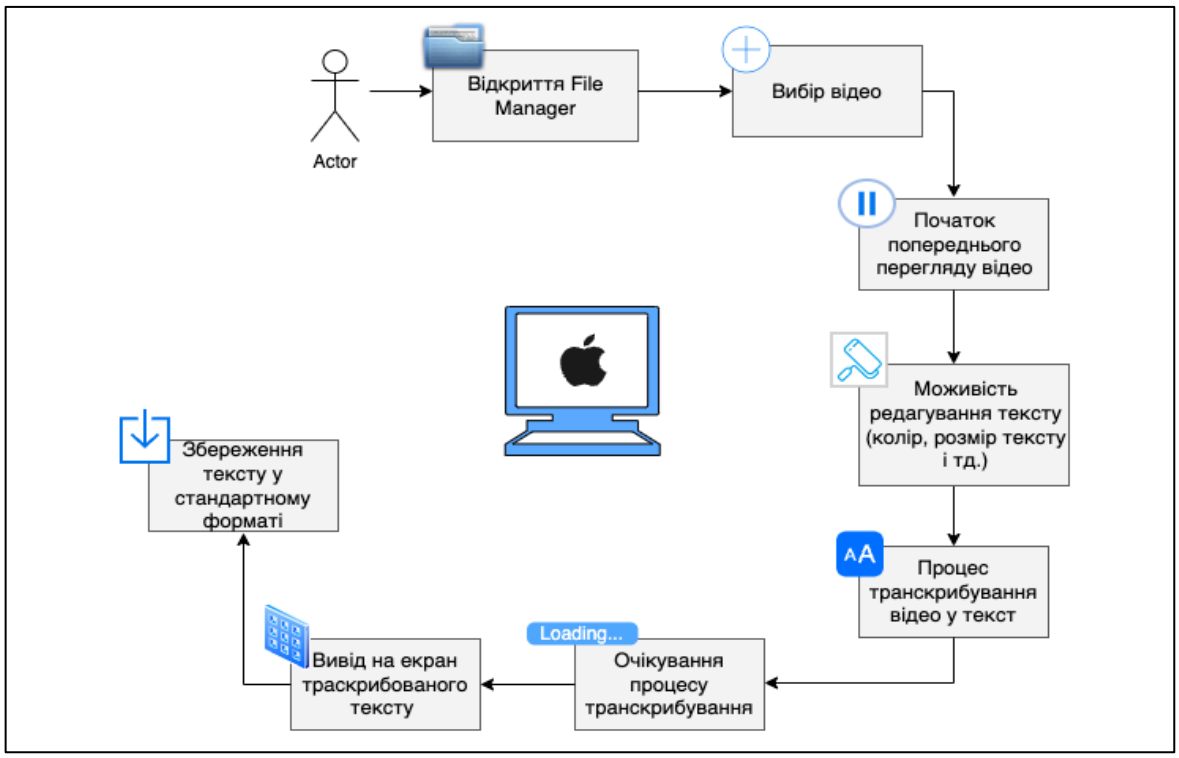


Рисунок 3.4 – Діаграма прецедентів

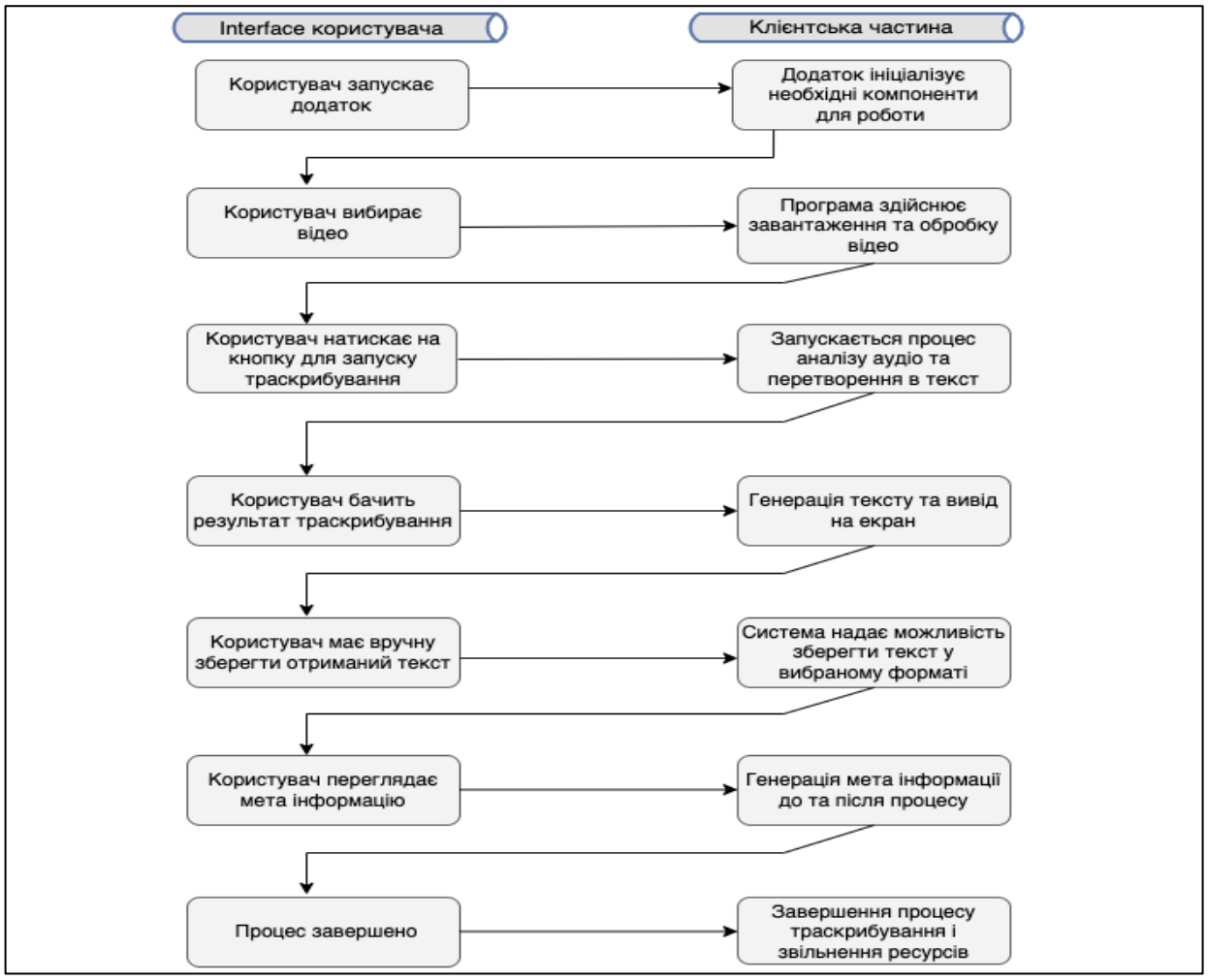


Рисунок 3.5 – Діаграма діяльності

Існує лише один актор додатку – користувач. Він може вибрати будь-яке відео на його розсуд, наприклад, різної якості, з аудіо доріжкою, великого розміру чи маленького. Після цього користувач вибирає інтервал для транскрибування відео. Потім запускається процес транскрибування. Після завершення, отриманий текст автоматично зберігається у галерею або в інший вибраний формат. Після завершення транскрибування на екрані з’являється інформація, яка показує мета інформацію до початку процесу та після цього для того, щоб побачити різницю та точність перетворення. Після завершення отриманий текст можна вручну зберегти у File Manager стандартному форматі.

Діаграма діяльності (Activity Diagram) – це один з типів діаграм, який використовується в UML для моделювання робочих процесів і динамічних аспектів системи. Вона відображає послідовність дій і процесів, які виконуються в рамках певної діяльності або бізнес-процесу [5, с. 138–139].

3.2 Файлова структура додатку

На рисунку 3.6 представлено дерево вихідних файлів, а на рисунках 3.7 – 3.9 зображена архітектура класів.

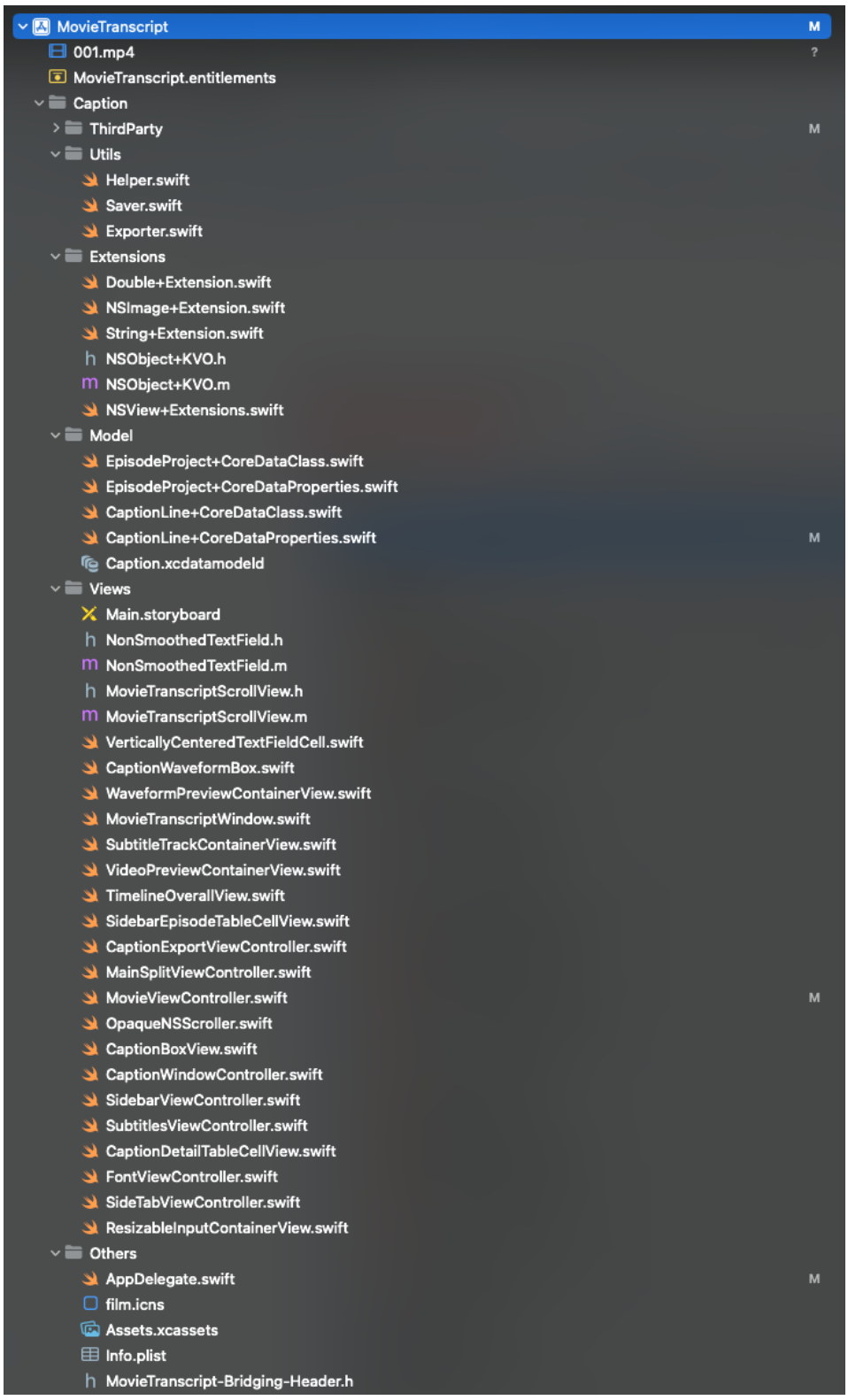


Рисунок 3.6 – Дерево вихідних файлів

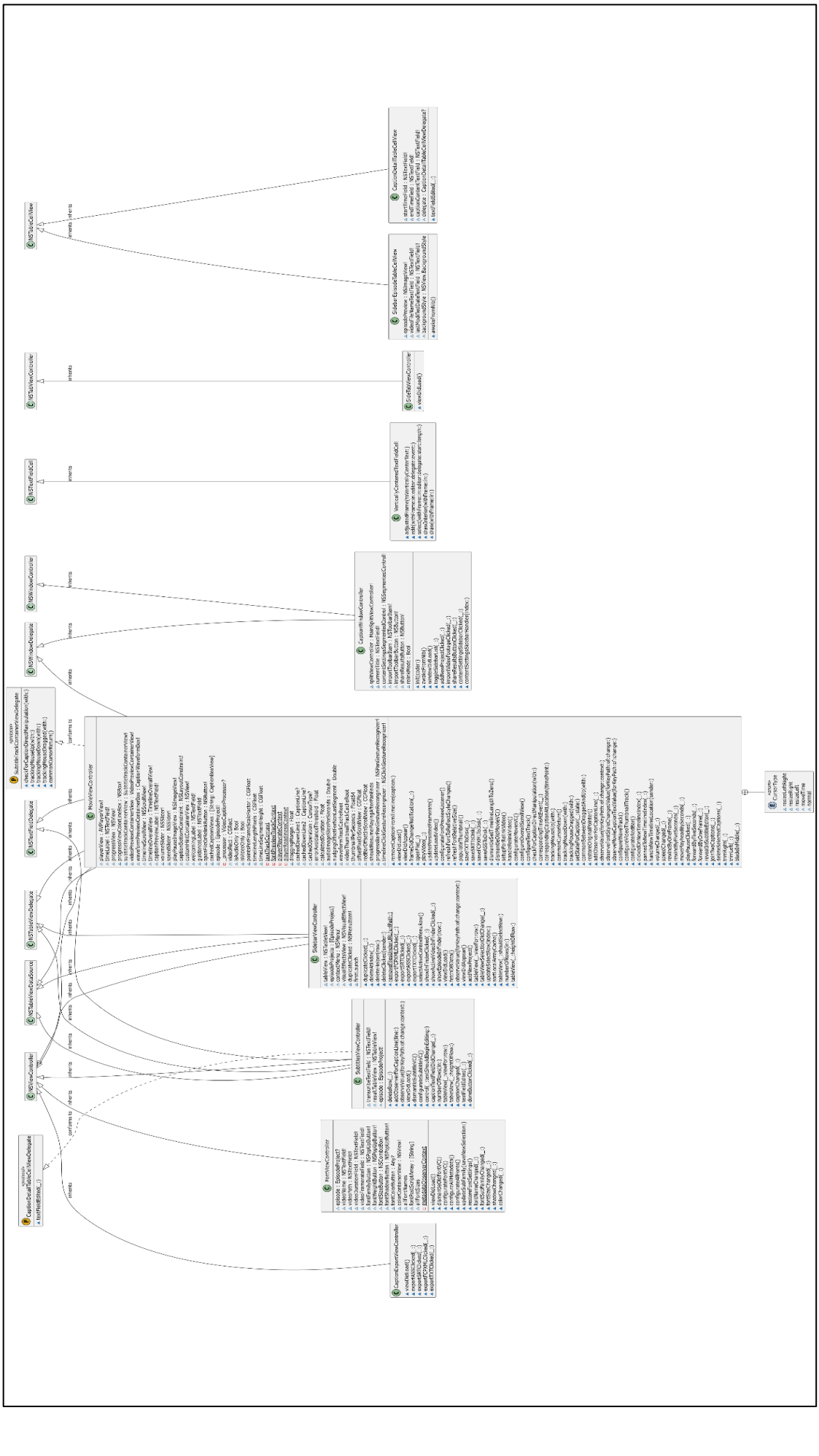


Рисунок 3.7 – Архітектура класів (1)

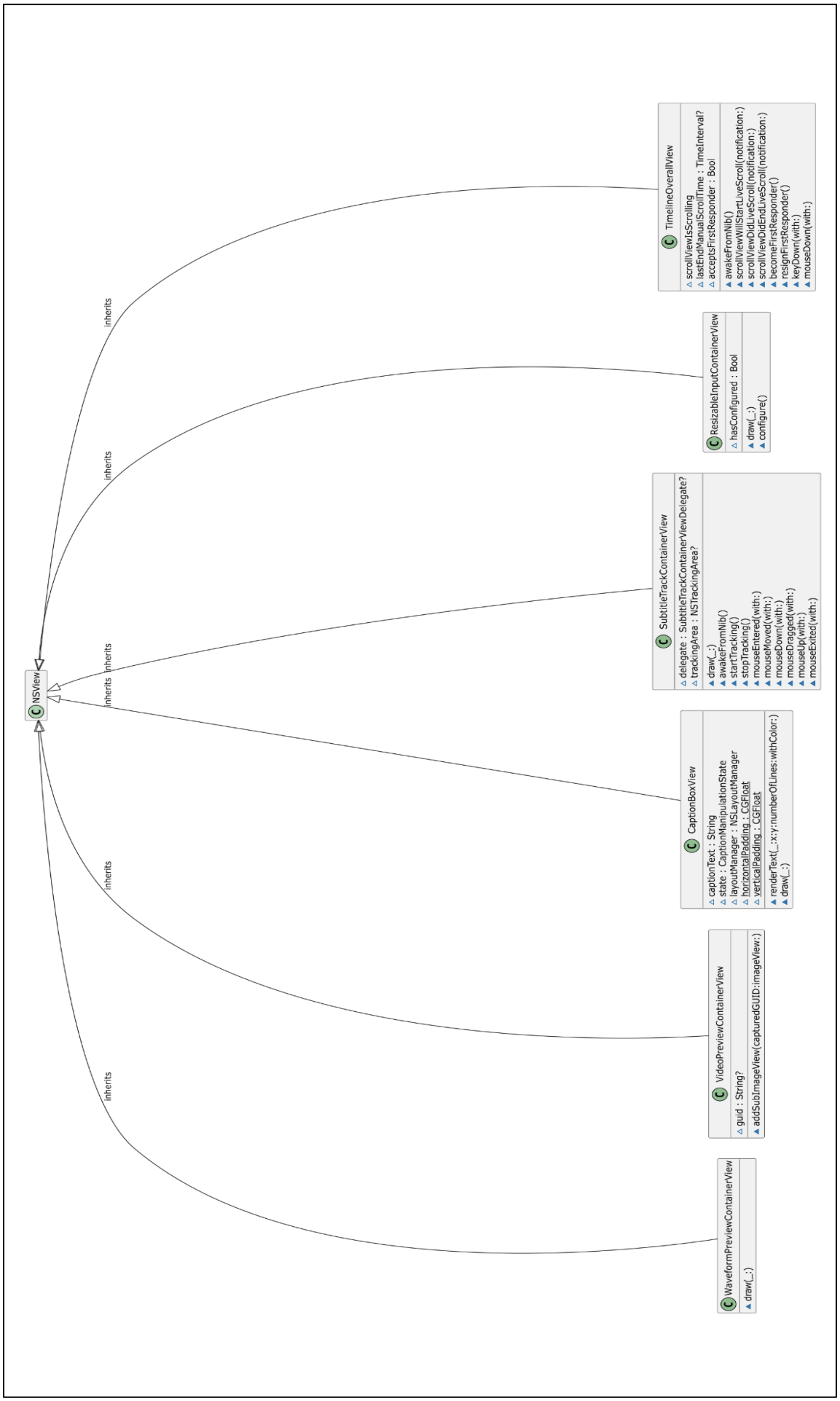


Рисунок 3.8 – Архітектура класів (2)

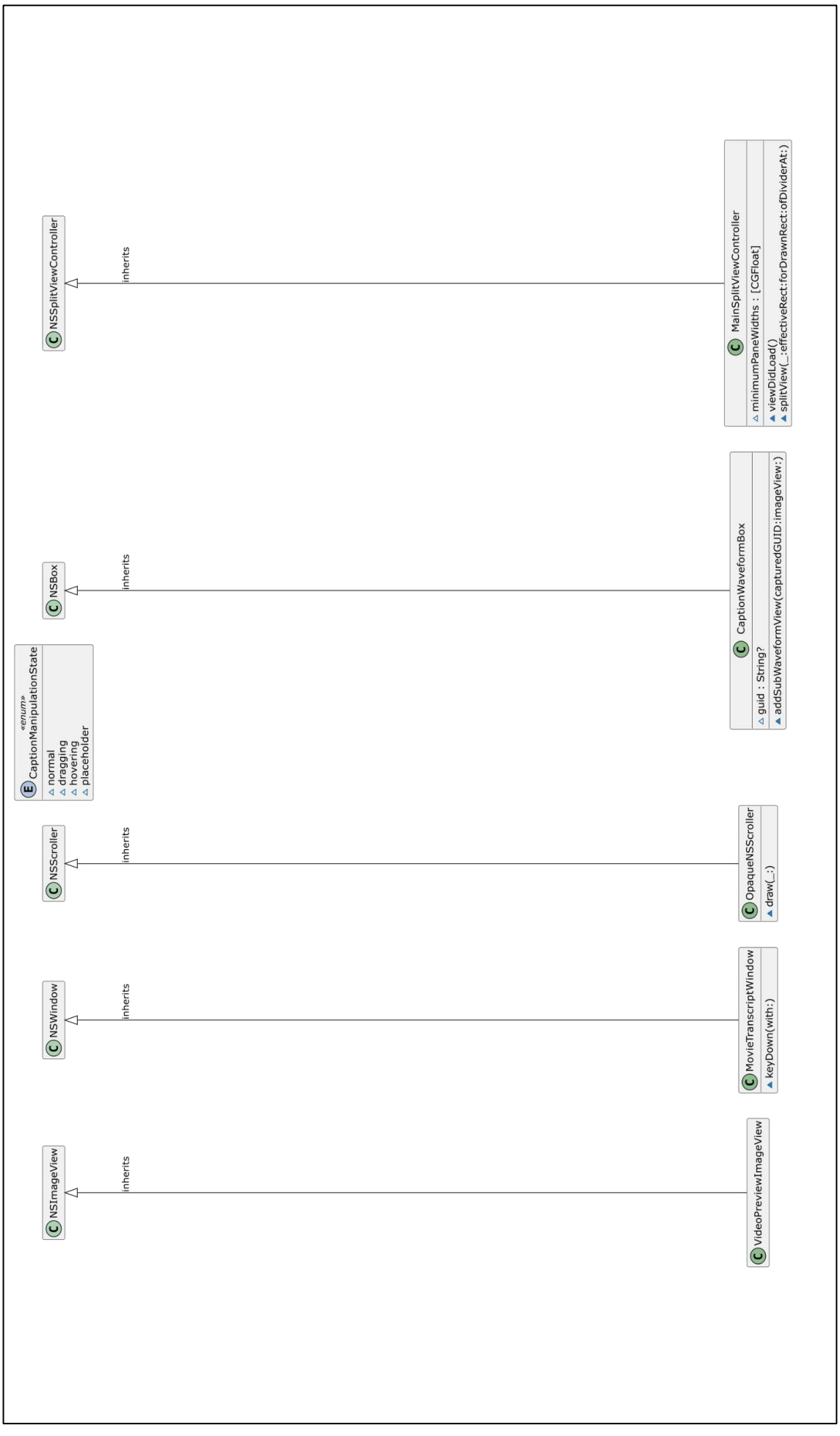


Рисунок 3.9 – Архітектура класів (3)

Кожен клас має чітко визначену відповідальність і виконує лише одну задачу, що дозволяє легко тестувати, змінювати і розширювати функціональність.

Всі складні операції (обробка аудіо, розпізнавання мови, експорт) інкапсульовані в окремих класах і не займають логіку UI-частини програми.

І сам додаток побудований так, щоб його можна було легко розширювати, додаючи нові сервіси або API без значного переписування існуючої логіки.

Ця архітектура дозволяє створити чистий та по можливості організований код, який достатньо легко підтримується та розширюється.

3.3 Організація середовища для розробки додатку

Виходячи, з попереднього розділу було обрано в якості набору ПЗ такі компоненти:

- операційна система MacOS 15.1 Sequoia (24B83);
- Xcode Version 16.0 – візуальна середа розробки;
- CocoaPods Version 1.10.5 – бібліотека;
- SourceTree – 4.2.4 (258).

3.4 Створення інтерфейсної частини

Розглянемо детальніше інтерфейс додатку. У процесі розробки практичної частини магістерської кваліфікаційної роботи було використано багато компонентів інтерфейсу користувача (UI) [9].

NSContainerView – компонент, який потрібен для організації основного макета програми, розділення області транскрипції, налаштувань і відображення прогресу.

NSTextField – поле введення для введення назв файлів чи коментарів, також поле виведення для відображення транскрибованого тексту.

NSButton – в додатку використано багато кнопок. Наприклад, для запуску, паузи, зупинки процесу транскрибування. Кнопки для зміни шрифтів, кольору тексту, для зберігання файлу і т.д.

NSProgressIndicator – компонент, який відображає прогрес загрузки файлу.

NSScrollView – компонент, який забеспечує прокручування в полі текстового виведення для роботи з довгими текстами.

NSImageView – компонент для відображення іконок або статусів (наприклад, успішного завершення чи помилок).

NSStackView – використовується для впорядкування і інших елементів інтерфейсу в горизонтальні чи вертикальні групи.

NSDragAndDrop – (реалізовується через Delegate) – дозволяє перетягувати аудіофайли безпосередньо в програму.

NSLabel – це текстові підписи для кнопок, полів вводу чи налаштувань.

NSMenu – головне меню додатку, яке містить різні функції для управління процесом та налаштувань.

NSMenuItem – елементи меню, що виконують окремі дії.

NSSubMenu – вкладене меню для більш структурованої організації дій.

Опис інтерфейсної частини додатку – це важлива складова технічної документації, яка допомагає зрозуміти, як виглядає і працює користувальницький (UI). Для правильного опису слід дотримуватись таких кроків:

Загальні відомості.

Назва додатку: MovieTranscript.

Призначення інтерфейсу:

– переглядати, редагувати та копіювати розпізнаний текст;
– зберігати транскрибований текст у файл в різних форматах;
– перегляд відео (можливо і в сповільненому режимі);
– вибір налаштування тексту субтитрів: розмір, шрифт, колір;

– можливість редагування транскрибованого тексту (зміна часу, зміна чи додавання тексту).

Опис основних екранів.

Це зображення головного вікна додатку MovieTranscript (див. рис. 3.10), призначеного для створення текстових транскрипцій або субтитрів з відео чи аудіофайлів. Він надає користувачам можливість завантажувати медіафайли, переглядати їх, редагувати розпізнаний текст субтитрів і налаштовувати вигляд тексту під час програвання відео.

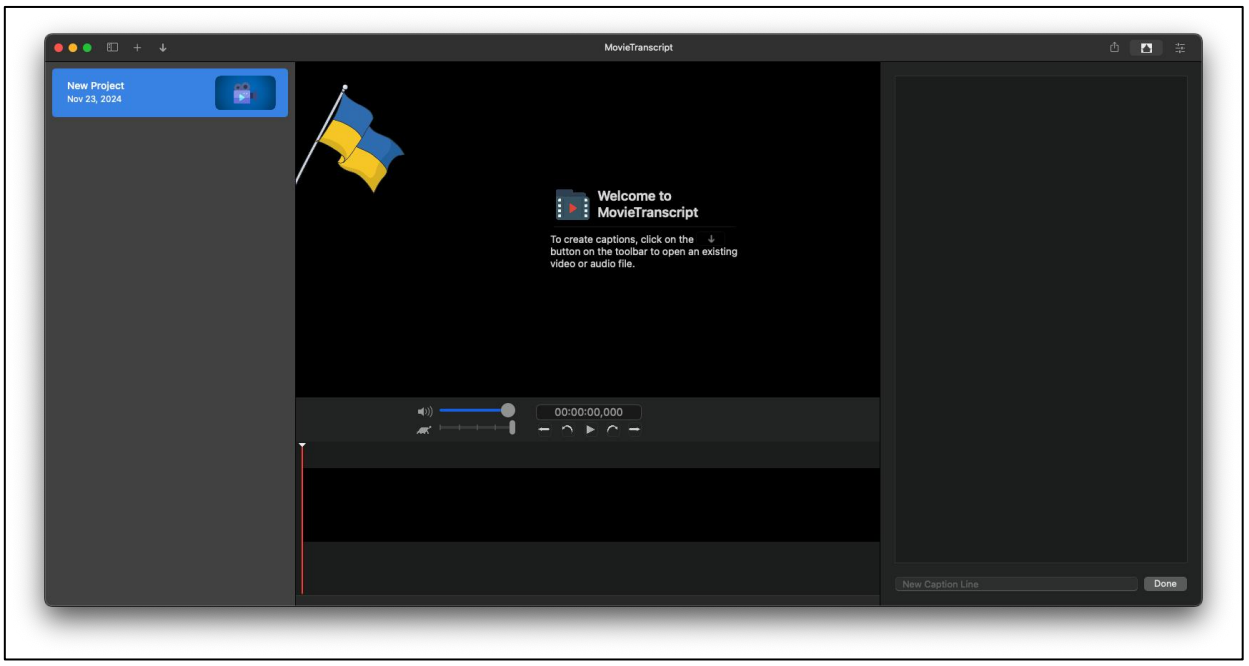


Рисунок 3.10 – Стартова сторінка додатку

Елементи інтерфейсу:

– ліва панель відображає список створених проєктів (див. рис. 3.11);

– центральна область відображає привітання і при створенні проєктів служить областю для перегляду відео або виведення аудіодоріжки (див. рис. 3.12).

На рисунку 3.13 зображена частина інтерфейсу програми, яка відображає дані відео файлу та налаштування параметрів відображення субтитрів. Інтерфейс поділено на два розділи: Video Metadata та Caption Appearance.

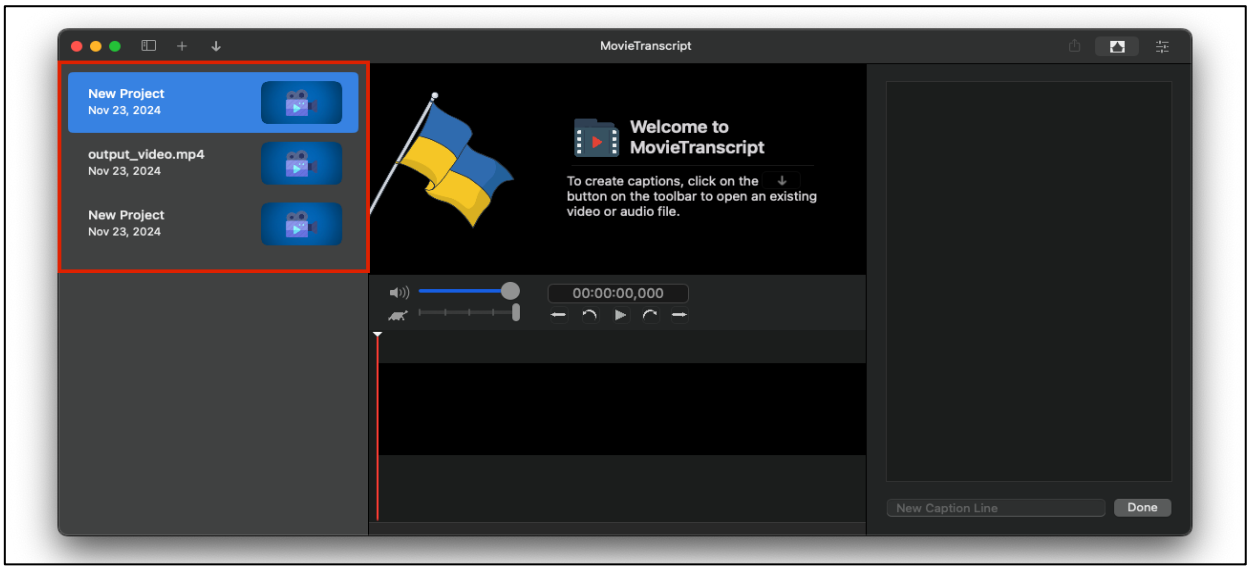


Рисунок 3.11 – Список створених проєктів у додатку



Рисунок 3.12 – Метадані відео у додатку

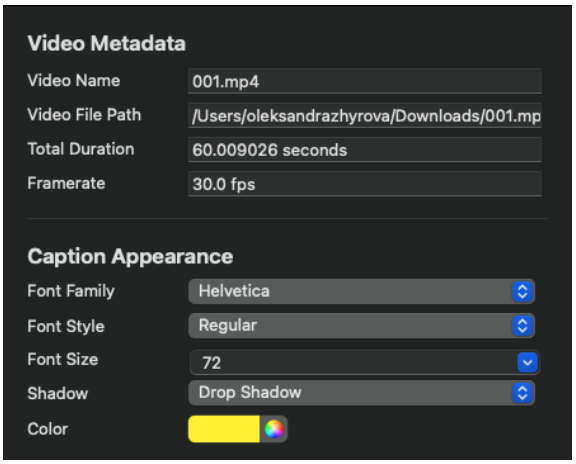


Рисунок 3.13 – Метадані відео у додатку

Video Metadata (Метадані відео) має наступні компоненти:

– назва відеофайлу;

– шлях до файлу на диску;

– загальна тривалість відео (сек);

– частота кадрів (кадрів в секунду).

Caption Appearance (Оформлення субтитрів) має такі компоненти:

– обраний шрифт для субтитрів;

– стиль шрифту;

– розмір шрифту;

– ефект тіні для тексту;

– колір тексту.

Ці налаштування дозволяють редагувати зовнішній вигляд субтитрів для відео, включаючи шрифт, стиль, розмір, тінь та колір.

Зображення, яке зображено (див. рис. 3.14), представляє знімок екрану, на якому відображено відео з доданими субтитрами. Ці субтитри є результатом роботи, за допомогою програмного забезпечення, написаного на Swift, і синхронізовані з відеорядом.



Рисунок 3.14 – транскрибування аудіо в текст

Додатково до цього, маємо на знімку транскрипцію відео, а саме текстовий запис усіх розмов чи звуків із відео, який включає тайм-коди для зручної синхронізації. Вона також створена і оброблена на мові Swift, що забезпечує її точність і автоматичну інтеграцією з субтитрами.

Розбивка відео покадрово – це набір кадрів, які були отримані за допомогою Swift-скриптів, що дозволяють обробляти відео з максимальною деталізацією, аналізуючи кожний кадр окремо. Такий підхід забезпечує:

– автоматизовану обробку відео та тексту завдяки алгоритмам на Swift;
– покадровий аналіз для корекції субтитрів та перевірки відповідності зображення та тексту;
– зручний і ефективний робочий процес, орієнтований на точність і високу якість і тексту.

На наступному зображенні демонструється функціонал збереження файла (див. рис. 3.15), в якому міститься транскрибований текст із відео.



Рисунок 3.15 – Демонстрація збереження файла

Відкривається вікно збереження, що дозволяє користувачеві вибрати бажаний формат файла (див. рис. 3.16).



Рисунок 3.16 – Доступні формати для збереження файлів

Доступні формати:

– TXT – звичайний текстовий файл, що підходить для простого перегляду або подальшого редагування (див. рис. 3.17, 3.18);
– SRT – стандартний формат субтитрів, який широко використовується в медіаплеєрах і відеоредакторах (див. рис. 3.19, 3.20);
– ASS – розширений формат субтитрів, з можливістю задавати шрифти, кольори, анімації, та інші якійсь стилістичні ефекти (див. рис. 3.21, 3.22);
– FCPXML – файл у форматі XML, який призначений для імпорту у різні відео редактори, для зручної роботи з субтитрами безпосередньо у проєкті (див. рис. 3.23, 3.24).

```
static func generateTXTFromArray(arrayForCaption: [CaptionLine]) -> String {
    var txtString = ""
    for i in 0..<arrayForCaption.count {
        if let str: String = arrayForCaption[i].caption {
            if str.count > 0 {
                txtString = txtString + "\(str)\n"
            }
        }
    }
    return txtString
}
```

Рисунок 3.17 – Створення TXT формату



Рисунок 3.18 – Результат зберігання у TXT форматі

```
// MARK: - Persistence
static func generateSRTFromArray(arrayForCaption: [CaptionLine]) -> String {
    var srtString = ""
    for i in 0..<arrayForCaption.count {
        let str: String = arrayForCaption[i].description
        if str.count > 0 {
            srtString = srtString + "\(i+1)\n\(str)\n\n"
        }
    }
    return srtString
}
```

Рисунок 3.19 – Створення SRT формату

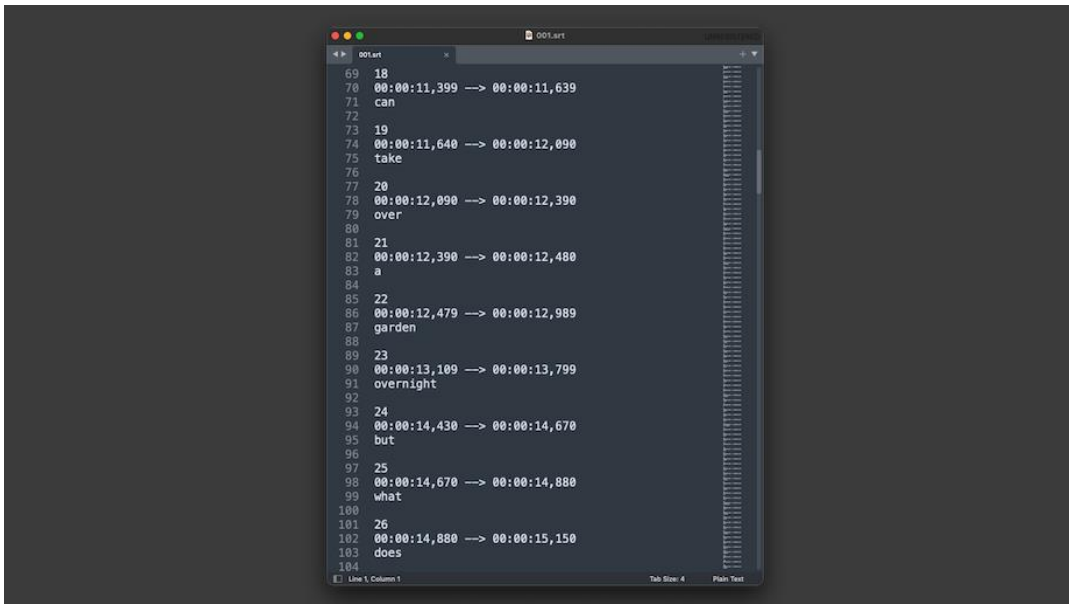


Рисунок 3.20 – Результат зберігання у SRT форматі

```swift
static func generateASSFromArray(arrayForCaption: [CaptionLine], episode: EpisodeProject) -> String {
    let fontFamilyName = episode.styleFontFamily ?? "Helvetica"
    var isBold = "0"
    var isItalic = "0"
    let fontFace = episode.styleFontWeight?.lowercased() ?? ""
    if fontFace.contains("bold") {
        isBold = "1"
    }
    if fontFace.contains("italic") || fontFace.contains("oblique") {
        isItalic = "1"
    }

    let fontSize = Float(episode.styleFontSize ?? "53") ?? 53
    let scaledResultingSize = "\(Int((20.0 / 53.0) * fontSize))"
    let fontColor = (episode.styleFontColor ?? "ffffff").replacingOccurrences(of: "#", with: "").uppercased()
    let shadow = episode.styleFontShadow == 1 ? "4" : "0"

    var assString = """
    [Script Info]
    Collisions: Normal
    ScaledBorderAndShadow: no
    ScriptType: v4.00+
    Synch Point: 1

    [V4+ Styles]
    Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, OutlineColour,
    BackColour, Bold, Italic, Underline, StrikeOut, ScaleX, ScaleY, Spacing, Angle,
    BorderStyle, Outline, Shadow, Alignment, MarginL, MarginR, MarginV, Encoding
    Style: Default,\(fontFamilyName),\(scaledResultingSize),&H00\(fontColor),&HF0000000,&H00000000,&H32000000,\(isBold),\(isItalic),0,0,100,100,0,0.00,1,0,\(shadow),2,5,5,15,1

    [Events]
    Format: Layer, Start, End, Style, Name, MarginL, MarginR, MarginV, Effect, Text
    """
    for i in 0..<arrayForCaption.count {
        let startEndTime = arrayForCaption[i].assStartEndTime
        if let content = arrayForCaption[i].caption {
            if content.count > 0 {
                assString = assString + """
                \nDialogue: 0,\(startEndTime),Default,,0,0,0,,{\\rDefault}\(content)
                """
            }
        }
    }
    return assString
}
```

Рисунок 3.21 – Створення ASS формату

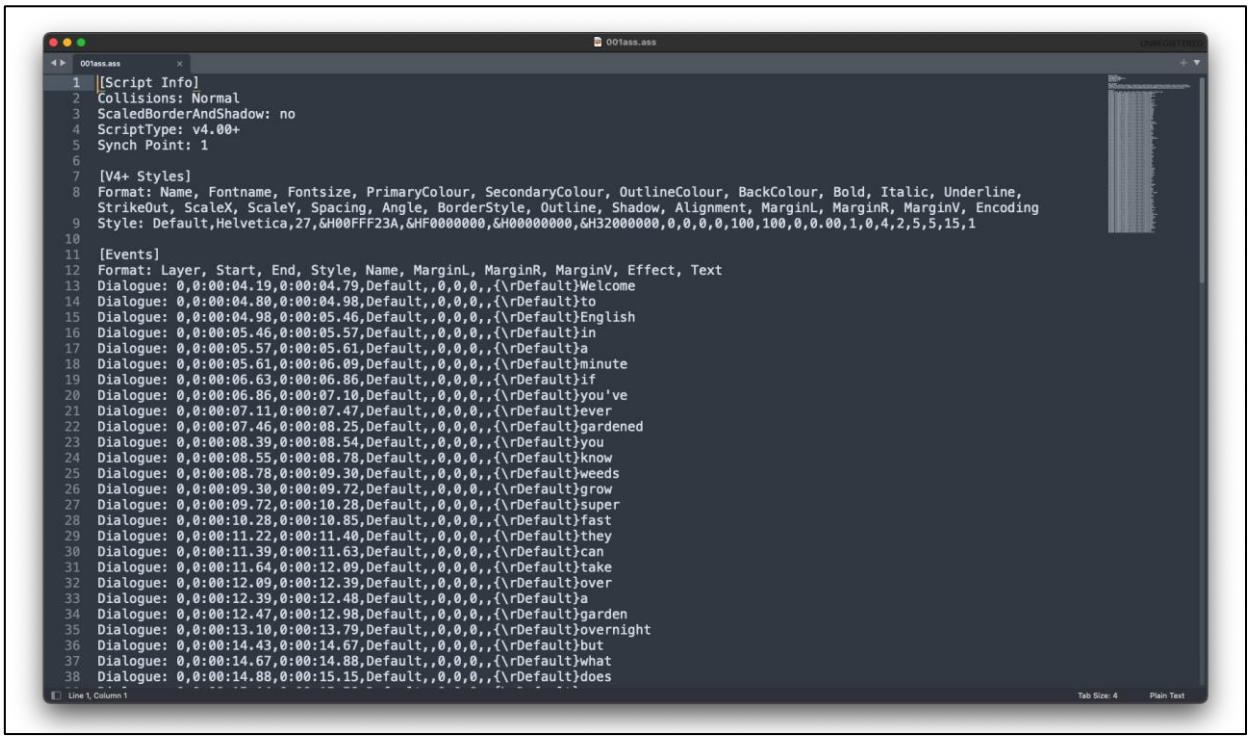


Рисунок 3.22 – Результат зберігання у ASS форматі

Ця функція забезпечує користувачам гнучкість і універсальність у збереженні результатів транскрипції, дозволяючи легко інтегрувати текст у різні робочі процеси, від базового редагування до професійного монтажу відео.

```swift
static func generateTimedFCPXMLFromArray(episode: EpisodeProject, arrayForCaption: [CaptionLine], fpsDouble: Double, callback: @escaping((_ success: Bool, _ result: String) ->())) {
    let totalDuration = Double(episode.videoDuration)
    var fpsFCPXValue = ""
    var templateName = ""
    for (doub, str) in fpsToFrameDuration {
        if fpsDouble.checkIsEqual(toDouble: doub, includingNumberOfFractionalDigits: 2) {
            fpsFCPXValue = str
        }
    }
    for (doub, str) in fpsToTemplateName {
        if fpsDouble.checkIsEqual(toDouble: doub, includingNumberOfFractionalDigits: 2) {
            templateName = str
        }
    }

    if fpsFCPXValue.count == 0 {
        let options: [Double] = [23.976, 24, 25, 29.97, 30, 50, 59.94, 60]
        var winningIndex = 0
        var winningDifference: Double = Double.greatestFiniteMagnitude
        for i in 0..<options.count {
            let absDiff = abs(options[i] - fpsDouble)
            if absDiff < winningDifference {
                winningIndex = i
                winningDifference = absDiff
            }
        }
        let dropdownOptions = ["23.976 fps", "24 fps", "25 fps", "29.97 fps", "30 fps", "50 fps", "59.94 fps", "60 fps"]
        Helper.displayInteractiveSheet(title: "Choose Framerate", text: "Before exporting, check the framerate of your Final Cut Pro X project. To view your project framerate in Final
            if selectedOK {
                #if DEBUG
                print("selected:\(selectedOK), index:\(index)")
                #endif
                let value = dropdownOptions[index].replacingOccurrences(of: " fps", with: "")
                if let newFpsDouble = Double(value) {
                    generateTimedFCPXMLFromArray(episode: episode, arrayForCaption: arrayForCaption, fpsDouble: newFpsDouble, callback: callback)
                }
            }
        }
    } else {
        let generated = finishGeneratingFCLXML(episode: episode, arrayForCaption: arrayForCaption, fpsFCPXValue: fpsFCPXValue, totalDuration: totalDuration, fpsDouble: fpsDouble, templ
        callback(true, generated)
    }
}
```

Рисунок 3.23 – Створення FCPXML формату

```
001.fcpxml.fcpxml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE fcpxml>

<fcpxml version="1.3">
    <project name="001" uid="6D05F0D0-DB25-4610-9CA9-5C64973D4975">
        <resources>
            <format id="r1" name="FFVideoFormat1080p30" frameDuration="100/3000s" width="1920" height="1080"></
            format>
            <effect id="r2" name="Basic Title" uid=".../Titles.localized/Bumper:Opener.localized/Basic
            Title.localized/Basic Title.moti"/>
        </resources>
        <sequence duration="36000/3600s" format="r1" tcStart="0s" tcFormat="NDF" audioLayout="stereo" audioRate="
        48k">
            <spine>
                <gap offset="0s" name="Master" duration="180000/3000s" start="0s">
                <!--Title No. 1 ++++++++++++++++++++++-->
                <title name="Welcome - Caption" lane="1" offset="12500/3000s" duration="1700/3000s" ref="r2" role=
                "titles.English_en">
                    <param name="Position" key="9999/999166631/999166633/1/100/101" value="1.0188 -436.431"/>
                    <param name="Flatten" key="9999/999166631/999166633/2/351" value="1"/>
                    <param name="Alignment" key="9999/999166631/999166633/2/354/999169573/401" value="1 (Center)"
                    />
                    <text>
                        <text-style ref="ts1-1">Welcome</text-style>
                    </text>
                    <text-style-def id="ts1-1">
                        <text-style font="Helvetica" fontSize="72" fontFace="Regular" fontColor="1.000000
                        0.949020 0.227451 1.000000" shadowColor="0 0 0 0.75" shadowOffset="5 315" shadowBlurRadius
                        ="2.14" alignment="center"/>
                    </text-style-def>
                    <note>en - 89F2F5B4-7614-4CA9-A0A3-4B241F801E05</note>
                </title>
                <!--Title No. 2 ++++++++++++++++++++++-->
                <title name="to - Caption" lane="1" offset="14400/3000s" duration="500/3000s" ref="r2" role="
                titles.English_en">
                    <param name="Position" key="9999/999166631/999166633/1/100/101" value="1.0188 -436.431"/>
                    <param name="Flatten" key="9999/999166631/999166633/2/351" value="1"/>
                    <param name="Alignment" key="9999/999166631/999166633/2/354/999169573/401" value="1 (Center)"
```

Рисунок 3.24 – Результат зберігання у FCPXML форматі

Збереження одного й того самого транскрибованого тексту в різних форматах показує гнучкість і універсальність даних, залежно від поставлених

цілей. Кожен формат створений для виконання конкретних завдань: від базового текстового аналізу до професійного відеомонтажу. Формат TXT забезпечує простоту й універсальність, SRT та ASS підходять для роботи із субтитрами, а FCPXML ідеально інтегрується у професійні відеоредактори. Це дозволяє адаптувати транскрипцію до різноманітних сценаріїв використання, розширюючи можливості роботи з текстом і відео.

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

Семантичні помилки – це 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.

В укладенні вищезазначеного можна сказати, що **тестування має дві мети.**

Перша – 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.

Друга – 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 приділяю особливу увагу якості продукту та ретельному тестуванню для виявлення різних типів помилок: технічних, візуальних, логічних тощо. Це є невід'ємною частиною процесу створення програм для macOS. Перед початком розробки додатків для macOS аналітики досліджують ринок, конкурентів, цільову аудиторію та актуальні тенденції. На основі цих даних визначаються ключові цілі та завдання програми, що формують стратегію проєкту. Ретельна увага до деталей на кожному етапі є основою для досягнення успішного результату. Створення додатків для macOS передбачає дотримання вимог Apple та специфічних особливостей цієї платформи. Важливо забезпечити стабільність роботи функцій і відповідність реалізованого функціоналу первісному дизайну та проєктним вимогам. Тестування відіграє ключову роль, оскільки помилки чи недоліки у роботі програми можуть призвести до втрати користувачів. Тому цьому процесу приділяється особлива увага, і тестувальник залучається з самого початку розробки – від аналізу навігаційних схем до перевірки макетів інтерфейсу. На завершальному етапі ми проводимо тестування для виявлення технічних і візуальних помилок, вносимо необхідні виправлення та доводимо додаток до повної готовності. Після цього програма публікується в Mac App Store із правильно підготовленими описами та метаданими для ефективного просування.

3.7 Висновки до третього розділу

Розробка програми – це складний процес, що охоплює безліч етапів і потребує широкого спектра знань і навичок у сфері проєктування, програмування та тестування.

РОЗДІЛ 4 ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ПРОЄКТУ

4.1 Бізнес-план розробки програмного забезпечення

У стадії розробки бізнес-плану на розробку програмного забезпечення додатку для MacOS з назвою MacTranscript.

Даний додаток призначений для звичайних користувачів, також підходить і для компаній, із можливістю розширення функціоналу та отримання від його використання. MovieTranscript – це додаток для транскрибування аудіо з можливістю редагування, також містить інші корисні функції.

4.2 Резюме

Послуги, які надаються, полягають у створених відповідних додатках для обробки аудіо та інформатизації відповідають певним КВЕДАМ, а саме:

– КВЕД 62.01 (комп’ютерне програмування);
– КВЕД 62.02 (консультації з питань інформатизації);
– КВЕД 58.29 (видання іншого програмного забезпечення).

Цільова аудиторія продукту включає користувачів, які регулярно працюють з мультимедіа – файлами, компанії і люди, які мають слухові або інші вади. Визначення цільової аудиторії допоможе у просуванні продукту та розробці ефективної рекламної кампанії.

При реалізації проєкту середня ціна становить 950000 грн, а час реалізації від 6 до 9 місяців. Загальний, на мою думку, можна припустити, враховуючи, що середня ціна становить 950000 грн, то такий проєкт може мати різні варіанти прибутку залежно від витрат і маржі. Якщо, наприклад прибутковість складає 20%, то прибуток може бути: 950000х20%=190 000 грн.

4.2.1 Маркетинг

Для того, щоб задовольнити потреби користувачів потрібно зробити зручним інтерфейс. Багато буде призначено саме розробці зручного дизайну. Окремо дизайну, швидкодія та надійність є дуже важливими пунктами. Цим повинна займатися команда бекенд та фронтенд, і 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 реалізації розробки проєкту будуть створенні реклами на різних сайтах, і пізніше напевно, буде створений власний сайт, для перегляду інформації про застосунок та його скачування.

4.2.2 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, а також переваги ведення ФОП описані вище, хоч і приватна підприємницька форма ведення бізнесу є прекрасним вибором для більшості фізичних осіб, які здійснюють, (наприклад, у сфері IT технологій), ця структура є хорошим вибором для компаній і стартапів, які планують рости чи мають високі бізнес-плани і потенційні зобов’язання.

4.2.3 Складання фінансового бюджету

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

Витрати на проєкт оцінюється розміром у 950000 грн, а перший прибуток очікується через рік.

Особа, яка займається проєктом, є фізичною особою підприємцем 3-ої групи оподаткування.

Податок на прибуток становить 3% або 5% залежно від суми доходу, і становить 1027.66 грн на місяць.

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

4.2.4 Оцінка можливих ризиків проєкту

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

ВИСНОВКИ

Під час виконання кваліфікаційної роботи мною були вивчені сучасні технології транскрибування аудіо, які значно спрощують робочі процеси та підвищують їх ефективність. Це дозволяє автоматизувати рутинні завдання, скорочуючи час, необхідний для обробки великих обсягів інформації. Завдяки використанню таких технологій можна досягти значної економії часу та зменшити ймовірність помилок, що є аспектом для професіоналів у сфері журналістики, медіа та освіти.

Можна зробити висновок, що доцільніше підібрати тип добавки відповідно до специфіки поставлених завдань. Вибір технологій і підходів, зокрема алгоритмів транскрибування, залежить від вимог до точності, швидкості та інтеграційних можливостей з іншими системами.

Для написання кваліфікаційної роботи використано операційну систему MacOS, інтегровану мову програмування Swift та середовище розробки Xcode. Ці інструменти дозволили ефективно розробити додаток із необхідним функціоналом, а також забезпечити зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

Також у дипломі клієнту створено частину додатку, які забезпечує взаємодію для обробки аудіо-файлів, їх транскрибування та подальшої обробки тексту. Розробка цієї частини вимагала застосування принципів об’єктно-орієнтованого програмування та інтеграції з різноманітними бібліотеками для роботи з аудіо- та текстовими даними.

Загалом, робота дозволила отримати глибоке розуміння процесу розробки додатків для транскрибування аудіо та їх оптимізації для реальних умов використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Cochin S. Swift Programming. Washington, 2004. 317 p.

2. Knaster S., Dalrymple M. Swift 2.0 and Programming for Mac OS. New York, 2010. 269 p.

3. Bennet G., Less B. Swift for Absolute Beginners. USA, 2014. 389 p.

4. Manning J., Nugent T. Learning Cocoa with Swift and Objective-C. USA, 2005. 420 p.

5. Chaudhary A. Swift Create Sheet. India, 2001. 420 p.

6. Beta Testing Made Simple with TestFlight. URL: http://deweloper.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjective (дата звернення: 14.10.2024).

7. Malik W., Dalrymple M. Learn Swift on the Mac. USA, 2007. 272 p.

8. Deployment and Development. *Platform Support swift.org*. URL: https://www.swift.org/platform-support/ (дата звернення: 19.10.2024).

9. Swift – Initialize model object with init. URL: https://stackoverflow.com/questions/68213705/swift-initialise-model-object-with-initfrom-decoder (дата звернення: 17.10.2024).

10. Dalrymple M. The Swift Programming Language. USA, 2011. 455 p.

ДОДАТОК А

Реалізація програми MovieTranscript

A.1 ComboColorWell

```
//  ComboColorWell.swift
//
//  Created by Oleksandra Zhyrova on 16-09-24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa

@available(OSX 10.12, *)
@available(OSX 10.12, *)
@available(OSX 10.12, *)
class ComboColorWell: NSControl {

    // MARK: - public vars
    @IBInspectable var color: NSColor {
        get {
            return comboColorWellCell.color
        }
        set {
            comboColorWellCell.color = newValue
        }
    }

    @IBInspectable var allowClearColor: Bool {
        get {
            return comboColorWellCell.allowClearColor
        }
        set {
            comboColorWellCell.allowClearColor = newValue
        }
    }

    // MARK: - private vars
    private var comboColorWellCell: ComboColorWellCell {
        guard let cell = cell as? ComboColorWellCell else {
fatalError("ComboColorWellCell not valid") }
        return cell
    }

    // MARK: - Overridden functions

    override func resignFirstResponder() -> Bool {
        comboColorWellCell.state = .off
        return super.resignFirstResponder()
    }
    // MARK: - init & private functions

    override init(frame frameRect: NSRect) {
        super.init(frame: frameRect)
        doInit()
    }
```

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

    private func doInit() {
        cell = ComboColorWellCell()
    }
}
@available(OSX 10.12, *)
extension ComboColorWell: NSColorChanging {
    func changeColor(_ sender: NSColorPanel?) {
        if let sender = sender {
            comboColorWellCell.colorAction(sender)
        }
    }
}
@available(OSX 10.12, *)
class ComboColorWellCell: NSActionCell {
       enum ControlArea {
        case nothing
        case color
        case button
    }
       enum MouseState {
        case outside
        case over(ControlArea)
        case down(ControlArea)
        case up(ControlArea)
    }

    // MARK: - public vars

    var color = NSColor.black {
        didSet {
            controlView?.needsDisplay = true
        }
    }
      var allowClearColor = true
    // MARK: - public functions

    func mouseEntered(with event: NSEvent) {
        if isEnabled {
            mouseMoved(with: event)
        }
    }

    func mouseExited(with event: NSEvent) {
        if isEnabled {
            mouseState = .outside
        }
    }

    func mouseMoved(with event: NSEvent) {
        if isEnabled {
            mouseState = .over(controlArea(for: event))
        }
    }

    @objc func colorAction(_ sender: ColorProvider) {
        action(for: sender.color)
    }

    private func action(for color: NSColor) {
```

```
        self.color = color
        if let control = controlView as? NSControl {
            control.sendAction(control.action, to: control.target)
        }
    }
    // MARK: - private vars

    private lazy var mouseTracker = {
        return MouseTracker(mouseEntered: { self.mouseEntered(with: $0) },
                            mouseExited: { self.mouseExited(with: $0) },
                            mouseMoved: { self.mouseMoved(with: $0) })
    }()

    private var mouseState = MouseState.outside {
        didSet {
            if mouseState != oldValue {
                handleMouseState(mouseState)
                controlView?.needsDisplay = true
            }
        }
    }

    private let imageInset = CGFloat(3.5)
    // MARK: - overrided vars
    override var controlView: NSView? {
        didSet {
            // add a tracking area to let our mouse tracker handle
significant events
            controlView?.addTrackingArea(NSTrackingArea(rect: NSZeroRect,

            options: [.mouseEnteredAndExited,
                    .mouseMoved,
                    .activeInKeyWindow,
                    .inVisibleRect],
                   owner: mouseTracker,
                    userInfo: nil))
        }
    }

    override var state: NSControl.StateValue {
        didSet {
            // handle the new state
            handleStateChange()
        }
    }

    // MARK: - overrided functions

    override func setNextState() {
        // disable next state default setting, called mainly by the default
cell mouse tracking
        return
    }

    override func draw(withFrame cellFrame: NSRect, in controlView: NSView) {
        // helper functions

        func fill(path: NSBezierPath, withColor color: NSColor =
.controlColor) {
            color.setFill()
            path.fill()
        }
         Fill the passed path with the passed gradient.
        func fill(path: NSBezierPath, withGradient gradient: NSGradient) {
```

```
            gradient.draw(in: path, angle: 90.0)
        }

        // hard coded colors and gradients
        let buttonGradient: NSGradient = {
            NSGradient(starting: NSColor(red: 17, green: 103, blue: 255),
                       ending: NSColor(red: 95, green: 165, blue: 255))!
        }()

        NSColor.black.withAlphaComponent(0.25).setStroke()
        // give some space to the control rect for anti aliasing
        let smoothRect = NSInsetRect(cellFrame, 0.5, 0.5)

        // the bezier path defining the control
        let path = NSBezierPath(roundedRect: smoothRect, xRadius: 6.0,
yRadius: 6.0)
        path.lineWidth = 0.0

        if state == .on {
            // on state always draws a selected button
            fill(path: path, withGradient: buttonGradient)
        } else {
            switch mouseState {
            case .outside,
                 .up:
                fill(path: path)
            case let .over(controlArea):
                switch controlArea {
                case .button:
                    // mouse over button draws a darker background
                    fill(path: path, withColor:
NSColor.lightGray.withAlphaComponent(0.25))
                default:
                    fill(path: path)
                }
            case let .down(controlArea):
                switch controlArea {
                case .button:
                    // clicked button draws selected
                    fill(path: path, withGradient: buttonGradient)
                default:
                    fill(path: path)
                }
            }
        }

         imageLiteral(resourceName: "ColorWheel").draw(in:
NSInsetRect(buttonArea(withFrame: cellFrame, smoothed: true), imageInset,
imageInset))
        NSBezierPath.clip(colorArea(withFrame: cellFrame))
        if color == .clear {
            fill(path: path, withColor: .white)
            let area = colorArea(withFrame: cellFrame)
            let blackPath = NSBezierPath()
            var point = area.origin
            blackPath.move(to: point)
            point = NSPoint(x: area.width, y: area.height)
            blackPath.line(to: point)
            point.x = area.origin.x
            blackPath.line(to: point)
            blackPath.close()
            path.addClip()
            fill(path: blackPath, withColor: .black)
        } else {
```

```
            fill(path: path, withColor: color)
        }

        path.setClip()
        path.stroke()
        if !isEnabled {
            fill(path: path, withColor: NSColor(calibratedWhite: 1.0, alpha:
0.25))
        }
        switch mouseState {
        case let .over(controlArea),
             let .down(controlArea):
            switch controlArea {
            case .color:
                 imageLiteral(resourceName: "CircledDownArrow").draw(in:
popoverButtonArea(withFrame: cellFrame, smoothed: true))
            default:
                break
            }
        default:
            break
        }

    }

    override func startTracking(at startPoint: NSPoint, in controlView:
NSView) -> Bool {
        switch controlArea(for: startPoint, in: controlView) {
        case .color:
            mouseState = .down(.color)
        case .button:
            mouseState = .down(.button)
        default:
            mouseState = .outside
        }
        return true
    }

    override func stopTracking(last lastPoint: NSPoint, current stopPoint:
NSPoint, in controlView: NSView, mouseIsUp flag: Bool) {
        if !flag {
            mouseState = .outside
            return
        }
        switch controlArea(for: stopPoint, in: controlView) {
        case .color:
            mouseState = .up(.color)
        case .button:
            mouseState = .up(.button)
        default:
            mouseState = .up(.nothing)
       }
    }

    override func continueTracking(last lastPoint: NSPoint, current
currentPoint: NSPoint, in controlView: NSView) -> Bool {
        mouseState = .down(controlArea(for: currentPoint, in: controlView))

        return true
    }

    // MARK: - private functions
    /**
     Handle mpuse state here, currently we're only interested in mouse ups.
```

```
     */
    private func handleMouseState(_ state: MouseState) {
        switch state {
            handleMouseUp(in: controlArea)
            mouseState = .over(controlArea)
        default:
            break
       }
    }
    /**
     Handle mouse up clicks here.
     */
    private func handleMouseUp(in controlArea: ControlArea) {
        switch controlArea {
        case .button:
            // toggle on and of state
            state = (state == .on ? .off : .on)
        case .color:
            // switch state off
            state = .off
            if colorsPopoverVisible {
                // popover already visible, just bail out
                return
            }
            // we need the control view to show the popove relative to it
            guard let view = controlView else { break }
            // create a popover
            let popover = NSPopover()
            popover.animates = false
            popover.behavior = .semitransient
            // make ourself its delegate
            popover.delegate = self
            // create a Color grid and set it as the popover content
            popover.contentViewController = ColorGridController(color: color,
                                                                 target: self,
                                                                 action:
#selector(colorAction(_:)),

allowClearColor: allowClearColor)
            // show the popover
            popover.show(relativeTo: popoverButtonArea(withFrame:
view.bounds), of: view, preferredEdge: .minY)
            // update the visible flag
            colorsPopoverVisible = true
        default:
            mouseState = .over(controlArea)
        }
    }

    /**
     Handle state change here.
     */
    private func handleStateChange() {
        let colorPanel = NSColorPanel.shared
        switch state {
        case .off:
            if colorPanel.isVisible,
                colorPanel.delegate === self {
                colorPanel.delegate = nil
            }
        case .on:
            if let window = controlView?.window,
                window.makeFirstResponder(controlView) {
                colorPanel.delegate = self
```

```
                colorPanel.showsAlpha = allowClearColor
                colorPanel.color = color
                colorPanel.orderFront(self)
            }
        default:
            break
        }
    }

    /**
     Get the rect of the control that displays the selected color.
     */
    private func colorArea(withFrame cellFrame: NSRect, smoothed: Bool =
false) -> NSRect {
        var rect = smoothed ? NSInsetRect(cellFrame, 0.5, 0.5) : cellFrame
        rect.size.width -= rect.size.height
        return rect
    }
    /**
     Get the rect of the control that displays the color panel button.
     */
    private func buttonArea(withFrame cellFrame: NSRect, smoothed: Bool =
false) -> NSRect {
        var rect = smoothed ? NSInsetRect(cellFrame, 0.5, 0.5) : cellFrame
        rect.origin.x += (rect.width - rect.height)
        rect.size.width = rect.height
        return rect
    }

    /**
     Get the rect of the control where a down arrow button should be drawn.
     */
    private func popoverButtonArea(withFrame cellFrame: NSRect, smoothed:
Bool = false) -> NSRect {
        let buttonSize = CGFloat(15.0)
        let rect = colorArea(withFrame: cellFrame, smoothed: smoothed)
        return NSRect(x: rect.width - (buttonSize + imageInset),
                      y: ceil((rect.height - buttonSize) / 2),
                      width: buttonSize, height: buttonSize)
    }
    /**
     Get the area of the control where a mouse event has occurred.
     */
    private func controlArea(for event: NSEvent) -> ControlArea {
        guard let controlView = controlView else { return .nothing }
        return controlArea(for: controlView.convert(event.locationInWindow,
from: nil), in: controlView)
    }

    /**
     Get the area of the control where a point lies.
     */
    private func controlArea(for point: NSPoint, in controlView: NSView) ->
ControlArea {
        if colorArea(withFrame: controlView.bounds).contains(point) {
            return .color
        } else if buttonArea(withFrame: controlView.bounds).contains(point) {
            return .button
        } else {
            return .nothing
        }
    }

}
```

```
// handle Color panel delegate events here.
@available(OSX 10.12, *)
extension ComboColorWellCell: NSWindowDelegate {
    func windowWillClose(_ notification: Notification) {
        state = .off
        controlView?.needsDisplay = true
    }
}
@available(OSX 10.12, *)
extension ComboColorWellCell: NSPopoverDelegate {
    func popoverWillClose(_ notification: Notification) {
        colorsPopoverVisible = false
    }
}
NSColor) {
      doColorAction(color)}

class MouseTracker: NSResponder {
    let mouseEnteredHandler: (_ : NSEvent) -> ()
    let mouseExitedHandler: (_ : NSEvent) -> ()
    let mouseMovedHandler: ((_ : NSEvent) -> ())?
    /**
     The designated initializer.
     Requires handlers for the entered and exited events.
     Moved event handler is optional.
     */
    init(mouseEntered enteredHandler: @escaping (_ event: NSEvent) -> (),
         mouseExited exitedHandler: @escaping (_ event: NSEvent) -> (),
         mouseMoved movedHandler: ((_ event: NSEvent) -> ())? = nil) {
        mouseEnteredHandler = enteredHandler
        mouseExitedHandler = exitedHandler
        mouseMovedHandler = movedHandler
        super.init()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    override func mouseEntered(with event: NSEvent) {
        mouseEnteredHandler(event)
    }

    override func mouseExited(with event: NSEvent) {
        mouseExitedHandler(event)
    }

    override func mouseMoved(with event: NSEvent) {
        mouseMovedHandler?(event)
    }

}
/**
 A controller for a grid view to show and select colors.
 */
@available(OSX 10.12, *)
class ColorGridController: NSViewController {

    // MARK: - public vars

    /**
     The color we want to show as selected in the grid.
     */
    var color = NSColor.black {
```

```
        didSet {
            // try to select the color in the grid view.
            (view as? ColorGridView)?.selectColor(color)
        }
    }

    /**
     Set this to false if you don't want the popover to show the clear color
in the grid.
     */
    var allowClearColor = true {
        didSet {
            // propagate setting to the grid view.
            (view as? ColorGridView)?.allowClearColor = allowClearColor
        }
    }
    // MARK: - private vars

    /**
     The target that will receive the action message, only it neither is nil,
when color has been chosen.
     */
    private weak var target: AnyObject?
    /**
     The action that will be sent to the target, only it neither is nil, when
color has been chosen.
     */
    private var action: Selector?
    /**
     The delegate that will be notified when color has been chosen.
     Deprecated approach.
     */
    private weak var delegate: ColorGridViewDelegate?

    // MARK: - init & overrided functions

    init() {
        super.init(nibName: nil, bundle: nil)
    }

    convenience init(delegate: ColorGridViewDelegate) {
        self.init()
        self.delegate = delegate
    }

    convenience init(color: NSColor, target: AnyObject, action: Selector,
allowClearColor: Bool = true) {
        self.init()
        self.color = color
        self.target = target
        self.action = action
        self.allowClearColor = allowClearColor
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
    override func loadView() {
        // create here our color grid
        let colorGrid = ColorGridView()
        view = colorGrid
        colorGrid.delegate = self
        colorGrid.allowClearColor = allowClearColor
        colorGrid.selectColor(color)
```

```
    }
}
// conform to the ColorGridViewDelegate protocol.
@available(OSX 10.12, *)
extension ColorGridController: ColorGridViewDelegate {
    /**
     Handle the color choice.
     */
    func colorGridView(_ colorGridView: ColorGridView, didChoose color:
NSColor) {
        self.color = color
        view.window?.performClose(self)
        if let target = target,
            let action = action {
            let _ = target.perform(action, with: self)
        }
        delegate?.colorGridView(colorGridView, didChoose: color)
    }
}
/**
 Add ColorProvider conformance to NSPanel
 */
@available(OSX 10.12, *)
extension ColorGridController: ColorProvider {}
/**
 The protocol for a delegate to handle color choice.
 */
@available(OSX 10.12, *)
protocol ColorGridViewDelegate: AnyObject {
    func colorGridView(_ colorGridView: ColorGridView, didChoose color:
NSColor)
}
/**
 A grid of selectable color view objects.
 */
@available(OSX 10.12, *)
class ColorGridView: NSGridView {
    // MARK: - public vars
    weak var delegate: ColorGridViewDelegate?

    /**
     An array of NSColor arrays, meant to be presented as columns in the
grid.
     */
    var colorArrays: [[NSColor]] = [[NSColor(red: 72, green: 179, blue: 255),
                                     NSColor(red: 18, green: 141, blue: 254),
                                     NSColor(red: 12, green: 96, blue: 172),
                                     NSColor(red: 7, green: 59, blue: 108),
                                     .white],
                                    [NSColor(red: 102, green: 255, blue: 228),
                                     NSColor(red: 36, green: 228, blue: 196),
                                     NSColor(red: 20, green: 154, blue: 140),
                                     NSColor(red: 14, green: 105, blue: 99),
                                     NSColor(red: 205, green: 203, blue: 203)],
                                    [NSColor(red: 122, green: 255, blue: 62),
                                     NSColor(red: 83, green: 212, blue: 42),
                                     NSColor(red: 32, green: 166, blue: 3),
                                     NSColor(red: 13, green: 97, blue: 2),
                                     NSColor(red: 128, green: 128, blue: 128)],
                                    [NSColor(red: 255, green: 255, blue: 85),
                                     NSColor(red: 249, green: 222, blue: 40),
                                     NSColor(red: 245, green: 173, blue: 9),
                                     NSColor(red: 253, green: 128, blue: 8),
                                     NSColor(red: 76, green: 76, blue: 76)],
```

```
                                    [NSColor(red: 253, green: 129, blue: 122),
                                     NSColor(red: 251, green: 76, blue: 62),
                                     NSColor(red: 230, green: 0, blue: 14),
                                     NSColor(red: 164, green: 0, blue: 2),
                                     .black],
                                    [NSColor(red: 252, green: 116, blue: 185),
                                     NSColor(red: 232, green: 67, blue: 151),
                                     NSColor(red: 189, green: 14, blue: 104),
                                     NSColor(red: 133, green: 1, blue: 76),
                                     .clear]] {
        didSet {
            setupGrid()
        }
    }

    /**
     Set this to false if you don't want the popover to show the clear color
in the grid.
     */
    var allowClearColor = true {
        didSet {
            if let colorView = colorView(for: .clear) {
                colorView.isHidden = !allowClearColor
            }
        }
    }

    // MARK: - public functions
    /**
     Try to select the element in the grid that represents the passed color.
     */
    @discardableResult func selectColor(_ color: NSColor) -> Bool {
        if let colorView = colorView(for: color) {
            colorView.selected = true
            return true
        }
        return false
    }
    // MARK: - init & overrided functions

    init() {
        super.init(frame: NSZeroRect)

        rowSpacing = 1.0
        columnSpacing = 1.0

        setupGrid()

    }

    convenience init(in view: NSView) {
        self.init()

        // make sure to disable autoresizing mask translations
        translatesAutoresizingMaskIntoConstraints = false

        // add the grid view programmatically (macOS 10.12 doesn't play well
with IB instantiated grids)
        view.addSubview(self)

        // hook the borders of the grid to the parent view
        view.addConstraints([NSLayoutConstraint(equalAttribute: .top,
 for: (self, view)),
                             NSLayoutConstraint(equalAttribute: .bottom,
```

```
for: (self, view)),
                                NSLayoutConstraint(equalAttribute: .trailing,
 for: (self, view)),
                                NSLayoutConstraint(equalAttribute: .leading,
for: (self, view))])


    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // MARK: - private functions
    /**
     Build the colors grid here.
     */
    private func setupGrid() {
        // start with an empty grid
        (0..<numberOfRows).forEach { removeRow(at: $0) }

        // get colors as arrays of ColorView objects
        let views = colorArrays.map {
            return $0.map { ColorView(color: $0, in: self) }
        }

        // Treat each array in views as a column of the grid
        views.forEach { addColumn(with: $0) }

        setPadding(5.0)

        // set grid elements size and placement
        (0..<numberOfColumns).forEach {
            let column = self.column(at: $0)
            column.width = 35
            column.xPlacement = .fill
        }

        (0..<numberOfRows).forEach {
            let row = self.row(at: $0)
            row.height = 20
            row.yPlacement = .fill
        }
    }

    /**
     Set top, bottom, left and right margins as padding.
     */
    private func setPadding(_ padding: CGFloat) {
        guard numberOfRows > 0 else { return }

        row(at: 0).topPadding = padding
        row(at: numberOfRows - 1).bottomPadding = padding

        guard numberOfColumns > 0 else { return }

        let firstCol = column(at: 0)
        let lastCol = column(at: numberOfColumns - 1)

        firstCol.leadingPadding = padding
        lastCol.trailingPadding = padding

    }
```

```
    /**
     Try to find the element in the grid that represents the passed color.
     */
    private func colorView(for color: NSColor) -> ColorView? {
        for (columnIndex, colorArray) in colorArrays.enumerated() {
            if let rowIndex = colorArray.firstIndex(of: color) {
                return column(at: columnIndex).cell(at: rowIndex).contentView
as? ColorView
            }
        }
        return nil
    }

    /**
     User has selected a color, tell it to the delegate.
     */
    fileprivate func colorSelected(_ color: NSColor) {
        delegate?.colorGridView(self, didChoose: color)
    }

}
/**
 A view to represent a color in a grid.
 */
@available(OSX 10.12, *)
class ColorView: NSView {

    // MARK: - public vars

    let color: NSColor

    var selected = false {
        didSet {
            needsDisplay = true
        }
    }
    // MARK: - private vars

    private weak var colorGridView: ColorGridView?
    // MARK: - init & overrided functions

    init(color: NSColor, in colorGridView: ColorGridView) {
        self.color = color
        self.colorGridView = colorGridView
        super.init(frame: NSZeroRect) // NSRect(origin: NSPoint(x: 0, y: 0),
size: NSSize(width: 50, height: 30)))
    }

    required init?(coder decoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    override func draw(_ dirtyRect: NSRect) {
        guard let context = NSGraphicsContext.current?.cgContext else {
return }

        if color == .clear {
            NSColor.white.setFill()
        } else {
            color.setFill()
        }

        context.fill(dirtyRect)
```

```
        if color == .clear {
            NSColor.red.setStroke()
            context.beginPath()
            context.move(to: dirtyRect.origin)
            context.addLine(to: CGPoint(x: dirtyRect.width, y:
dirtyRect.height))
            context.strokePath()
        }

        if selected {
            NSColor.white.setStroke()
        } else {
            NSColor.gray.withAlphaComponent(0.5).setStroke()
        }
        context.stroke(dirtyRect, width: selected ? 2.0 : 1.0)
    }

    override func mouseDown(with event: NSEvent) {
        selected = true
    }
    override func mouseDragged(with event: NSEvent) {
        let point = self.convert(event.locationInWindow, from: nil)
        selected = bounds.contains(point)
    }
    override func mouseUp(with event: NSEvent) {
        if selected {
            colorGridView?.colorSelected(color)
        }
    }
}
// MARK: - Protocols & Extensions
/**
 handy protocol for classes that have a color var
*/
@objc protocol ColorProvider {
    var color: NSColor { get set }
}
/**
 Add ColorProvider conformance to NSPanel
 */
extension NSColorPanel: ColorProvider {}
/**
 Add equatable conformance to MouseState enum
 */
@available(OSX 10.12, *)
extension ComboColorWellCell.MouseState: Equatable {
    static func == (lhs: ComboColorWellCell.MouseState, rhs:
ComboColorWellCell.MouseState) -> Bool {
        switch lhs {
        case .outside:
            switch rhs {
            case .outside:
                return true
            default:
                return false
            }
        case let .over(leftArea):
            switch rhs {
            case let .over(rightArea):
                return leftArea == rightArea
            default:
                return false
            }
        case let .down(leftArea):
```

```
            switch rhs {
            case let .down(rightArea):
                return leftArea == rightArea
            default:
                return false
            }
        case let .up(leftArea):
            switch rhs {
            case let .up(rightArea):
                return leftArea == rightArea
            default:
                return false
            }
        }
    }
}
extension NSLayoutConstraint {
    public convenience init(equalAttribute: NSLayoutConstraint.Attribute,
                            for items: (NSView, NSView?),
                            multiplier: CGFloat = 1.0,
                            constant: CGFloat = 0.0) {

        self.init(item: items.0,
                  attribute: equalAttribute,
                  relatedBy: .equal,
                  toItem: items.1,
                  attribute: (items.1 != nil) ?
                    equalAttribute :
                    .notAnAttribute,
                  multiplier: multiplier,
                  constant: constant)
    }
}
extension NSColor {
    convenience init(red: Int, green: Int, blue: Int, alpha: CGFloat = 1.0) {
        self.init(calibratedRed: CGFloat(red) / 255, green: CGFloat(green) /
255, blue: CGFloat(blue) / 255, alpha: alpha)
    }
}
class myTextView: NSTextView {}
```

A.2 Double+Extension

```
//
//  Double+Extension.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 18.11.2024.
//  Copyright © 2024 zho. All rights reserved.
//
extension Double {
       func checkIsEqual(toDouble pDouble : Double,
includingNumberOfFractionalDigits : Int) -> Bool {
        let denominator         : Double = pow(10.0,
Double(includingNumberOfFractionalDigits))
        let maximumDifference   : Double = 1.0 / denominator
        let realDifference      : Double = fabs(self - pDouble)
        if realDifference >= maximumDifference {
```

```
            return false
        } else {
            return true
        }
    }
}
```

A.3 NSImage+Extension

```
//  NSImage+Extension.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 18.11.2024.
//  Copyright © 2024 zho. All rights reserved.
//
extension NSImage {
    var pngData: Data? {
        guard let tiffRepresentation = tiffRepresentation, let bitmapImage =
NSBitmapImageRep(data: tiffRepresentation) else { return nil }
        return bitmapImage.representation(using: .png, properties: [:])
    }
    func pngWrite(to url: URL, options: Data.WritingOptions = .atomic) ->
Bool {
        do {
            try pngData?.write(to: url, options: options)
            return true
        } catch {
            print(error)
            return false
        }
    }
}
```

A.4 String+Extension

```
//  String+Extension.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 18.11.2024.
//  Copyright © 2024 Bright. All rights reserved.
//
extension String {
    var withoutFileExtension: String {
        get {
            var components = self.components(separatedBy: ".")
            guard components.count > 1 else { return self }
            components.removeLast()
            return components.joined(separator: ".")
        }
    }
    var fileExtension: String {
        get {
            let components = self.components(separatedBy: ".")
```

```
            guard components.count > 1 else { return "" }
            return components.last ?? ""
        }
    }
}
```

A.5 EpisodeProject+CoreDataProperties

```
//  EpisodeProject+CoreDataProperties.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 10/17/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Foundation
import CoreData
extension EpisodeProject {
    @nonobjc public class func fetchRequest() ->
NSFetchRequest<EpisodeProject> {
        return NSFetchRequest<EpisodeProject>(entityName: "EpisodeProject")
    }
    @NSManaged public var guidIdentifier: String?
    @NSManaged public var videoDescription: String?
    @NSManaged public var videoURL: String?
    @NSManaged public var thumbnailURL: String?
    @NSManaged public var creationDate: NSDate?
    @NSManaged public var modifiedDate: NSDate?
    @NSManaged public var styleFontFamily: String?
    @NSManaged public var styleFontWeight: String?
    @NSManaged public var styleFontColor: String?
    @NSManaged public var styleFontSize: String?
    @NSManaged public var framerate: Float
    @NSManaged public var videoDuration: Float
    @NSManaged public var styleFontShadow: Int16
    @NSManaged public var arrayForCaption: NSOrderedSet?
}
// MARK: Generated accessors for arrayForCaption
extension EpisodeProject {
    @objc(insertObject:inArrayForCaptionAtIndex:)
    @NSManaged public func insertIntoArrayForCaption(_ value: CaptionLine,
at idx: Int)
    @objc(removeObjectFromArrayForCaptionAtIndex:)
    @NSManaged public func removeFromArrayForCaption(at idx: Int)
    @objc(insertArrayForCaption:atIndexes:)
    @NSManaged public func insertIntoArrayForCaption(_ values: [CaptionLine],
 at indexes: NSIndexSet)
    @objc(removeArrayForCaptionAtIndexes:)
    @NSManaged public func removeFromArrayForCaption(at indexes: NSIndexSet)
    @objc(replaceObjectInArrayForCaptionAtIndex:withObject:)
    @NSManaged public func replaceArrayForCaption(at idx: Int,
 with value: CaptionLine)
    @objc(replaceArrayForCaptionAtIndexes:withArrayForCaption:)
    @NSManaged public func replaceArrayForCaption(at indexes: NSIndexSet,
with values: [CaptionLine])
    @objc(addArrayForCaptionObject:)
    @NSManaged public func addToArrayForCaption(_ value: CaptionLine)
    @objc(removeArrayForCaptionObject:)
    @NSManaged public func removeFromArrayForCaption(_ value: CaptionLine)
```

```
    @objc(addArrayForCaption:)
    @NSManaged public func addToArrayForCaption(_ values: NSOrderedSet)
    @objc(removeArrayForCaption:)
    @NSManaged public func removeFromArrayForCaption(_ values: NSOrderedSet)
}
```

A.6 CaptionLine+CoreDataProperties

```
//
//  CaptionLine+CoreDataProperties.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 10/17/24.
//  Copyright © 2024 zho. All rights reserved.

import Foundation
import CoreData
extension CaptionLine {
    @nonobjc public class func fetchRequest() -> NSFetchRequest<CaptionLine>
{
        return NSFetchRequest<CaptionLine>(entityName: "CaptionLine")
    }

    @NSManaged public var guidIdentifier: String?
    @NSManaged public var caption: String?
    @NSManaged public var startingTime: Float
    @NSManaged public var endingTime: Float
    @NSManaged public var episodeProject: EpisodeProject?
}
import AVFoundation
import Speech
extension CaptionLine
{
    static func createCaptionLine(captionLineMetaInfo : CaptionLineMetaInfo)
-> CaptionLine
    {
        let description = NSEntityDescription.entity(forEntityName:
"CaptionLine", in: Helper.context!)
        let cap = CaptionLine(entity: description!, insertInto:
Helper.context!)
        cap.guidIdentifier = NSUUID().uuidString
        cap.caption = captionLineMetaInfo.caption
        cap.startingTime = captionLineMetaInfo.startingTime
        cap.endingTime = captionLineMetaInfo.endingTime
        return cap
    }
}
class CaptionLineMetaInfo {
    var caption: String
    var startingTime: Float
    var endingTime: Float
    init(caption: String, startingTime: Float, endingTime: Float) {
        self.caption = caption
        self.startingTime = startingTime
        self.endingTime = endingTime
    }
}
class VideoCaptionProcessor {
```

```
    private let recognizer = SFSpeechRecognizer(locale: Locale(identifier:
"en-US"))!
    private let videoFileURL: URL
    init(videoFileURL: URL) {
        self.videoFileURL = videoFileURL
    }
    func processCaptions(completion: @escaping ([CaptionLineMetaInfo]?,
 Error?) -> Void) {
        extractAudioSamples { [weak self] audioFileURL, error in
            guard let self = self
            else { return }

            if let error = error {
                completion(nil, error)
                return
            }
            guard let audioFileURL = audioFileURL else {
                completion(nil, NSError(domain: "AudioFileExtraction", code:
-1, userInfo: [NSLocalizedDescriptionKey: "Failed to extract audio file"]))
                return
            }
            self.recognizeSpeech(from: audioFileURL, completion: completion)
        }}
    private func extractAudioSamples(completion: @escaping (URL?, Error?) ->
Void) {
        let asset = AVAsset(url: videoFileURL)
        asset.loadTracks(withMediaType: .audio)
        { traks, error in
            print("traks: \(traks), error: \(error)")
            guard let audioTrack = asset.tracks(withMediaType: .audio).first
else {
                completion(nil, NSError(domain: "AudioTrack", code: -1,
userInfo: [NSLocalizedDescriptionKey: "No audio track found"]))
                return
            } do {
                let outputURL =
FileManager.default.temporaryDirectory.appendingPathComponent("extractedAudio
.m4a")
                if FileManager.default.fileExists(atPath: outputURL.path) {
                    try FileManager.default.removeItem(at: outputURL)  }

                let exportSession = AVAssetExportSession(asset: asset,
presetName: AVAssetExportPresetAppleM4A)!
                exportSession.outputFileType = .m4a
                exportSession.outputURL = outputURL
                exportSession.timeRange = CMTimeRange(start: .zero, duration:
asset.duration)
                exportSession.exportAsynchronously {
                    if exportSession.status == .completed {
                        completion(outputURL, nil)
                    } else {
                        completion(nil, exportSession.error)
                    }
                }
            } catch {
                completion(nil, error)
            }
        }
    }
    private func recognizeSpeech(from audioFileURL: URL, completion:
@escaping ([CaptionLineMetaInfo]?, Error?) -> Void) {
        let request = SFSpeechURLRecognitionRequest(url: audioFileURL)
        request.shouldReportPartialResults = false
        recognizer.recognitionTask(with: request) { result, error in
```

```
            if let error = error {
                completion(nil, error)
                return  }
            guard let result = result else {
                completion(nil, NSError(domain: "SpeechRecognition", code: -
1, userInfo: [NSLocalizedDescriptionKey: "No recognition result"]))
                return   }
            var captionLineMetaInfos = [CaptionLineMetaInfo]()
            for segment in result.bestTranscription.segments {
                let caption = segment.substring
                let startingTime = Float(segment.timestamp)
                let endingTime = startingTime + Float(segment.duration)
                captionLineMetaInfos.append(CaptionLineMetaInfo(caption:
caption, startingTime: startingTime, endingTime: endingTime))   }
            completion(captionLineMetaInfos, nil)
        } } }
```

A.7 VerticallyCenteredTextFieldCell

```
//
//  VerticallyCenteredTextFieldCell.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 10/17/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
class VerticallyCenteredTextFieldCell: NSTextFieldCell {
    func adjustedFrame(toVerticallyCenterText rect: NSRect) -> NSRect {
        // super would normally draw text at the top of the cell
        var titleRect = super.titleRect(forBounds: rect)
        let minimumHeight = self.cellSize(forBounds: rect).height
        titleRect.origin.y += (titleRect.height - minimumHeight) / 2
        titleRect.size.height = minimumHeight
        return titleRect
    }
    override func edit(withFrame rect: NSRect, in controlView: NSView, editor
textObj: NSText, delegate: Any?, event: NSEvent?) {
        super.edit(withFrame: adjustedFrame(toVerticallyCenterText: rect),
in: controlView, editor: textObj, delegate: delegate, event: event)
    }
    override func select(withFrame rect: NSRect, in controlView: NSView,
editor textObj: NSText, delegate: Any?, start selStart: Int, length
selLength: Int) {
        super.select(withFrame: adjustedFrame(toVerticallyCenterText: rect),
in: controlView, editor: textObj, delegate: delegate, start: selStart,
length: selLength)
    }
    override func drawInterior(withFrame cellFrame: NSRect, in controlView:
NSView) {
        super.drawInterior(withFrame: adjustedFrame(toVerticallyCenterText:
cellFrame),
in: controlView)
    }
    override func draw(withFrame cellFrame: NSRect, in controlView: NSView) {
        super.draw(withFrame: cellFrame,
in: controlView)
    }
}
```

A.8 MovieTranscriptWindow

```
//  MovieTranscriptWindow.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 10/17/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
class MovieTranscriptWindow: NSWindow {
    override func keyDown(with event: NSEvent) {
        #if DEBUG
        print("Intercepted keyDown: \(event)")
        #endif
        if event.keyCode == 49 && event.characters == " " {
            AppDelegate.movieVC()?.playPauseClicked(self)
        } else {
            super.keyDown(with: event)
        } } }
```

A.9 VideoPreviewContainerView

```
//  VideoPreviewContainerView.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 10/18/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
class VideoPreviewContainerView: NSView {
    var guid: String?
    func addSubImageView(capturedGUID: String?, imageView:
VideoPreviewImageView) {
        if let cap = capturedGUID {
            if (self.guid == cap) {
                self.addSubview(imageView)
            }
        }
    }
}
class VideoPreviewImageView: NSImageView {}
```

A.10 TimelineOverallView

```
//
//  TimelineOverallView.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 10/18/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
```

```
class TimelineOverallView: NSView {
    override func awakeFromNib() {
        let nc =  NotificationCenter.default
        nc.addObserver(self, selector:
#selector(scrollViewWillStartLiveScroll(notification:)), name:
NSScrollView.willStartLiveScrollNotification, object: nil)
        nc.addObserver(self, selector:
#selector(scrollViewDidLiveScroll(notification:)), name:
NSScrollView.didLiveScrollNotification, object: nil)
        nc.addObserver(self, selector:
#selector(scrollViewDidEndLiveScroll(notification:)), name:
NSScrollView.didEndLiveScrollNotification, object: nil)
    }
    var scrollViewIsScrolling = false
    var lastEndManualScrollTime: TimeInterval?
    @objc func scrollViewWillStartLiveScroll(notification: Notification){
        scrollViewIsScrolling = true
        #if DEBUG
        print("scrollViewWillStartLiveScroll: \(#function) ")
        #endif
    }
    @objc func scrollViewDidLiveScroll(notification: Notification){
        scrollViewIsScrolling = true
        #if DEBUG
        // print("scrollViewDidLiveScroll: \(#function) ")
        #endif
    }
    @objc func scrollViewDidEndLiveScroll(notification: Notification){
        scrollViewIsScrolling = false
        lastEndManualScrollTime = Date().timeIntervalSince1970
        #if DEBUG
        print("scrollViewDidEndLiveScroll: \(#function) ")
        #endif
    }
    // MARK: - Keyboard Handling
    override var acceptsFirstResponder: Bool {
        return true
    }
    override func becomeFirstResponder() -> Bool {
        return true
    }
    override func resignFirstResponder() -> Bool {
        return true
    }
    override func keyDown(with event: NSEvent) {
        if event.keyCode == 49 && event.characters == " " {
            AppDelegate.movieVC()?.playPauseClicked(self)
        } else {
            interpretKeyEvents([event])
        }
    }
    override func mouseDown(with event: NSEvent) {
        window?.makeFirstResponder(self)
    }
}
```

A.11 SidebarEpisodeTableCellView

```
//  SidebarEpisodeTableCellView.swift
//  MovieTranscript
```

```
//
//  Created by Oleksandra Zhyrova on 11/20/24.
//  Copyright © 2024 MyOrganization. All rights reserved.
//
import Cocoa
class SidebarEpisodeTableCellView: NSTableCellView {
    @IBOutlet var episodePreview: NSImageView!
    @IBOutlet weak var videoFileNameTextField: NSTextField!
    @IBOutlet weak var lastModifiedDateTextField: NSTextField!
    override func awakeFromNib() {
        if #available(macOS 10.14, *) {
        } else {
            videoFileNameTextField.font = NSFont.systemFont(ofSize: 14,
weight: .bold)
        }
    }
    override var backgroundStyle: NSView.BackgroundStyle {
        willSet {
            if newValue == .dark {
                videoFileNameTextField.textColor = NSColor.white
                lastModifiedDateTextField.textColor = NSColor.white
            } else {
                videoFileNameTextField.textColor = NSColor.labelColor
                lastModifiedDateTextField.textColor = NSColor.labelColor
            }}}}
```

A.12 CaptionExportViewController

```
//  CaptionExportViewController.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 11/20/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
class CaptionExportViewController: NSViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
    }
    @IBAction func exportASSClicked(_ sender: NSButton) {
        AppDelegate.movieVC()?.saveASSToDisk(self)
    }
    @IBAction func exportSRTClicked(_ sender: NSButton) {
        AppDelegate.movieVC()?.saveSRTToDisk(self)
    }
    @IBAction func exportFCPXMLClicked(_ sender: NSButton) {
        AppDelegate.movieVC()?.saveFCPXMLToDisk(self)
    }
    @IBAction func exportTXTClicked(_ sender: NSButton) {
        AppDelegate.movieVC()?.saveTXTToDisk(self)  }}
```

A.13 MainSplitViewController

```
//
//  MainSplitViewController.swift
```

```
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 11/11/24.
//  Copyright © 2024 MyOrganization. All rights reserved.
//
import Cocoa
class MainSplitViewController: NSSplitViewController {
    var minimumPaneWidths: [CGFloat] = [85, 450, 450]
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do view setup here.
    }
    override func splitView(_ splitView: NSSplitView, effectiveRect
proposedEffectiveRect: NSRect, forDrawnRect drawnRect: NSRect, ofDividerAt
dividerIndex: Int) -> NSRect {
        return NSZeroRect
    } }
```

A.14 ViewController

```
//  ViewController.swift
//  Caption
//
//  Created by Oleksandra Zhyrova on 11/12/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
import AVKit
import AVFoundation
class MyTextField: NSTextField {
    override func setFrameOrigin(_ newOrigin: NSPoint) {
        print("**** MyTextField.setFrameOrigin: \(newOrigin)")
        super.setFrameOrigin(newOrigin)
    }

    override func setFrameSize(_ newSize: NSSize) {
        print("**** MyTextField.setFrameSize: \(newSize)")
        super.setFrameSize(newSize)
    }
}
@objc class MovieViewController: NSViewController, NSTableViewDelegate,
NSTableViewDataSource, NSTextFieldDelegate,
NSWindowDelegate {
    @IBOutlet weak var playerView: AVPlayerView!
    @IBOutlet weak var timeLabel: NSTextField!
    @IBOutlet weak var progressView: NSView!
    @IBOutlet weak var progressViewColorLineBox: NSBox!
    @IBOutlet weak var subtitleTrackContainerView:
SubtitleTrackContainerView!
    @IBOutlet weak var videoPreviewContainerView: VideoPreviewContainerView!
    @IBOutlet weak var waveformPreviewContainerBox: CaptionWaveformBox!
    @IBOutlet weak var timelineScrollView: NSScrollView!
    @IBOutlet weak var timelineOverallView: TimelineOverallView!
    @IBOutlet weak var captionPreviewLabel: MyTextField!
    @IBOutlet weak var volumeSlider: NSSlider!
    @IBOutlet weak var speedSlider: NSSlider!
    @IBOutlet weak var playPauseImageView: NSImageView!
    @IBOutlet weak var captionBottomConstraint: NSLayoutConstraint!
    @IBOutlet weak var customHintContainerView: NSView!
```

```
    @IBOutlet weak var welcomingLabel: NSTextField!
    @IBOutlet weak var guidanceLabel: NSTextField!
    @IBOutlet weak var openFileOrRelinkButton: NSButton!
    var episode: EpisodeProject!
    //MARK: - View Controller Lifecycle
    override func viewDidLoad() {
        super.viewDidLoad()
        if #available(OSX 10.11, *) {
            timeLabel.font = NSFont.monospacedDigitSystemFont(ofSize: 15,
weight: .regular)
        } else {
            timeLabel.font = NSFont.systemFont(ofSize: 15)
        }
        if #available(OSX 10.14, *) {
            timelineScrollView.appearance = NSAppearance.init(named:
.darkAqua)
            timelineScrollView.backgroundColor = .controlBackgroundColor
        } else {
            timelineScrollView.backgroundColor = NSColor.init(calibratedRed:
20/255, green: 20/255, blue: 20/255, alpha: 1)
        }
        playerView.wantsLayer = true
        customHintContainerView.wantsLayer = true
        captionPreviewLabel.wantsLayer = true
        customHintContainerView.layer?.zPosition = 1
        captionPreviewLabel.layer?.zPosition = 1
        #if DEBUG
        print("customHintContainerView.layer is: \(String(describing:
customHintContainerView.layer)), captionPreviewLabel.layer is:
\(String(describing: captionPreviewLabel.layer))")
        #endif
        AppDelegate.subtitleVC()?.dismantleSubtitleVC()
        AppDelegate.subtitleVC()?.configurateSubtitleVC()
        subtitleTrackContainerView.wantsLayer = true
        subtitleTrackContainerView.layer?.backgroundColor =
NSColor.clear.cgColor
        videoPreviewContainerView.wantsLayer = true
        videoPreviewContainerView.layer?.backgroundColor =
NSColor.black.cgColor
        waveformPreviewContainerBox.wantsLayer = true
        waveformPreviewContainerBox.layer?.backgroundColor =
NSColor.blue.cgColor

    }

    override func viewDidAppear() {
        self.view.window?.delegate = self
        self.playerView.postsFrameChangedNotifications = true
    }
    @objc func frameDidChangeNotification(_ sender: Any) {
        refreshFontRelativeSize()
    }
    // MARK: - Buttons and IBActions
    @IBAction func openFile(_ sender: Any) {
        if self.episode == nil {
            Helper.displayInteractiveSheet(title: "Create a Project", text:
"To import a video, you need to create a project. Click on your newly created
project in the sidebar, then import a video to create captions.",
firstButtonText: "Create Project", secondButtonText: "Dismiss") { (result) in
                if result {
                    AppDelegate.sourceListVC()?.addNewProject()
                    AppDelegate.sourceListVC()?.updateSelectRow(index: 0)
                }
            }
```

```
            return
        }
        if self.episode.videoURL != nil {
            if FileManager.default.fileExists(atPath: self.episode.videoURL!)
{
                Helper.displayInteractiveSheet(title: "Create new project?",
text: "The current project already has an associated video. Would you like to
create a new project instead?", firstButtonText: "Create New Project",
secondButtonText: "Cancel", callback: { (firstButtonClicked) in
                    if (firstButtonClicked) {
                        AppDelegate.sourceListVC()?.addNewProject()
                        AppDelegate.sourceListVC()?.updateSelectRow(index: 0)
                    }
                })
            } else {
                handleRelinkVideo()
            }
            return
        }
        Helper.displayOpenFileDialog { (hasFile, result, path) in
            if !hasFile {
                return
            }
            if let id = self.episode.guidIdentifier {
                Helper.removeFilesUnderURL(urlPath:
"~/Library/Caches/com.zho.MovieTranscript/audio_thumbnail/\(id)")
                Helper.removeFilesUnderURL(urlPath:
"~/Library/Caches/com.zho.MovieTranscript/video_thumbnail/\(id)")
                self.dismantleOldMovieVC()
            }
            if self.episode == nil {
                AppDelegate.sourceListVC()?.updateSelectRow(index: 0)
            }
            self.episode.arrayForCaption = []
            self.episode.videoDescription = result!.lastPathComponent
            self.playVideo(result!)
        }
    }
    func playVideo(_ videoURL: URL) {
        self.episode.videoURL = videoURL.path
        AppSandboxFileAccess()?.accessFileURL(videoURL,
persistPermission: true,
with: {
            self.episode.player = AVPlayer(url: videoURL)
            self.customHintContainerView.isHidden = true
            AppDelegate.mainWindow()?.shareResultsButton.isEnabled = true
            self.episode.player?.addObserver(self, forKeyPath: "rate",
options: [.new], context: &MovieViewController.playerPlayrateContext)
            self.playerView.player = self.episode.player
            self.episode.player?.addObserver(self, forKeyPath: "status",
options: [.new], context: &MovieViewController.playerReadinessContext)
        })
    }
    func updatePersistedFramerate() {
        if self.episode != nil {
            if let tracks = self.episode.player?.currentItem?.asset.tracks {
                if !tracks.isEmpty {
                    let framerate = tracks[0].nominalFrameRate
                    self.episode.framerate = framerate
                    AppDelegate.fontVC()?.configureAllMetadata()
                }
            }
        }
    }
```

```
    private var _processor : VideoCaptionProcessor?
    @objc func updateLoadVideo() {
        self.updatePersistedFramerate()

        let updateUIBlock: () -> Void = { [self] in
            self.episode.videoDuration =
Float(CMTimeGetSeconds((self.episode.player?.currentItem?.asset.duration)!))
            self.populateThumbnail()
            self.configureOverallScrollView()
            self.configurateRedBar()
            self.configureVideoThumbnailTrack()
            refreshFontToReflectStyleChanges()
            self.configurateFontPreviewListener()
            AppDelegate.fontVC()?.configureAllMetadata()
            if let desc = self.episode.videoDescription {
                AppDelegate.setCurrentEpisodeTitle(desc.withoutFileExtension)
            }
        }

        if self.episode.arrayForCaption?.count ?? 0 <= 0 {
            let videoURL = URL(fileURLWithPath: self.episode.videoURL!)
            let processor = VideoCaptionProcessor(videoFileURL: videoURL)

            processor.processCaptions { [self] captions, error in
                DispatchQueue.main.async {
                    if let error = error {
                        print("Error: \(error.localizedDescription)")
                    } else if let captions = captions {

                        for line in captions {
                            print("Caption: \(line.caption), Start:
\(line.startingTime), End: \(line.endingTime)")
                            let captionLine =
CaptionLine.createCaptionLine(captionLineMetaInfo: line )
                            print("captionLine: \(captionLine)")
                            self.episode.addToArrayForCaption(captionLine)
                        }
                        self.episode.creationDate = NSDate()
                        try! Helper.context?.save()

                        updateUIBlock()

                    }
                }
            }
            _processor = processor
        }
        else
        {
            updateUIBlock()
        }
    }
    func configurateFontPreviewListener() {
        episode.addObserver(self, forKeyPath: "styleFontColor", options:
[.new], context: &MovieViewController.fontPreviewTrackContext)
        episode.addObserver(self, forKeyPath: "styleFontFamily", options:
[.new], context: &MovieViewController.fontPreviewTrackContext)
        episode.addObserver(self, forKeyPath: "styleFontShadow", options:
[.new], context: &MovieViewController.fontPreviewTrackContext)
        episode.addObserver(self, forKeyPath: "styleFontSize", options:
[.new], context: &MovieViewController.fontPreviewTrackContext)
        episode.addObserver(self, forKeyPath: "styleFontWeight", options:
[.new], context: &MovieViewController.fontPreviewTrackContext)
    }
```

```
    func refreshFontToReflectStyleChanges() {
        var postScriptName = "Helvetica"
        let size = self.episode.styleFontSize ?? "53"
        guard let arrayofSubs =
NSFontManager.shared.availableMembers(ofFontFamily:
self.episode.styleFontFamily ?? "Helvetica"),
            let floatSize = Float(size) else { return }
        for i in 0..<arrayofSubs.count {
            if let nameOfSubFamily = arrayofSubs[i][1] as? String {
                if nameOfSubFamily == self.episode.styleFontWeight {
                    postScriptName = arrayofSubs[i][0] as? String ??
"Helvetica"
                    break
                }
            }
        }
        guard let desiredFont = NSFont.init(name: postScriptName, size:
CGFloat(floatSize)) else { return }
        self.captionPreviewLabel.font = desiredFont
        self.captionPreviewLabel.textColor = NSColor(hexString:
self.episode.styleFontColor ?? "#ffffff")
        if self.episode.styleFontShadow != 0 {
            let shadow: NSShadow = NSShadow()
            shadow.shadowBlurRadius = 1.07
            shadow.shadowOffset = NSMakeSize(1, 1.5)
            shadow.shadowColor = NSColor.black.withAlphaComponent(0.75)
            self.captionPreviewLabel.shadow = shadow
        } else {
            self.captionPreviewLabel.shadow = nil
        }
        refreshFontRelativeSize()
    }
    var videoRect: CGRect {
        get {
            guard let item = self.playerView.player?.currentItem, let track =
item.asset.tracks(withMediaType: .video).first else {return CGRect.zero}
            let trackSize = track.naturalSize
            let videoViewSize = self.playerView.bounds.size
            let trackRatio = trackSize.width / trackSize.height
            let videoViewRatio = videoViewSize.width / videoViewSize.height
            var newSize: CGSize
            if (videoViewRatio > trackRatio) {
                newSize = CGSize(width: trackSize.width *
videoViewSize.height / trackSize.height, height: videoViewSize.height)
            } else {
                newSize = CGSize(width: videoViewSize.width, height:
trackSize.height * videoViewSize.width / trackSize.width);
            }
            let newX = (videoViewSize.width - newSize.width) / 2;
            let newY = (videoViewSize.height - newSize.height) / 2;
            return CGRect(x: newX, y: newY, width: newSize.width, height:
newSize.height)
        }
    }
    func refreshFontRelativeSize() {
        guard let asset = self.playerView.player?.currentItem?.asset else
{return}
        let videoFrame = videoRect
        if isAudioOnly {
            captionBottomConstraint.constant = 14
        } else {
            let margin = CGFloat(40)
            let diffHeights = (playerView.frame.height - videoFrame.height) /
2
```

```
            if diffHeights > 0 {
                captionBottomConstraint.constant = margin + 14 + diffHeights
            } else {
                captionBottomConstraint.constant = margin + 14
            }
        }
        let shrinkingPercentage = videoFrame.size.width /
asset.tracks[0].naturalSize.width
        let size = self.episode.styleFontSize ?? "53"
        let sizeFloat = CGFloat(Float(size) ?? 53)
        let shrunkFontSize = sizeFloat * shrinkingPercentage
        guard let oldFont = self.captionPreviewLabel.font else {return}
        let newFont = NSFont(descriptor: oldFont.fontDescriptor, size:
shrunkFontSize)
        self.captionPreviewLabel.font = newFont
    }
    var isAudioOnly: Bool {
        get {
            guard let asset = self.playerView.player?.currentItem?.asset else
{ return false }
            if asset.tracks(withMediaType: .video).count == 0 { return true }
            return false
        }
    }
    var isVideoOnly: Bool {
        get {
            guard let asset = self.playerView.player?.currentItem?.asset else
{ return false }
            if asset.tracks(withMediaType: .audio).count == 0 { return true }
            return false
        }
    }
    func populateThumbnail() {
        if (self.episode.thumbnailURL == nil) {
            if isAudioOnly { return }
            let sourceURL = self.episode!.videoURL!
            let asset = AVAsset(url: URL(fileURLWithPath: sourceURL))
            let imageGenerator = AVAssetImageGenerator(asset: asset)
            let time = CMTimeMake(value: 1, timescale: 1)
            do {
                let imageRef = try imageGenerator.copyCGImage(at: time,
actualTime: nil)
                let thumbnail = NSImage(cgImage: imageRef, size:
NSSize(width: imageRef.width, height: imageRef.height))
                _ =
self.applicationDataDirectory().appendingPathComponent("thumbnails")
                try FileManager.default.createDirectory(at:
self.applicationDataDirectory().appendingPathComponent("thumbnails"),
withIntermediateDirectories: true, attributes: nil)
                let destinationURL =
self.applicationDataDirectory().appendingPathComponent("thumbnails").appendin
gPathComponent("\(NSUUID().uuidString).png")
                let result = thumbnail.pngWrite(to: destinationURL)
                print("Writing thumbnail: \(result)")
                self.episode.thumbnailURL = destinationURL.path
            } catch {
                print(error)
            }
        }
    }
    func saveTXTToDisk(_ sender: Any) {
        Saver.saveEpisodeToDisk(self.episode, type: .txt)
    }
```

```
    func saveSRTToDisk(_ sender: Any) {
        Saver.saveEpisodeToDisk(self.episode, type: .srt)
    }

    func saveFCPXMLToDisk(_ sender: Any) {
        Saver.saveEpisodeToDisk(self.episode, type: .fcpXML)
    }
    func saveASSToDisk(_ sender: Any) {
        Saver.saveEpisodeToDisk(self.episode, type: .ass)
    }
    func dismantleSetTimelineLengthToZero() {
        self.timelineOverallView.setFrameSize(NSSize(width: 0, height:
self.timelineOverallView.frame.size.height))
    }
    func dismantleOldMovieVC() {
        self.timeLabel.stringValue = "00:00:00,000"
        welcomingLabel.stringValue = "Welcome to\nMovieTranscript"
        guidanceLabel.stringValue = "To create captions, click on the
button on the toolbar to open an existing video or audio file."
        dismantleSetTimelineLengthToZero()
        openFileOrRelinkButton.image = NSImage(named: "import")
        customHintContainerView.isHidden = false
        AppDelegate.mainWindow()?.shareResultsButton.isEnabled = false
        AppDelegate.setCurrentEpisodeTitle(nil)
        AppDelegate.mainWindow()?.relinkMode = false
        captionBottomConstraint.constant = 14
        self.captionPreviewLabel.stringValue = ""
        self.playerView.player?.safelyRemoveObserver(self, forKeyPath:
"status")
        volumeSlider.floatValue = 1
        speedSlider.floatValue = 1
        self.videoPreviewContainerView.guid = nil

        NotificationCenter.default.removeObserver(self)
        self.playerView.player?.safelyRemoveObserver(self, forKeyPath:
"rate")
        if (episode != nil) {
            self.episode.player?.safelyRemoveObserver(self, forKeyPath:
"rate")
            episode.safelyRemoveObserver(self, forKeyPath: "arrayForCaption")
            if let arr = self.episode.arrayForCaption?.array as?
[CaptionLine] {
                for line in arr {
                    line.safelyRemoveObserver(self, forKeyPath: "caption")
                    line.safelyRemoveObserver(self, forKeyPath:
"startingTime")
                    line.safelyRemoveObserver(self, forKeyPath: "endingTime")
                }
            }
            episode.safelyRemoveObserver(self, forKeyPath: "styleFontColor")
            episode.safelyRemoveObserver(self, forKeyPath: "styleFontFamily")
            episode.safelyRemoveObserver(self, forKeyPath: "styleFontShadow")
            episode.safelyRemoveObserver(self, forKeyPath: "styleFontSize")
            episode.safelyRemoveObserver(self, forKeyPath: "styleFontWeight")
        }
        for view in videoPreviewContainerView.subviews {
            view.removeFromSuperview()
        }
        for view in waveformPreviewContainerBox.contentView!.subviews {
            if (view.tag != -1) {
                view.removeFromSuperview()
            }
        }
        self.subtitleTrackContainerView.subviews = []
```

```
        self.videoPreviewContainerView.subviews = []
        self.progressView.setFrameOrigin(NSPoint(x: 0, y:
self.progressView.frame.origin.y))
    }
    func setupForRelinkVideo() {
        AppDelegate.mainWindow()?.relinkMode = true
        openFileOrRelinkButton.image = NSImage(named: "link")
        welcomingLabel.stringValue = "Relinking Is\nRequired"
        guidanceLabel.stringValue = "To relink your video, click on the
button on the toolbar. Then, you will be able to continue working on the
project."
    }
    func handleRelinkVideo() {
        Helper.displayInteractiveSheet(title: "Video deleted or moved", text:
"The video associated with this captioning project cannot be found. It may
have been deleted or moved. Please relink the video.", firstButtonText:
"Relink Video", secondButtonText: "Cancel") { (shouldRelink) in
            if (shouldRelink) {
                Helper.displayOpenFileDialog(callback: { (hasSelected,
fileURL, filePath) in
                    guard let newURL = fileURL else {return}
                    self.episode.videoURL = newURL.path
                    if FileManager.default.fileExists(atPath: newURL.path) {
                        self.playVideo(newURL)
                    } else {
                        Helper.displayInformationalSheet(title: "Relinking
failed", text: "Unable to relink to your newly selected media.")
                    }
                })
            }
        }
    }
    func configurateMovieVC() {
        if self.episode == nil {
            return
        }
        self.episode.modifiedDate = NSDate()
        NotificationCenter.default.addObserver(self, selector:
#selector(frameDidChangeNotification(_:)), name:
NSView.frameDidChangeNotification, object: self.playerView)
        if let url = self.episode.videoURL {
            if FileManager.default.fileExists(atPath: url) {
                self.playVideo(URL(fileURLWithPath: url))
            } else {
                setupForRelinkVideo()
            }
        } else {
            self.episode.player = AVPlayer()
            self.playerView.player = self.episode.player
            self.episode.player?.addObserver(self, forKeyPath: "rate",
options: [.new], context: &MovieViewController.playerPlayrateContext)
            print("Can't configurate movie VC")
        }
    }
    var pointsPerFrameScaleFactor: CGFloat = 45
    var timelineLengthPixels: CGFloat {
        get {
            let player = self.episode.player!
            return
CGFloat(CMTimeGetSeconds((player.currentItem?.asset.duration)!)) *
pointsPerFrameScaleFactor
        }
    }
    var timeLineSegmentHeight: CGFloat {
```

```
        return 64
    }
    // MARK: - Custom Timeline
    func configureOverallScrollView() {
        self.timelineOverallView.setFrameSize(NSSize(width:
timelineLengthPixels + 100, height:
self.timelineOverallView.frame.size.height))
    }
    private static var textTrackContext = 0
    private static var fontPreviewTrackContext = 1
    private static var playerPlayrateContext = 5
    private static var playerReadinessContext = 6
    func correspondingTimeAtEvent(_ event: NSEvent) -> Float {
        let location =
self.subtitleTrackContainerView.convert(event.locationInWindow, from: nil).x
        //        print(location)
        let percentage = Float(location / self.timelineLengthPixels)
        let timePoint = percentage * self.calculatedDuration
        return timePoint
    }
    let draggingMargin: Float = 0.25
    func correspondingCaptionAtLocation(timePoint: Float) -> (line1:
CaptionLine?, line2: CaptionLine?, cursorType: CursorType) {
        if let eparr = episode.arrayForCaption?.array as? [CaptionLine] {
            if eparr.count < 2 {
                return (nil, nil, .normal)
            }
            for i in 0..<eparr.count - 1 {
                let captionLine = eparr[i]
                let captionLineNext = eparr[i + 1]
                let diffStarting = timePoint - captionLine.startingTime
                let diffEnding = timePoint - captionLine.endingTime
                let diffThisNext = abs(captionLineNext.startingTime -
captionLine.endingTime)
                // let diffNextStartThisEnd = (captionLineNext.startingTime -
captionLine.endingTime)
                if diffThisNext < 0.1 && abs(diffEnding) < 0.1 {
                    return (captionLine, captionLineNext, .resizeLeftRight)
                } else if timePoint > captionLine.endingTime && timePoint <
captionLineNext.startingTime + draggingMargin && timePoint >=
captionLineNext.startingTime {
                    return (captionLineNext, nil, .resizeRight)
                } else if timePoint > captionLineNext.startingTime &&
abs(diffStarting) <= draggingMargin {
                    print(".resizeRight, diffStarting:\(diffStarting)")
                    return (captionLine, nil, .resizeRight)
                } else if abs(diffEnding) <= draggingMargin {
                    print(".resizeLeft, diffEnding:\(diffEnding)")
                    return (captionLine, nil, .resizeLeft)
                } else {
                    // return (captionLine, nil, .selection)
                    // print(".passing, diffStarting:\(diffStarting),
diffEnding: \(diffEnding)")
                }
            }
            for i in 0..<eparr.count {
                let captionLine = eparr[i]
                if timePoint > captionLine.startingTime && timePoint <
captionLine.startingTime + draggingMargin {
                    return (captionLine, nil, .resizeRight)
                } else if timePoint < captionLine.endingTime && timePoint >
captionLine.endingTime - draggingMargin {
                    return (captionLine, nil, .resizeLeft)
```

```
                } else if timePoint > captionLine.startingTime +
draggingMargin && timePoint < captionLine.endingTime - draggingMargin {
                    return (captionLine, nil, .moveTime)
                }
            }
        }
        return (nil, nil, .normal)
    }
    enum CursorType {
        case resizeLeftRight
        case resizeRight
        case resizeLeft
        case moveTime
        case normal
    }
    var cachedDownLine1: CaptionLine?
    var cachedDownLine2: CaptionLine?
    var cachedOperation: CursorType?
    func trackingMouseDown(with event: NSEvent) {
        let timePoint = correspondingTimeAtEvent(event)
        let cache = correspondingCaptionAtLocation(timePoint: timePoint)
        cachedDownLine1 = cache.line1
        cachedDownLine2 = cache.line2
        cachedOperation = cache.cursorType
    }
    let errorAvoidanceThreshold: Float = 0.3
    func restoreOriginalPointer() {
        self.view.window?.enableCursorRects()
        NSCursor.arrow.set()
    }
    var calculatedDuration: Float {
        get {
            return
Float(CMTimeGetSeconds((self.episode.player!.currentItem?.asset.duration)!))
        }
    }
    func addObserverForCaptionLine(_ captionLine: CaptionLine) {
        captionLine.addObserver(self, forKeyPath: "caption", options: [.new,
.initial], context: &MovieViewController.textTrackContext)
        captionLine.addObserver(self, forKeyPath: "startingTime", options:
[.new, .initial], context: &MovieViewController.textTrackContext)
        captionLine.addObserver(self, forKeyPath: "endingTime", options:
[.new, .initial], context: &MovieViewController.textTrackContext)
    }
    override func observeValue(forKeyPath keyPath: String?, of object: Any?,
change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {
        guard context == &MovieViewController.textTrackContext || context ==
&MovieViewController.fontPreviewTrackContext || context ==
&MovieViewController.playerPlayrateContext || context ==
&MovieViewController.playerReadinessContext else {
            super.observeValue(forKeyPath: keyPath, of: object, change:
change, context: context)
            return
        }
        if context == &MovieViewController.fontPreviewTrackContext {
            self.observeFontStyleChangedValue(forKeyPath: keyPath, of:
object, change: change)
        } else if context == &MovieViewController.playerPlayrateContext {
            if keyPath == "rate" {
                if let player = self.playerView.player {
                    if player.rate > 0 {
                        self.playPauseImageView.image = NSImage(named:
"tb_pause")
                    } else {
```

```
                        self.playPauseImageView.image = NSImage(named:
"tb_play")
                    }
                }
            }
        } else if context == &MovieViewController.playerReadinessContext {
            guard let player = self.playerView.player else {return}
            if player.status == .readyToPlay {
                self.updateLoadVideo()
                // episode.player?.play()
            }
        }
    }
    func observeFontStyleChangedValue(forKeyPath keyPath: String?, of object:
Any?, change: [NSKeyValueChangeKey : Any]?) {
        refreshFontToReflectStyleChanges()
    }
    var videoThumbnailTrackCacheRoot =
("~/Library/Caches/com.zho.MovieTranscript/video_thumbnail" as
NSString).expandingTildeInPath as String
    let thumbnailPerSeconds: Float64 = 2
    func configureVideoThumbnailTrack() {
        if isAudioOnly {
            return
        }
        let computedGUIDForTask = NSUUID().uuidString
        videoPreviewContainerView.guid = computedGUIDForTask
        self.videoPreviewContainerView.setFrameSize(NSSize(width:
timelineLengthPixels, height:
self.videoPreviewContainerView.frame.size.height))
        guard let asset = self.episode.player?.currentItem?.asset else
{return}
        let videoThumbnailTrackCacheFolder =
"\(videoThumbnailTrackCacheRoot)/\(self.episode.guidIdentifier ?? "unknown")"
        let urlForCreation = URL(fileURLWithPath:
videoThumbnailTrackCacheFolder, isDirectory: true)
        do { try FileManager.default.createDirectory(at: urlForCreation,
withIntermediateDirectories: true, attributes: nil) }catch { print(error) }
        guard let duration = self.episode.player?.currentItem?.asset.duration
else { return }
        let totalSeconds = CMTimeGetSeconds(duration)
        var secondIndex: Float64 = 1
        var imageIndex: Int = 0
        if (totalSeconds.isNaN) {
            return
        }
        let numberOfThumbnails = Int(totalSeconds / self.thumbnailPerSeconds)
        let widthOfThumbnail = self.timelineLengthPixels /
CGFloat(numberOfThumbnails)
        let imageGenerator = AVAssetImageGenerator(asset: asset)
        imageGenerator.maximumSize = CGSize(width: 320, height: 240);
        while (secondIndex < totalSeconds) {
            let screenshotTime = CMTime(seconds: Double(secondIndex),
preferredTimescale: 1)
            let capturedIndex = imageIndex
            let tentativePath =
"\(videoThumbnailTrackCacheFolder)/\(capturedIndex).jpg"
            if FileManager.default.fileExists(atPath: tentativePath) {
                let taskToLoadImage = DispatchWorkItem {
                    let loadedImage = NSImage.init(contentsOfFile:
tentativePath)
                    let taskToPlaceIntoView = DispatchWorkItem {
```

```
                        let imageView = VideoPreviewImageView(frame:
NSRect(x: widthOfThumbnail * CGFloat(capturedIndex), y: 0, width:
widthOfThumbnail, height: self.timeLineSegmentHeight))
                        imageView.imageScaling = .scaleProportionallyUpOrDown
                        imageView.imageFrameStyle = .grayBezel
                        imageView.image = loadedImage

self.videoPreviewContainerView.addSubImageView(capturedGUID:
computedGUIDForTask, imageView: imageView)
                    }

                    DispatchQueue.main.async(execute: taskToPlaceIntoView)
                }
                DispatchQueue.global(qos: .userInitiated).async(execute:
taskToLoadImage)
            } else {
                DispatchQueue.global(qos: .userInitiated).async {
                    do {

imageGenerator.generateCGImagesAsynchronously(forTimes: [NSValue(time:
screenshotTime)], completionHandler: { (time1, image, time2, result, error)
in
                            if result == .succeeded {
                                let image = NSImage(cgImage: image!, size:
NSSize(width: image!.width, height: image!.height))
                                let taskToSave = DispatchWorkItem {
                                    image.saveAsFile(with: .jpeg, withName:
tentativePath)
                                }
                                DispatchQueue.global(qos:
.userInitiated).async(execute: taskToSave)
                                let task = DispatchWorkItem {
                                    let imageView =
VideoPreviewImageView(frame: NSRect(x: widthOfThumbnail *
CGFloat(capturedIndex), y: 0, width: widthOfThumbnail, height:
self.timeLineSegmentHeight))
                                    imageView.imageScaling =
.scaleProportionallyUpOrDown
                                    imageView.imageFrameStyle = .grayBezel
                                    imageView.image = image

self.videoPreviewContainerView.addSubImageView(capturedGUID:
computedGUIDForTask, imageView: imageView)
                                }
                                //
self.accumulatedMainQueueTasks.append(task)
                                DispatchQueue.main.async(execute: task)
                            } else {
                                print("Failed with: \(String(describing:
error))")
                            }
                        })
                    } catch {print("Error: \(error)")}
                }
            }
            secondIndex += self.thumbnailPerSeconds
            imageIndex += 1
        }
    }
    let offsetPixelInScrollView: CGFloat = 8
    let redBarOffsetInScrollView: CGFloat = 8
    func configurateRedBar() {
        let interval = CMTime(value: 1, timescale: 30)
```

```
        self.episode.player?.addPeriodicTimeObserver(forInterval: interval,
queue: DispatchQueue.main, using: { (progressTime) in
            let seconds = CMTimeGetSeconds(progressTime)
            let timeCode = Helper.secondFloatToString(float: seconds)
            self.timeLabel.stringValue = timeCode
            //lets move the slider thumb
            if let duration = self.episode.player?.currentItem?.duration {
                let durationSeconds = CMTimeGetSeconds(duration)
                let percent = CGFloat(seconds / durationSeconds)
                let originPoint = NSPoint(x: self.timelineLengthPixels *
percent + self.offsetPixelInScrollView - self.redBarOffsetInScrollView, y:
self.progressView.frame.origin.y)
                if (originPoint.x.isNaN || originPoint.y.isNaN) {
                    return
                }
                self.progressView.setFrameOrigin(originPoint)
            }
            var matched: Bool = false
            self.episode.arrayForCaption?.enumerateObjects({ (element, index,
stop) in
                if let captionLine = element as? CaptionLine {
                    if let s = self.episode.player?.currentTime().seconds {
                        let sec = Float(s)
                        if sec > captionLine.startingTime && sec <
captionLine.endingTime {
                            self.captionPreviewLabel.stringValue =
captionLine.caption ?? ""
                            matched = true
                        }
                    }
                }
            })
            if matched == false {
                self.captionPreviewLabel.stringValue = ""
            }
            let rightHandSideInView = self.progressView.frame.origin.x >
self.timelineScrollView.contentView.bounds.origin.x +
self.timelineScrollView.contentView.bounds.size.width
            let leftHandSideInView = self.progressView.frame.origin.x <
self.timelineScrollView.contentView.bounds.origin.x
            if !self.timelineOverallView.scrollViewIsScrolling &&
(rightHandSideInView || leftHandSideInView) {
                #if DEBUG
                // print("The playhead is no longer in view.
self.timelineScrollView.bounds:\(self.timelineScrollView.bounds),
self.progressView.frame:\(self.progressView.frame),
self.timelineScrollView.contentView.bounds:
\(self.timelineScrollView.contentView.bounds)")
                #endif
                // Time to scroll to make the new timestamp visible!
                var targetFrame = self.progressView.frame
                if (self.progressBarPanGestureRecongnizer.state == .began ||
self.progressBarPanGestureRecongnizer.state == .changed) {
                    // We only scrub a little to make the playhead visible if
the user is manually dragging the playhead
                    targetFrame.size.width = 2 *
(self.progressView.frame.size.width)
                    targetFrame.origin.x -= self.offsetPixelInScrollView

self.timelineScrollView.contentView.scrollToVisible(targetFrame)
                } else {
                    if let stopTime =
self.timelineOverallView.lastEndManualScrollTime {
```

```
                        let differenceBetweenScrollerEndtimeAndCurrentTime =
Date().timeIntervalSince1970 - stopTime
                        if differenceBetweenScrollerEndtimeAndCurrentTime < 3
{
                            // Bailing auto-scrolling since user just
initiated manual scroll gesture. This gives them an opportunity to manually
change the playhead position.
                            return
                        } else {
                            // Once the 3 second grace period expires, we re-
enable auto scrolling.
                            self.timelineOverallView.lastEndManualScrollTime
= nil
                        }
                    }
                    // We scrub to populate more of the timeline when video
playback naturally led the playhead to progress
                    targetFrame.size.width =
self.timelineScrollView.frame.width
                    targetFrame.origin.x -= self.offsetPixelInScrollView
                    self.timelineScrollView.scroll(to: targetFrame.origin,
animationDuration: 0.15)
                }
            } else {
                 // print("The playhead is in view.
self.timelineScrollView.bounds:\(self.timelineScrollView.bounds),
self.progressView.frame:\(self.progressView.frame),
self.timelineScrollView.contentView.bounds:
\(self.timelineScrollView.contentView.bounds)")
            }
        })
    }
    @IBAction func clickedOnNewTimelineIndex(_ sender:
NSClickGestureRecognizer) {
        handleNewTimelineLocation(sender: sender)
    }
    var shouldResumePlayingAfterPanEnds = false
    @IBOutlet weak var progressBarPanGestureRecongnizer:
NSPanGestureRecognizer!
    @IBOutlet weak var timelineClickGestureRecongnizer:
NSClickGestureRecognizer!
    @IBAction func pannedToNewTimelineIndex(_ sender: NSPanGestureRecognizer)
{
        if sender.state == .began {
            if let player = playerView.player {
                if player.rate > 0 {
                    shouldResumePlayingAfterPanEnds = true
                    playerView.player?.pause()
                } else {
                    shouldResumePlayingAfterPanEnds = false
                }
            }
        } else if sender.state == .ended || sender.state == .cancelled ||
sender.state == .failed {
            if shouldResumePlayingAfterPanEnds {
                playerView.player?.play()
            }
        }
        if sender.state == .began || sender.state == .changed {
            progressViewColorLineBox.fillColor = NSColor.yellow
            progressViewColorLineBox.needsDisplay = true
        } else {
            progressViewColorLineBox.fillColor = NSColor.red
            progressViewColorLineBox.needsDisplay = true
```

```
        }
        handleNewTimelineLocation(sender: sender)
    }
    func handleNewTimelineLocation(sender: NSGestureRecognizer) {
        if (self.episode == nil || self.episode.player == nil ||
self.episode.player?.currentItem == nil) {
            return
        }
        let location = sender.location(in: timelineOverallView).x -
self.offsetPixelInScrollView
        let percent = location / self.timelineLengthPixels
        if let duration = self.episode.player?.currentItem?.duration {
            let totalSeconds = CMTimeGetSeconds(duration)
            let value = Float64(percent) * totalSeconds
            let timeScale =
self.episode.player?.currentItem?.asset.duration.timescale ?? 1
            let exactTime = CMTime(seconds: value, preferredTimescale:
timeScale)
            self.episode.player!.seek(to: exactTime, toleranceBefore:
CMTime.zero, toleranceAfter: CMTime.zero)
        }
    }
    // MARK: - Buttons in the middle HUD
    @IBAction func volumeChanged(_ sender: NSSlider) {
        self.playerView.player?.volume = sender.floatValue
    }
    @IBAction func speedChanged(_ sender: NSSlider) {
        self.playerView.player?.rate = sender.floatValue
    }
    @IBAction func rewindByOneFrame(_ sender: Any) {
        if let item = self.playerView.player?.currentItem {
            item.step(byCount: -1)
        }
    }
    @IBAction func rewindByFiveSeconds(_ sender: Any) {
        movePlayheadBySeconds(-5)
    }
    func movePlayheadBySeconds(_ seconds: Double) {
        if let _ = self.episode.player?.currentItem?.duration, let
currentTime = self.episode.player?.currentTime().seconds {
            let value = currentTime + seconds
            let timeScale =
self.episode.player?.currentItem?.asset.duration.timescale ?? 1
            let exactTime = CMTime(seconds: value, preferredTimescale:
timeScale)
            self.episode.player!.seek(to: exactTime, toleranceBefore:
CMTime.zero, toleranceAfter: CMTime.zero)
        }
    }
    @IBAction func playPauseClicked(_ sender: Any) {
        if let player = playerView.player {
            if player.rate > 0 {
                self.playerView.player?.pause()
            } else {
                self.playerView.player?.play()
            }
        }
    }
    @IBAction func forwardByFiveSeconds(_ sender: Any) {
        movePlayheadBySeconds(5)
    }
    @IBAction func forwardByOneFrame(_ sender: Any) {
        if let item = self.playerView.player?.currentItem {
            item.step(byCount: 1)
```

```
        }
    }
    // MARK: - Right HUD toolset
    @IBAction func revealInSubtitleEditor(_ sender: Any) {
    }
    @IBAction func joinTwoCaptions(_ sender: Any) {
    }
    @IBAction func deleteSelectedCaptions(_ sender: Any) {
    }
    @IBAction func trimRight(_ sender: Any) {
    }
    @IBAction func trimLeft(_ sender: Any) {
    }
    @IBAction func bladeInMiddle(_ sender: Any) {
    }
}
```

A.15 OpaqueNSScroller

```
//  OpaqueNSScroller.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 11/28/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
class OpaqueNSScroller: NSScroller {
    override func draw(_ dirtyRect: NSRect) {
        super.draw(dirtyRect)
        NSColor.clear.setFill()
        dirtyRect.fill()
        // whatever style you want here for knob if you want
        knobStyle = .dark
    }
}
```

A.16 CaptionWindowController

```
//
//  CaptionWindowController.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 2/16/19.
//  Copyright © 2024 MyOrganization. All rights reserved.
//
import Cocoa
class CaptionWindowController: NSWindowController, NSWindowDelegate {
    weak var splitViewController: MainSplitViewController!
    @IBOutlet weak var currentTitle: NSTextField!
    @IBOutlet weak var contentSettingsSegmentedControl: NSSegmentedControl!
    @IBOutlet weak var importToolbarItem: NSToolbarItem!
    @IBOutlet weak var importToolbarButton: NSButton!
    @IBOutlet weak var shareResultsButton: NSButton!
```

```
    var relinkMode: Bool = false {
        didSet {
            if relinkMode == true {
                importToolbarItem.label = "Relink"
                importToolbarButton.image = NSImage(named: "link")
                AppDelegate.openOrRelinkMenuItem()?.title = "Relink Video or
Audio..."
            } else {
                importToolbarItem.label = "Import"
                importToolbarButton.image = NSImage(named: "import")
                AppDelegate.openOrRelinkMenuItem()?.title = "Open Video or
Audio..."
            }
        }
    }
    required init?(coder: NSCoder) {
        super.init(coder: coder)
        shouldCascadeWindows = true
    }
    override func awakeFromNib() {
        super.awakeFromNib()
        shouldCascadeWindows = true
    }
    override func windowDidLoad() {
        super.windowDidLoad()
        window?.titleVisibility = .hidden
        self.window?.delegate = self
        shouldCascadeWindows = true
        if let svc = contentViewController as? MainSplitViewController {
            self.splitViewController = svc
        }
    }
    @IBAction func toggleSidebarList(_ sender: Any) {
        splitViewController.splitViewItems[0].isCollapsed =
!splitViewController.splitViewItems[0].isCollapsed
    }
    @IBAction func addNewProjectClicked(_ sender: Any) {
        AppDelegate.sourceListVC()?.addNewProject()
    }
    @IBAction func importVideoFootageClicked(_ sender: Any) {
        let movieVC = splitViewController.splitViewItems[1].viewController
as! MovieViewController
        movieVC.openFile(self)
    }
    @IBAction func shareResultsButtonClicked(_ sender: NSButton) {
        if let vc = storyboard?.instantiateController(withIdentifier:
"ExportCaptions") as? NSViewController {
            self.contentViewController?.present(vc, asPopoverRelativeTo:
sender.bounds, of: sender, preferredEdge: NSRectEdge.maxY, behavior:
.transient)
        }
    }
    @IBAction func contentSettingsSidebarClicked(_ sender:
NSSegmentedControl) {
        contentSettingsSidebarHandler(index: sender.selectedSegment)
    }
    func contentSettingsSidebarHandler(index: Int) {
        if index < 2 {
            if let tabView = AppDelegate.sideTabVC()?.tabView, let selected =
tabView.selectedTabViewItem {
                if tabView.indexOfTabViewItem(selected) == index {
                    // collapse sidebar
                    splitViewController.splitViewItems[2].isCollapsed =
!splitViewController.splitViewItems[2].isCollapsed
```

```
                    if (splitViewController.splitViewItems[2].isCollapsed) {
                        contentSettingsSegmentedControl.setSelected(false,
forSegment: 0)
                        contentSettingsSegmentedControl.setSelected(false,
forSegment: 1)
                    } else {
                        contentSettingsSegmentedControl.setSelected(true,
forSegment: index)
                    }
                } else {
                    AppDelegate.sideTabVC()?.tabView.selectTabViewItem(at:
index)
                    splitViewController.splitViewItems[2].isCollapsed = false
                    contentSettingsSegmentedControl.setSelected(true,
forSegment: index)
                }
            }
         }}}



//  SidebarViewController.swift
```

A.17 SidebarViewController

```
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 3/10/19.
//  Copyright © 2024 MyOrganization. All rights reserved.
//
import Cocoa
class SidebarViewController: NSViewController, NSTableViewDelegate,
NSTableViewDataSource {
    @IBOutlet weak var tableView: NSTableView!
    var episodeProjects: [EpisodeProject] = []
    @IBOutlet var contextMenu: NSMenu!
    @IBOutlet weak var visualEffectsView: NSVisualEffectView!
    @IBAction func duplicateClicked(_ sender: Any) {
    }
    @IBAction func deleteAtIndex(_ sender: Any) {
        if tableView.selectedRow >= 0 {
            deleteHelper(row: tableView.selectedRow)
        }
    }
    func deleteHelper(row: Int) {
        let project = episodeProjects[row]
        /*if project != nil*/ do {
            Helper.displayInteractiveSheet(title: "Delete Project", text:
"Are you sure you want to delete this project? This will remove all subtitles
associated with the video \(project.videoDescription ?? ""). While the video
itself won't be deleted from your disk, the deletion of this project and
associated subtitles is not recoverable.\n\nIf you would like to preseve a
copy of the associated subtitle, cancel the deletion, and export the subtitle
as a SRT or FCPXML tile.", firstButtonText: "Delete Project",
secondButtonText: "Cancel") { (result) in
                if result {
                    var newRow = row + 1
                    if newRow > self.episodeProjects.count - 1 {
```

```
                        newRow = row - 1
                    }
                    if newRow < 0 || self.episodeProjects.count <= 0 {
                        self.addNewProject()
                        newRow = 0
                    }
                    if let id = project.guidIdentifier {
                        Helper.removeFilesUnderURL(urlPath:
"~/Library/Caches/com.zho.MovieTranscript/audio_thumbnail/\(id)")
                        Helper.removeFilesUnderURL(urlPath:
"~/Library/Caches/com.zho.MovieTranscript/video_thumbnail/\(id)")
                    }
                    //        tableView.selectRowIndexes(IndexSet(integer:
newRow), byExtendingSelection: false)
                    if let index = self.episodeProjects.firstIndex(of:
project) {
                        self.episodeProjects.remove(at: index)
                        project.player?.pause()
                        project.player?.replaceCurrentItem(with: nil)
                        project.player = nil
                        AppDelegate.subtitleVC()?.dismantleSubtitleVC()
                        AppDelegate.movieVC()?.dismantleOldMovieVC()
                        AppDelegate.movieVC()?.configurateMovieVC()
                        AppDelegate.subtitleVC()?.configurateSubtitleVC()
                        AppDelegate.fontVC()?.dismantleOldFontVC()
                        AppDelegate.fontVC()?.configurateFontVC()
                    }
                    Helper.context?.delete(project)
                    self.tableView.reloadData()
                    self.updateSelectRow(index: self.tableView.selectedRow)
                }
            }
        }
    }
    @IBAction func deleteClicked(sender: Any) {
        let row = tableView.clickedRow
        deleteHelper(row: row)
    }
    static func removeFilesUnderURL(urlPath: String) {
        let cacheURL = (urlPath as NSString).expandingTildeInPath as String
        guard URL(string: cacheURL) != nil else {return}
        let fileManager = FileManager.default
        do {
            try fileManager.removeItem(atPath: cacheURL)
        } catch {
            print(error)
        }
    }
    @IBAction func exportFCPXMLClicked(_ sender: Any) {
        let episode = episodeProjects[tableView.clickedRow]
        Saver.saveEpisodeToDisk(episode, type: .fcpXML)
    }
    @IBAction func exportSRTClicked(_ sender: Any) {
        let episode = episodeProjects[tableView.clickedRow]
        Saver.saveEpisodeToDisk(episode, type: .srt)
    }
    @IBAction func exportASSClicked(_ sender: Any) {
        let episode = episodeProjects[tableView.clickedRow]
        Saver.saveEpisodeToDisk(episode, type: .ass)
    }
    @IBAction func exportTXTClicked(_ sender: Any) {
        let episode = episodeProjects[tableView.clickedRow]
        Saver.saveEpisodeToDisk(episode, type: .txt)
    }
```

```
    func selectActiveContextMenuRow() {
        if tableView.selectedRow != tableView.clickedRow {
            updateSelectRow(index: tableView.clickedRow)
        }
    }
    @IBAction func showInFinderClicked(_ sender: Any) {
        showEpisodeInFinder(row: tableView.clickedRow)
    }
    func showActiveVideoInFinderClicked(_ sender: Any) {
        showEpisodeInFinder(row: tableView.selectedRow)
    }
    func showEpisodeInFinder(row: Int) {
        let episode = episodeProjects[row]
        if let url = episode.videoURL {

NSWorkspace.shared.activateFileViewerSelecting([URL(fileURLWithPath: url)])
        }
    }
    @IBOutlet weak var duplicateClicked: NSMenuItem!
    override func viewDidLoad() {
        super.viewDidLoad()
        if #available(OSX 10.14, *) {
            self.visualEffectsView.material = .sidebar
        } else {
            self.visualEffectsView.material = .light
        }
        fetchDBData()
        tableView.delegate = self
        tableView.dataSource = self
        tableView.allowsEmptySelection = false
        tableView.selectionHighlightStyle = .sourceList
    }
    func fetchDBData() {
        do {
            let fetchRequest: NSFetchRequest<EpisodeProject> =
EpisodeProject.fetchRequest()
            fetchRequest.sortDescriptors = [NSSortDescriptor(key:
"modifiedDate", ascending: false)]
            episodeProjects = try Helper.context!.fetch(fetchRequest)
        } catch {
            print("Can't fetch persistence store with: \(error)")
        }
        for episode in episodeProjects {
            if episode.guidIdentifier == nil {
                episode.guidIdentifier = NSUUID().uuidString
            }
            episode.addObserver(self, forKeyPath: "videoURL", options:
[.new], context: nil)
            episode.addObserver(self, forKeyPath: "thumbnailURL", options:
[.new], context: nil)
            episode.addObserver(self, forKeyPath: "videoDescription",
options: [.new], context: nil)
        }
        #if DEBUG
        print("Fetch DB data has succeeded.")
        #endif
    }
    override func observeValue(forKeyPath keyPath: String?, of object: Any?,
change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {
        if let project = object as? EpisodeProject, let index =
episodeProjects.firstIndex(of: project) {
            tableView.reloadData(forRowIndexes: IndexSet(integer: index),
columnIndexes: IndexSet(integer: 0))
        }
```

```
    }
    var firstLaunch = true
    override func viewDidAppear() {
        if (episodeProjects.count == 0) {
            addNewProject()
        }
        if firstLaunch {
            #if DEBUG
            print("Setting selected row to the first row on launch.")
            #endif
            updateSelectRow(index: 0)
            firstLaunch = false
        }
    }
    func addNewProject() {
        let description = NSEntityDescription.entity(forEntityName:
"EpisodeProject", in: Helper.context!)
        let episode = EpisodeProject(entity: description!, insertInto:
Helper.context!)
        episode.guidIdentifier = NSUUID().uuidString
        episode.creationDate = NSDate()
        episode.modifiedDate = NSDate()
        episode.styleFontShadow = 1
        episodeProjects.append(episode)
        fetchDBData()
        tableView.insertRows(at: IndexSet(integer: 0), withAnimation:
.slideDown)
        tableView.reloadData(forRowIndexes: IndexSet(integer: 0),
columnIndexes: IndexSet(integer: 0))
    }
    func tableView(_ tableView: NSTableView, viewFor tableColumn:
NSTableColumn?, row: Int) -> NSView? {
        if let view = tableView.makeView(withIdentifier:
NSUserInterfaceItemIdentifier(rawValue: "SidebarEpisodeTableCellView"),
owner: self) as? SidebarEpisodeTableCellView {
            let episode = episodeProjects[row]
            view.videoFileNameTextField.stringValue =
episode.videoDescription ?? "New Project"
            let formatter = DateFormatter.init()
            formatter.dateFormat = "MMM dd, yyyy"
            if let date = episode.modifiedDate as Date? {
                view.lastModifiedDateTextField.stringValue =
formatter.string(from: date)
            }
            if let url = episode.thumbnailURL, let image =
NSImage(contentsOf: URL(fileURLWithPath: url)) {
                view.episodePreview?.image = image
            } else {
                view.episodePreview?.image = NSImage(named: "bunny")
            }
            return view
        }
        return nil
    }
    func tableViewSelectionDidChange(_ notification: Notification) {
        updateSelectRow(index: tableView.selectedRow)
    }
    func updateSelectRow(index: Int) {
        Helper.appWindow().makeKeyAndOrderFront(self)
        #if DEBUG
        print("The state of various VCs: \(String(describing:
AppDelegate.movieVC())), \(String(describing: AppDelegate.subtitleVC())),
\(String(describing: AppDelegate.fontVC()))")
        #endif
```

```
        if (AppDelegate.movieVC() == nil || AppDelegate.subtitleVC() == nil
|| AppDelegate.fontVC() == nil) {
            Timer.scheduledTimer(withTimeInterval: 0.15, repeats: false) {
(timer) in
                #if DEBUG
                print("We're giving loading the row a second try after the
timer.")
                self.updateSelectRow(index: index)
                #endif
            }
            return
        }
        tableView.selectRowIndexes(IndexSet(integer: index),
byExtendingSelection: false)
        if index < 0 {
            return
        }
        for project in episodeProjects {
            if let player = project.player {
                player.pause()
            }
        }
        AppDelegate.subtitleVC()?.dismantleSubtitleVC()
        let project = episodeProjects[index]
        AppDelegate.movieVC()?.dismantleOldMovieVC()
        AppDelegate.movieVC()?.episode = project
        AppDelegate.movieVC()?.configurateMovieVC()
        episodeProjects.swapAt(index, 0)
        tableView.moveRow(at: index, to: 0)
        tableView.reloadData(forRowIndexes: IndexSet(integersIn:
0..<episodeProjects.count), columnIndexes: IndexSet(integer: 0))
        AppDelegate.subtitleVC()?.episode = project
        AppDelegate.subtitleVC()?.configurateSubtitleVC()
        AppDelegate.fontVC()?.dismantleOldFontVC()
        AppDelegate.fontVC()?.episode = project
        AppDelegate.fontVC()?.configurateFontVC()
        #if DEBUG
        print("Selected: \(project)")
        #endif
    }
    func sortLocalArrayCache() {
        episodeProjects.sort { (ep1, ep2) -> Bool in
            if let m1 = ep1.modifiedDate, let m2 = ep2.modifiedDate {
                return m1.compare(m2 as Date) ==
ComparisonResult.orderedDescending
            }
            return false
        }
    }
    func tableView(_ tableView: NSTableView, shouldSelectRow row: Int) ->
Bool {
        if row < 0 {
            return false
        }
        return true
    }
    func numberOfRows(in tableView: NSTableView) -> Int {
        return episodeProjects.count
    }
    func tableView(_ tableView: NSTableView, heightOfRow row: Int) -> CGFloat
{
        return 66
    }
}
```

A.18 SubtitlesViewController

```
//  SubtitlesViewController.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 11/26/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
import AVKit
@objc class SubtitlesViewController: NSViewController, NSTableViewDelegate,
NSTableViewDataSource, NSTextFieldDelegate,
CaptionDetailTableCellViewDelegate {
    @IBOutlet weak var transcribeTextField: NSTextField!
    @IBOutlet weak var resultTableView: NSTableView!
    weak var episode: EpisodeProject! {
        didSet {
            if episode != nil && episode.arrayForCaption != nil {
                episode.arrayForCaption?.enumerateObjects({ (obj, index,
stop) in
                    if let line = obj as? CaptionLine {
                        self.addObserverForCaptionLine(line: line)
                    }
                })
            }}
    }
    @IBAction func deleteRow(_ sender: NSMenuItem) {
        let row = resultTableView.clickedRow
        if let correspondingCaption = episode.arrayForCaption?.object(at:
row) as? CaptionLine {
            episode.removeFromArrayForCaption(correspondingCaption)
            Helper.context?.delete(correspondingCaption)
            resultTableView.reloadData()
        }
    }
    func addObserverForCaptionLine(line: CaptionLine) {
        line.addObserver(self, forKeyPath: "caption", options: [.new],
context: nil)
        line.addObserver(self, forKeyPath: "endingTime", options: [.new],
context: nil)
        line.addObserver(self, forKeyPath: "startingTime", options: [.new],
context: nil)
    }
    override func observeValue(forKeyPath keyPath: String?, of object: Any?,
change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {
        var matchedIndex: Int?
        if let captionLine = object as? CaptionLine {
            episode.arrayForCaption?.enumerateObjects({ (obj, index, stop) in
                if let captionMatch = obj as? CaptionLine {
                    if captionLine == captionMatch {
                        matchedIndex = index
                        stop.pointee = true
                    }
                }
            })
        }
        if matchedIndex != nil {
            self.resultTableView.reloadData()
//            self.resultTableView.reloadData(forRowIndexes:
IndexSet(integer: index), columnIndexes: IndexSet(integer: 0))
        }
    }
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
    }
    func dismantleSubtitleVC() {
        self.episode = nil
        self.resultTableView.reloadData()
    }
    func configurateSubtitleVC() {
        self.resultTableView.delegate = self
        self.resultTableView.dataSource = self
        self.transcribeTextField.delegate = self
        self.resultTableView.reloadData()
        self.transcribeTextField.stringValue = ""
    }
    // MARK: - TextField Controls
    func control(_ control: NSControl, textShouldBeginEditing fieldEditor:
NSText) -> Bool {
        if (episode.player == nil) {
            return true
        }
        if (control == transcribeTextField) {
            episode.player?.pause()
            if let last = episode.arrayForCaption?.lastObject as?
CaptionLine, let time = episode.player?.currentTime() {
                let currentTimeFloat = Float(CMTimeGetSeconds(time))
                if (currentTimeFloat < last.startingTime) {
                    last.startingTime = (currentTimeFloat - 2) > 0 ?
currentTimeFloat - 2 : 0
                    last.endingTime = currentTimeFloat
                } else {
                    let timeDelta = currentTimeFloat - last.startingTime
                    if timeDelta > 7 {
                        last.startingTime = (currentTimeFloat - 2) > 0 ?
currentTimeFloat - 2 : 0
                    }
                    last.endingTime = Float(CMTimeGetSeconds(time))
                }
            }
        }
        return true
    }
    @IBAction func captionTextFieldDidChange(_ sender: NSTextField) {
        if (episode == nil || episode.player == nil) {
            sender.stringValue = ""
            Helper.displayInformationalSheet(title: "A video is required
before adding captions.", text: "Please open a video first, then add captions
to the video.")
            return
        }
        if (sender == transcribeTextField) {
            episode.player?.play()
            if sender.stringValue == "" {
                if let last = episode.arrayForCaption?.lastObject as?
CaptionLine, let time = episode.player?.currentTime() {
                    last.startingTime = Float(CMTimeGetSeconds(time))
                }
            } else {
                if let last = episode.arrayForCaption?.lastObject as?
CaptionLine {
                    last.caption = sender.stringValue
                }
                sender.stringValue = ""
                var new: CaptionLine!
```

```
                if let lastEndingTime = (episode.arrayForCaption?.lastObject
as? CaptionLine)?.endingTime {
                    let description =
NSEntityDescription.entity(forEntityName: "CaptionLine", in: Helper.context!)
                    new = CaptionLine(entity: description!, insertInto:
Helper.context!)
                    new.guidIdentifier = NSUUID().uuidString
                    new.caption = ""
                    new.startingTime = lastEndingTime
                    new.endingTime = lastEndingTime // this is likely wrong,
although better than 0 for integrity
                    self.addObserverForCaptionLine(line: new)
                } else {
                    let description =
NSEntityDescription.entity(forEntityName: "CaptionLine", in: Helper.context!)
                    new = CaptionLine(entity: description!, insertInto:
Helper.context!)
                    new.guidIdentifier = NSUUID().uuidString
                    new.caption = ""
                    new.startingTime =
Float(CMTimeGetSeconds((episode.player?.currentTime())!))
                    new.endingTime = new.startingTime // this is likely
wrong, although better than 0 for integrity
                    self.addObserverForCaptionLine(line: new)
                }
                episode.addToArrayForCaption(new)
            }
            self.resultTableView.reloadData()
            if resultTableView.numberOfRows > 0 {

self.resultTableView.scrollRowToVisible(resultTableView.numberOfRows - 1)
            }
        }
    }
    // MARK: - Table View Delegate/Data Source
    func numberOfRows(in tableView: NSTableView) -> Int {
        if episode == nil || episode.arrayForCaption == nil {
            return 0
        }
        return episode.arrayForCaption?.count ?? 0
    }
    func tableView(_ tableView: NSTableView, viewFor tableColumn:
NSTableColumn?, row: Int) -> NSView? {
        if let cell = tableView.makeView(withIdentifier:
NSUserInterfaceItemIdentifier(rawValue: "CaptionCell"), owner: self) as?
CaptionDetailTableCellView {
            let correspondingCaption = episode.arrayForCaption?.object(at:
row) as? CaptionLine
            cell.startTimeField?.stringValue =
"\(correspondingCaption!.startingTimeString)"
            cell.endTimeField?.stringValue =
"\(correspondingCaption!.endingTimeString)"
            if let cap = correspondingCaption!.caption {
                cell.captionContentTextField?.stringValue = cap
            } else {
                cell.captionContentTextField?.stringValue = ""
            }
            cell.captionContentTextField?.isEditable = true
            cell.delegate = self
            return cell
        }
        return nil
    }
    func tableView(_ tableView: NSTableView, heightOfRow row: Int) -> CGFloat
{
        return 71
```

```
    }
    @IBAction func captionChanged(_ sender: NSTextField) {
        let row = resultTableView.row(for: sender)
        if let ep = episode.arrayForCaption?.object(at: row) as? CaptionLine
{
            ep.caption = sender.stringValue
        }
    }
    func textFieldEdited(_ sender: Any) {
        if let cell = sender as? CaptionDetailTableCellView {
            let rowIndex = resultTableView.row(for: cell)
            if let cl = episode.arrayForCaption?.object(at: rowIndex) as?
CaptionLine {
                cl.caption = cell.captionContentTextField.stringValue
            }
        }
    }
    @IBAction func doneButtonClicked(_ sender: Any) {
        self.resignFirstResponder()
        self.transcribeTextField.resignFirstResponder()
        NSApp.mainWindow?.makeFirstResponder(AppDelegate.movieVC())
    }
}
```

A.19 CaptionDetailTableCellView

```
//
//  CaptionDetailTableCellView.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 11/20/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
class CaptionDetailTableCellView: NSTableCellView {
    @IBOutlet weak var startTimeField: NSTextField!
    @IBOutlet weak var endTimeField: NSTextField!
    @IBOutlet var captionContentTextField: NSTextField!
    var delegate: CaptionDetailTableCellViewDelegate?
    @IBAction func textFieldEdited(_ sender: Any) {
        delegate?.textFieldEdited(self)
    }
}
protocol CaptionDetailTableCellViewDelegate {
    func textFieldEdited(_ sender: Any)
}
```

A.20 FontViewController

```
//  FontViewController.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 11/20/24.
//  Copyright © 2024 zho. All rights reserved.
```

```
//
import Cocoa
class FontViewController: NSViewController {
    weak var episode: EpisodeProject?
    @IBOutlet weak var videoName: NSTextField!
    @IBOutlet weak var videoPath: NSTextField!
    @IBOutlet weak var videoDurationField: NSTextField!
    @IBOutlet weak var videoFramerateField: NSTextField!
    @IBOutlet weak var fontFamilyButton: NSPopUpButton!
    @IBOutlet weak var fontWeightButton: NSPopUpButton!
    @IBOutlet weak var fontSizeButton: NSComboBox!
    @IBOutlet weak var fontShadowButton: NSPopUpButton!
    var fontColorButton: Any?
    @IBOutlet weak var colorContainerView: NSView!
    let allFontNames = NSFontManager.shared.availableFontFamilies
    var fontPostScriptArray: [String] = []
    let allFontSizes = [24, 36, 48, 53, 64, 72, 96]
    override func viewDidLoad() {
        super.viewDidLoad()
        if #available(OSX 10.12, *) {
            let button = ComboColorWell(frame: NSRect(x: 0, y: 0, width: 80,
height: 23))
            fontColorButton = button
            button.target = self
            button.action = #selector(colorChanged(_:))
            button.allowClearColor = false
            self.colorContainerView.addSubview(button)
        } else {
            let well = NSColorWell(frame: NSRect(x: 0, y: 0, width: 80,
height: 23))
            fontColorButton = well
            well.target = self
            well.action = #selector(colorChanged(_:))
            self.colorContainerView.addSubview(well)
        }
        configurateAllFonts()
    }
    func dismantleOldFontVC() {
        if episode != nil {
            episode?.safelyRemoveObserver(self, forKeyPath:
"videoDescription")
            episode?.safelyRemoveObserver(self, forKeyPath: "videoURL")
            episode?.safelyRemoveObserver(self, forKeyPath: "videoDuration")
            episode?.safelyRemoveObserver(self, forKeyPath: "framerate")
        }
    }
    func configurateFontVC() {
        restoreFontSettings()
        configureAllMetadata()
    }
    func configureAllMetadata() {
        if let up = episode?.videoURL {
            let url = URL(fileURLWithPath: up)
            self.videoName.stringValue = url.lastPathComponent
            self.videoPath.stringValue = url.path
        } else {
            self.videoName.stringValue = ""
            self.videoPath.stringValue = ""
        }
        self.videoDurationField.stringValue = "\(episode?.videoDuration ?? 0)
seconds"
        self.videoFramerateField.stringValue = "\(episode?.framerate ?? 0)
fps"
    }
```

```
    private static var metadataObserverContext = 2
    func configurateAllFonts() {
        fontFamilyButton.removeAllItems()
        fontFamilyButton.addItems(withTitles: allFontNames)
        updateSubFamily(saveNewSelection: false)
        fontSizeButton.removeAllItems()
        fontSizeButton.addItems(withObjectValues: allFontSizes)
        fontSizeButton.selectItem(at: 3)
    }
    func updateSubFamily(saveNewSelection: Bool) {
        fontWeightButton.removeAllItems()
        if let selectedFamily = fontFamilyButton.titleOfSelectedItem {
            if let arrayofSubs =
NSFontManager.shared.availableMembers(ofFontFamily: selectedFamily)  {
                var resultingSub:[String] = []
                fontPostScriptArray = []
                for i in 0..<arrayofSubs.count {
                    if let nameOfSubFamily = arrayofSubs[i][1] as? String {
                        resultingSub.append(nameOfSubFamily)
                    }
                    if let nameOfPostScript = arrayofSubs[i][0] as? String {
                        fontPostScriptArray.append(nameOfPostScript)
                    }
                }
                fontWeightButton.addItems(withTitles: resultingSub)
            }
        }
        if saveNewSelection {
            self.episode?.styleFontWeight = fontWeightButton.title
        }
    }
    func restoreFontSettings() {
        fontFamilyButton.selectItem(withTitle: self.episode?.styleFontFamily
?? "Helvetica")
        updateSubFamily(saveNewSelection: false)
        fontWeightButton.selectItem(withTitle: self.episode?.styleFontWeight
?? "Regular")
        fontSizeButton.stringValue = self.episode?.styleFontSize ?? "53"
        fontShadowButton.selectItem(at: Int(self.episode?.styleFontShadow ??
1))
        if #available(OSX 10.12, *) {
            if let well = fontColorButton as? ComboColorWell {
                well.color = NSColor(hexString: self.episode?.styleFontColor
?? "#ffffff") ?? NSColor.white
            }
        } else {
            if let well = fontColorButton as? NSColorWell {
                well.color = NSColor(hexString: self.episode?.styleFontColor
?? "#ffffff") ?? NSColor.white
            }
        }
    }
    @IBAction func fontNameChanged(_ sender: NSPopUpButton) {
        updateSubFamily(saveNewSelection: true)
        self.episode?.styleFontFamily = sender.title
    }
    @IBAction func fontSubFamilyChanged(_ sender: NSPopUpButton) {
        self.episode?.styleFontWeight = sender.title
    }
    @IBAction func fontSizeChanged(_ sender: NSComboBox) {
        self.episode?.styleFontSize = sender.stringValue
    }
    @IBAction func shadowChanged(_ sender: NSPopUpButton) {
        self.episode?.styleFontShadow = Int16(sender.indexOfSelectedItem)
```

```
    }
    @IBAction func colorChanged(_ sender: Any) {
        if #available(OSX 10.12, *) {
            if let well = sender as? ComboColorWell {
                self.episode?.styleFontColor = well.color.hexString
            }
        } else {
            if let well = sender as? NSColorWell {
                self.episode?.styleFontColor = well.color.hexString
            }
        }
    }
}
```

A.21 ResizableInputContainerView

```
//  ResizableInputContainerView.swift
//  MovieTranscript
//
//  Created by Oleksandra Zhyrova on 11/20/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
class ResizableInputContainerView: NSView {
    var hasConfigured: Bool = false
    override func draw(_ dirtyRect: NSRect) {
        super.draw(dirtyRect)
        if !hasConfigured {
            configure()
        }
    }
    func configure() {
        for view in self.subviews {
            if view.tag != 1 && self.frame.width > 140 {
                view.setFrameSize(NSSize(width: self.frame.width - 130,
height: view.frame.height))
            }
        }
    }}
```

A.22 AppDelegate

```
//
//  AppDelegate.swift
//
//
//  Created by Oleksandra Zhyrova on 08/29/24.
//  Copyright © 2024 zho. All rights reserved.
//
import Cocoa
@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {
    @IBOutlet weak var darkMenuItem: NSMenuItem!
```

```
    @IBOutlet weak var lightMenuItem: NSMenuItem!
    @IBOutlet weak var followSystemMenuItem: NSMenuItem!
    @IBOutlet weak var openOrRelinkMenuItem: NSMenuItem!
    static func mainStoryboard() -> NSStoryboard? {
        if #available(OSX 10.13, *) {
            return NSStoryboard.main
        } else {
            return NSStoryboard(name: "Main", bundle: nil)
        }
    }
    static func openOrRelinkMenuItem() -> NSMenuItem? {
        if let delegate = NSApp.delegate as? AppDelegate {
            return delegate.openOrRelinkMenuItem
        }
        return nil
    }
    static func setCurrentEpisodeTitle(_ title: String?) {
        if let window = NSApp.mainWindow?.windowController as?
CaptionWindowController {
            window.currentTitle.stringValue = title ?? "MovieTranscript"
        }
    }
    static func mainWindow() -> CaptionWindowController? {
        return NSApp.mainWindow?.windowController as? CaptionWindowController
    }
    static func sourceListVC() -> SidebarViewController? {
        if let splitVC = NSApp.mainWindow?.contentViewController as?
MainSplitViewController, let sourceListVC =
splitVC.splitViewItems[0].viewController as? SidebarViewController {
            return sourceListVC
        }
        return nil
    }
    static func movieVC() -> MovieViewController? {
        if let splitVC = NSApp.mainWindow?.contentViewController as?
MainSplitViewController, let movieVC =
splitVC.splitViewItems[1].viewController as? MovieViewController {
            return movieVC
        }
        return nil
    }
    static func sideTabVC() -> SideTabViewController? {
        if let splitVC = NSApp.mainWindow?.contentViewController as?
MainSplitViewController, let tabVC = splitVC.splitViewItems[2].viewController
as? SideTabViewController {
            return tabVC
        }
        return nil
    }
    static func subtitleVC() -> SubtitlesViewController? {
        if let tabVC = AppDelegate.sideTabVC(), let subVC =
tabVC.tabViewItems[0].viewController as? SubtitlesViewController {
            return subVC
        }
        return nil
    }
    static func fontVC() -> FontViewController? {
        if let tabVC = AppDelegate.sideTabVC(), let fontVC =
tabVC.tabViewItems[1].viewController as? FontViewController {
            return fontVC
        }
        return nil
    }
    @IBOutlet weak var dividerAboveAppearance: NSMenuItem!
```

```
    @IBOutlet weak var appearanceSubmenu: NSMenuItem!
    func applicationWillFinishLaunching(_ notification: Notification) {
        if #available(OSX 10.14, *) {
            if let pref = UserDefaults.standard.value(forKey:
"AppearancePreference") as? Int {
                if pref == 0 {
                    useDarkAppearance(self)
                } else if pref == 1 {
                    useLightAppearance(self)
                } else if pref == 2 {
                    followSystemAppearance(self)
                }
            } else {
                useDarkAppearance(self)
            }
        } else {
            dividerAboveAppearance.isHidden = true
            appearanceSubmenu.isHidden = true
        }
    }
    @objc func saveAll() {
        self.saveEverything(self)
    }
    func applicationDidFinishLaunching(_ aNotification: Notification) {
        Timer.scheduledTimer(timeInterval: 30, target: self, selector:
#selector(saveAll), userInfo: nil, repeats: true)
    }
    func applicationWillTerminate(_ aNotification: Notification) {
        do {
            let fetchRequest: NSFetchRequest<EpisodeProject> =
EpisodeProject.fetchRequest()
            let episodeProjects = try Helper.context!.fetch(fetchRequest)
            for proj in episodeProjects {
                if proj.videoURL == nil {
                    Helper.context!.delete(proj)
                }
            }
        } catch {
            print(error)
        }
        saveEverything(self)
    }

    func applicationShouldHandleReopen(_ sender: NSApplication,
hasVisibleWindows flag: Bool) -> Bool {
        if flag == false {

            for window in sender.windows {
                if (window.delegate?.isKind(of:
CaptionWindowController.self)) == true {
                    window.makeKeyAndOrderFront(self)
                }
            }
        }

        return true
    }

    func applicationShouldTerminateAfterLastWindowClosed(_ sender:
NSApplication) -> Bool {
        return true
    }
    @IBAction func openNewWindow(_ sender: Any) {
        AppDelegate.sourceListVC()?.addNewProject()
```

```
    }
    @IBAction func openVideoFile(_ sender: NSMenuItem) {
        AppDelegate.movieVC()?.openFile(self)
    }

    @IBAction func saveSRTFile(_ sender: NSMenuItem) {
        AppDelegate.movieVC()?.saveSRTToDisk(self)
    }

    @IBAction func saveTXTFile(_ sender: Any) {
        AppDelegate.movieVC()?.saveTXTToDisk(self)
    }
    @IBAction func saveFCPXMLFile(_ sender: Any) {
        AppDelegate.movieVC()?.saveFCPXMLToDisk(self)
    }
    @IBAction func saveASSFile(_ sender: Any) {
        AppDelegate.movieVC()?.saveASSToDisk(self)
    }
    @IBAction func saveEverything(_ sender: Any) {
        saveAction(nil)
    }
    @IBAction func useDarkAppearance(_ sender: Any) {
        if #available(OSX 10.14, *) {
            NSApp.appearance = NSAppearance(named: .darkAqua)
            UserDefaults.standard.setValue(0, forKey: "AppearancePreference")
            darkMenuItem.state = .on
            lightMenuItem.state = .off
            followSystemMenuItem.state = .off
        }
    }
    @IBAction func useLightAppearance(_ sender: Any) {
        if #available(OSX 10.14, *) {
            NSApp.appearance = NSAppearance(named: .aqua)
            UserDefaults.standard.setValue(1, forKey: "AppearancePreference")
            darkMenuItem.state = .off
            lightMenuItem.state = .on
            followSystemMenuItem.state = .off
        }
    }
    @IBAction func followSystemAppearance(_ sender: Any) {
        if #available(OSX 10.14, *) {
            NSApp.appearance = nil
            UserDefaults.standard.setValue(2, forKey: "AppearancePreference")
            darkMenuItem.state = .off
            lightMenuItem.state = .off
            followSystemMenuItem.state = .on
        }
    }
    // MARK: - Core Data stack
    lazy var applicationDocumentsDirectory: URL = {
        // The directory the application uses to store the Core Data store
file. This code uses a directory named "Test.OldCoreData" in the user's
Application Support directory.
        let urls = FileManager.default.urls(for:
.applicationSupportDirectory, in: .userDomainMask)
        let appSupportURL = urls[urls.count - 1]
        return appSupportURL.appendingPathComponent("Caption")
    }()
    lazy var managedObjectModel: NSManagedObjectModel = {
        // The managed object model for the application. This property is not
optional. It is a fatal error for the application not to be able to find and
load its model.
        let modelURL = Bundle.main.url(forResource: "Caption", withExtension:
"momd")!
```

```
        return NSManagedObjectModel(contentsOf: modelURL)!
    }()
    lazy var persistentStoreCoordinator: NSPersistentStoreCoordinator = {
        // The persistent store coordinator for the application. This
implementation creates and returns a coordinator, having added the store for
the application to it. (The directory for the store is created, if
necessary.) This property is optional since there are legitimate error
conditions that could cause the creation of the store to fail.
        let fileManager = FileManager.default
        var failError: NSError? = nil
        var shouldFail = false
        var failureReason = "There was an error creating or loading the
application's saved data."
        // Make sure the application files directory is there
        do {
            let properties = try (self.applicationDocumentsDirectory as
NSURL).resourceValues(forKeys: [URLResourceKey.isDirectoryKey])
            if !(properties[URLResourceKey.isDirectoryKey]! as
AnyObject).boolValue {
                failureReason = "Expected a folder to store application data,
found a file \(self.applicationDocumentsDirectory.path)."
                shouldFail = true
            }
        } catch  {
            let nserror = error as NSError
            if nserror.code == NSFileReadNoSuchFileError {
                do {
                    try fileManager.createDirectory(atPath:
self.applicationDocumentsDirectory.path, withIntermediateDirectories: true,
attributes: nil)
                } catch {
                    failError = nserror
                }
            } else {
                failError = nserror
            }
        }
        // Create the coordinator and store
        var coordinator: NSPersistentStoreCoordinator? = nil
        if failError == nil {
            coordinator = NSPersistentStoreCoordinator(managedObjectModel:
self.managedObjectModel)
            let url =
self.applicationDocumentsDirectory.appendingPathComponent("Caption.sqlite")
            do {
                try coordinator!.addPersistentStore(ofType:
NSSQLiteStoreType, configurationName: nil, at: url, options: nil)
            } catch {
                failError = error as NSError
            }
        }
        if shouldFail || (failError != nil) {
            // Report any error we got.
            var dict = [String: AnyObject]()
            dict[NSLocalizedDescriptionKey] = "Failed to initialize the
application's saved data" as AnyObject
            dict[NSLocalizedFailureReasonErrorKey] = failureReason as
AnyObject
            if failError != nil {
                dict[NSUnderlyingErrorKey] = failError
            }
            let error = NSError(domain: "YOUR_ERROR_DOMAIN", code: 9999,
userInfo: dict)
            NSApplication.shared.presentError(error)
```

```
            abort()
        } else {
            return coordinator!
        }
    }()
    lazy var managedObjectContext: NSManagedObjectContext = {
        let coordinator = self.persistentStoreCoordinator
        var managedObjectContext = NSManagedObjectContext(concurrencyType:
.mainQueueConcurrencyType)
        managedObjectContext.persistentStoreCoordinator = coordinator
        return managedObjectContext
    }()
    // MARK: - Core Data Saving and Undo support
    @IBAction func saveAction(_ sender: AnyObject!) {
        // Performs the save action for the application, which is to send the
save: message to the application's managed object context. Any encountered
errors are presented to the user.
        if !managedObjectContext.commitEditing() {
            NSLog("\(NSStringFromClass(type(of: self))) unable to commit
editing before saving")
        }
        if managedObjectContext.hasChanges {
            do {
                try managedObjectContext.save()
            } catch {
                let nserror = error as NSError
                NSApplication.shared.presentError(nserror)
            }
        }
    }
    func windowWillReturnUndoManager(_ window: NSWindow) -> UndoManager? {
        // Returns the NSUndoManager for the application. In this case, the
manager returned is that of the managed object context for the application.
        return managedObjectContext.undoManager
    }
    func applicationShouldTerminate(_ sender: NSApplication) ->
NSApplication.TerminateReply {
        // Save changes in the application's managed object context before
the application terminates.
        if !managedObjectContext.commitEditing() {
            NSLog("\(NSStringFromClass(type(of: self))) unable to commit
editing to terminate")
            return .terminateCancel
        }
        if !managedObjectContext.hasChanges {
            return .terminateNow
        }
        do {
            try managedObjectContext.save()
        } catch {
            let nserror = error as NSError
            // Customize this code block to include application-specific
recovery steps.
            let result = sender.presentError(nserror)
            if (result) {
                return .terminateCancel
            }
            let question = NSLocalizedString("Could not save changes while
quitting. Quit anyway?", comment: "Quit without saves error question
message")
            let info = NSLocalizedString("Quitting now will lose any changes
you have made since the last successful save", comment: "Quit without saves
error question info");
```

```
            let quitButton = NSLocalizedString("Quit anyway", comment: "Quit
anyway button title")
            let cancelButton = NSLocalizedString("Cancel", comment: "Cancel
button title")
            let alert = NSAlert()
            alert.messageText = question
            alert.informativeText = info
            alert.addButton(withTitle: quitButton)
            alert.addButton(withTitle: cancelButton)
            let answer = alert.runModal()
            if answer == NSApplication.ModalResponse.alertFirstButtonReturn {
                return .terminateCancel
            }
        }
        // If we got here, it is time to quit.
        return .terminateNow
    }
    // MARK: - Menu Bar Items
    @IBAction func togglePlaybackPausing(_ sender: Any) {
        AppDelegate.movieVC()?.playPauseClicked(sender)
    }
    @IBAction func revealVideoInFinderClicked(_ sender: Any) {
        AppDelegate.sourceListVC()?.showActiveVideoInFinderClicked(self)
    }
    @IBAction func projectNavigatorClicked(_ sender: Any) {
        AppDelegate.mainWindow()?.toggleSidebarList(self)
    }
    @IBAction func captionEditorClicked(_ sender: Any) {
        AppDelegate.mainWindow()?.contentSettingsSidebarHandler(index: 0)
    }
    @IBAction func styleEditorClicked(_ sender: Any) {
        AppDelegate.mainWindow()?.contentSettingsSidebarHandler(index: 1)
    }
    @IBAction func focusOnPlayer(_ sender: Any) {
        NSApp.mainWindow?.makeFirstResponder(AppDelegate.movieVC())
    }
    @IBAction func replaceCaptionsWithSRT(_ sender: Any) {
    }
}


//  SwiftHEXColors.swift
//
// Copyright (c) 2014 SwiftHEXColors contributors
//
// Permission is hereby granted, free of charge, to any person obtaining a
copy
// of this software and associated documentation files (the "Software"), to
deal
// in the Software without restriction, including without limitation the
rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
```

```
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
// THE SOFTWARE.
#if os(iOS) || os(tvOS)
    import UIKit
    typealias SWColor = UIColor
#else
    import Cocoa
    typealias SWColor = NSColor
#endif
private extension Int {
    func duplicate4bits() -> Int {
        return (self << 4) + self
    }
}
/// An extension of UIColor (on iOS) or NSColor (on OSX) providing HEX color
handling.
public extension SWColor {
    var fcpxString: String {
        let string = NSString(format:"%.6f %.6f %.6f %.6f",
self.redComponent, self.greenComponent, self.blueComponent,
self.alphaComponent)
        return string as String
    }
    var hexString: String {
        guard let rgbColor =
usingColorSpaceName(NSColorSpaceName.calibratedRGB) else {
            return "FFFFFF"
        }
        let red = Int(round(rgbColor.redComponent * 0xFF))
        let green = Int(round(rgbColor.greenComponent * 0xFF))
        let blue = Int(round(rgbColor.blueComponent * 0xFF))
        let hexString = NSString(format: "#%02X%02X%02X", red, green, blue)
        return hexString as String
    }
    /**
     Create non-autoreleased color with in the given hex string. Alpha will
be set as 1 by default.
     - parameter hexString: The hex string, with or without the hash
character.
     - returns: A color with the given hex string.
     */

    convenience init?(hexString: String) {
        self.init(hexString: hexString, alpha: 1.0)
    }
    private convenience init?(hex3: Int, alpha: Float) {
        self.init(red:   CGFloat( ((hex3 & 0xF00) >> 8).duplicate4bits() ) /
255.0,
                  green: CGFloat( ((hex3 & 0x0F0) >> 4).duplicate4bits() ) /
255.0,
                  blue:  CGFloat( ((hex3 & 0x00F) >> 0).duplicate4bits() ) /
255.0,
                  alpha: CGFloat(alpha))
    }
    private convenience init?(hex6: Int, alpha: Float) {
        self.init(red:   CGFloat( (hex6 & 0xFF0000) >> 16 ) / 255.0,
                  green: CGFloat( (hex6 & 0x00FF00) >> 8 ) / 255.0,
                  blue:  CGFloat( (hex6 & 0x0000FF) >> 0 ) / 255.0, alpha:
CGFloat(alpha))
    }
    /**
     Create non-autoreleased color with in the given hex string and alpha.
```

```
     - parameter hexString: The hex string, with or without the hash
character.
     - parameter alpha: The alpha value, a floating value between 0 and 1.
     - returns: A color with the given hex string and alpha.
     */
    convenience init?(hexString: String, alpha: Float) {
        var hex = hexString
        // Check for hash and remove the hash
        if hex.hasPrefix("#") {
            hex = String(hex[hex.index(after: hex.startIndex)...])
        }
        guard let hexVal = Int(hex, radix: 16) else {
            self.init()
            return nil
        }
        switch hex.count {
        case 3:
            self.init(hex3: hexVal, alpha: alpha)
        case 6:
            self.init(hex6: hexVal, alpha: alpha)
        default:
            // Note:
            // The swift 1.1 compiler is currently unable to destroy
partially initialized classes in all cases,
            // so it disallows formation of a situation where it would have
to.  We consider this a bug to be fixed
            // in future releases, not a feature. -- Apple Forum
            self.init()
            return nil
        }
    }
    /**
     Create non-autoreleased color with in the given hex value. Alpha will be
set as 1 by default.
     - parameter hex: The hex value. For example: 0xff8942 (no quotation).
     - returns: A color with the given hex value
     */
    convenience init?(hex: Int) {
        self.init(hex: hex, alpha: 1.0)
    }
    /**
     Create non-autoreleased color with in the given hex value and alpha
     - parameter hex: The hex value. For example: 0xff8942 (no quotation).
     - parameter alpha: The alpha value, a floating value between 0 and 1.
     - returns: color with the given hex value and alpha
     */
    convenience init?(hex: Int, alpha: Float) {
        if (0x000000 ... 0xFFFFFF) ~= hex {
            self.init(hex6: hex, alpha: alpha)
        } else {
            self.init()
            return nil
        }
    }
}
```