Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ НА IOS ДЛЯ РЕДАГУВАННЯ ЗОБРАЖЕНЬ

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення»

Керівник курсової роботи
ст. викладач Борозенний С. О.

_____
(підпис)
“____” _____ 2020 р.

Виконав студент 3 р. н.
Мокрий М. В.
“____” _____ 2020 р.

Київ – 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем

доцент, к.ф.-м.н.

О. П. Жежерун

_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Мокрому Михайлу Вікторовичу 3-го курсу факультету інформатики

ТЕМА Розробка мобільного додатку на iOS для редагування зображень

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Огляд операційної системи iOS

2 Аналіз предметної області

3 Проектування архітектури програми

4 Розробка додатку

Висновки

Список літератури

Дата видачі „___” _____ 2020 р. Керівник _____

Завдання отримав _____

Тема: Розробка мобільного додатку на iOS для редагування зображень

Календарний план виконання роботи:

№ п/п	Назва етапу	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу.	09.11.2019	
2.	Огляд технічної літератури за темою роботи.	28.11.2019	
3.	Знайомство з мовою програмування Swift та її основними можливостями	Грудень 2019	
3.	Проектування архітектури додатку	Січень 2019	
4.	Програмування додатку	Лютий-Березень 2020	
5.	Написання текстової частини роботи.	Квітень 2020	
6.	Створення слайдів для доповіді та написання доповіді.	10.05.2020	
7.	Захист курсової роботи		

Студент Мокрий М. В.

Керівник Борозенний О. С.

“_____”_____

Зміст

Анотація

Курсова робота присвячена розробці мобільного додатку на базі операційної системи iOS 13 для редагування графічних зображень. Для розробки цього мобільного додатку була використана мова програмування Swift 5, інтегроване середовище розробки Xcode, бібліотеки з кодом у відкритому доступі Sketch та SemiModalViewController і архітектура MVC.

Основні функції системи: Малювання і редагування графічних зображень.

Ключові слова: iOS, Xcode, Swift, Apple, iPhone, архітектура MVC, графічний редактор, мобільний додаток.

Вступ

Сучасний світ просто неможливо уявити без мобільних пристроїв, які оточують нас звідусіль. Десять років назад мобільний пристрій був тільки засобом зв'язку, справно виконуючи своє єдине призначення. Сьогодні ж телефон виконує безліч функцій, і їх кількість збільшується з року в рік. Оплата рахунку у магазині за допомогою NFC-чіпу, пам'ятні фотографії з подорожі, зроблені за допомогою камери власного смартфону, сховище для зберігання особистих даних, програвач для улюбленої музики — все це лише невелика частина того, що пропонує нам сучасний мобільний пристрій.

Неможливо уявити людину, яка щодня б не використовувала свого мобільного помічника. Саме тому ринок мобільних пристроїв величезний і дуже перспективний, а все більше розробників програмного забезпечення звертають на нього увагу. В пріоритеті стають не просто мобільні веб-версії застосунків, а повноцінні мобільні додатки, які значно покращують зручність використання та збільшують рівень взаємодії з користувачем. Тому кількість мобільних продуктів, які створенні, аби задовільнити будь-які потреби користувача, просто колосальна. Але далеко не кожен з цих продуктів має хоч якусь цінність, окрім часу, витраченого на його завантаження, і обсягу пам'яті, яку він займає. Мобільний додаток повинен не тільки бездоганно виконувати поставлену перед ним задачу, він також повинен зацікавлювати користувача своїм зручним інтерфейсом, розумно реалізованим функціоналом, яким зручно користуватися і який був би ретельно спроектований не за один день. Він має містити додаткові можливості і приємні нововведення, які будуть заохочувати людей користуватися саме цим додатком, і які виокремлять його з-поміж десятків аналогічних йому. На превеликий жаль, більшість мобільних застосунків не можуть цим похизуватися, найчастіше вони зроблені нашвидкуруч, без зайвої турботи про комфортність користувачів. Вони є лише засобом заробити та

платформою для розміщення дратівливої реклами, тому не набирають великої популярності.

Я обрав саме цю тему тому, що я впевнений, що мобільній додаток повинен приносити задоволення і бути корисним людині, для якої він був створений. Витративши певний час на перегляд офіційного онлайн магазину додатків App Store, я зауважив, що велика кількість розміщених у цьому магазині застосунків не зацікавлюють користувачів, бо вони часто дублюють один одного. Одних тільки будильників можна знайти декілька десятків, не кажучи вже про інші прикладні додатки, такі як: калькулятори, блокноти, застосунки для редагування фотографій та інші.

Тому метою цієї роботи є розробка додатку для редагування графічних зображень, який буде легким у використанні, мати інтуїтивно зрозумілий інтерфейс та досконало виконувати свої функції.

1 Операційна система iOS

1.1 Загальні відомості

iOS – це операційна система розроблена компанією Apple. Була представлена в січні 2007 року Стівом Джобсом разом з презентацією першого смартфону iPhone на всесвітній конференції Macworld Conference & Expo. На той час це був технологічний прорив у сфері мобільних технологій. Дана операційна система була першою, яка підтримувала технологію Multitouch[1]. Вона дозволяла взаємодіяти з екраном смартфону на зовсім іншому рівні, без використання стилусу і цілого набору механічних кнопок. Простота і зручність у використанні, можливість керувати мобільним пристроєм за допомогою жестів пальців — це все вплинуло і на розвиток мобільних додатків. Зручний інтерфейс без зайвого перевантаження текстовою інформацією, великі масштабовані елементи, на які зручно натискати пальцями рук, акцент не на багатофункціональності додатку, а на можливості додатку досконало виконувати певну функцію, для якої він і був створений. Хоча iOS друга по популярності операційна система в світі, поступаючись лише Android, вона безперечно має свої переваги. На відмінну від Android, вона підтримується лише на пристроях кампанії Apple, що значно покращує розробку мобільних додатків і їх тестування, бо всі технічні і фізичні характеристики мобільних телефонів заздалегідь відомі. Також хочу наголосити на тривалій підтримці застарілих моделей мобільних пристроїв. Мій мобільний телефон 2015-го року випуску навіть зараз можна оновити до найновішої, станом на квітень 2020 року, iOS 13, і це не може не захоплювати. Дана операційна система є надійною, менш уразливою перед шкідливими програмними засобами та має зручний інтерфейс. Варто зазначити, що продукція Apple об’єднується в екосистему, в якій різні пристрої взаємодіють собою.

1.2 Мова програмування

Довгі роки основною мовою програмування для розробки мобільних додатків на iOS була Objective-C[2], своєрідна мова побудована на основі мови C з використанням можливостей ООП і динамічною типізацією. Але у липні 2014 року все змінилося. Apple представила абсолютно нову об'єктно-орієнтовану мову програмування Swift[3], яка своїм зручним синтаксисом і новими можливостями суттєво полегшила життя iOS-розробників і сприяла збільшенню їх кількості у всьому світі.

Основні переваги Swift над Objective-C:

- Класи більше не розбиваються на інтерфейс і його реалізацію;
- Спрощення синтаксису створення полів і властивостей класів;
- Optional і Generics типи;
- Підтримка функціонального програмування;
- Збільшена безпека управління пам'яттю;
- Підвищена читабельність коду і зменшення його кількості;
- Повноцінна взаємодія з кодом написаним на Objective-C;
- Підтримка динамічних бібліотек.

Наразі Swift стабільно входить в 15 найпопулярніших мов програмування у світі за індексом TIOBE[4], який оцінює популярність мови програмування на основі обчислення результатів пошукових запитів, що містять згадку про мову. Аналізуючи дані за квітень 2020 року Swift займає 11 сходинку, що свідчить про високу популярність для такої, відносно молодої по сучасним міркам, мови. Тому використовувати іншу мову, для розробки мого програмного застосунку, я не бачу сенсу.

1.3 Середовище розробки

Починаючи з 2003 року Xcode[5] являється єдиним повноцінним інтегрованим середовищем розробки програмних засобів для iOS, watchOS, tvOS та MacOS. Безкоштовно поширюється через онлайн магазин додатків App Store. Дане середовище розробки містить усі необхідні засоби та надає інструменти для успішної розробки, дизайну, тестування, налагодження інтерфейсу різнопланових додатків для різних продуктів Apple. Явною перевагою Xcode є наявність Interface Builder — застосунку інтегрованого в Xcode, який містить інструментарій для створення графічних інтерфейсів. За допомогою простоти і легкості у використанні, весь GUI можна зробити в Interface Builder, а потім зв'язати всі візуальні елементи з файлами реалізації, так званими контролерами, де описується логіка взаємодії з ними. Також це середовище розробки містить інструментарій для налагодження додатку, який здатний відловити помилку як у коді, так і у інтерфейсі та розібратися з витоком пам'яті; віртуальний емулятор пристроїв та можливість напряму запускати і тестувати програми на власному смартфоні; вбудовану систему контролю версій Git; функцію імітації геолокації, так необхідну при розробці додатків, які взаємодіють з картами; Swift Playgrounds — середовище розробки Swift, яке дозволяє швидко запустити частину коду для перевірки його працездатності, або протестувати роботу певного алгоритму без компіляції цілого проекту.

Безперечно, це не єдине інтегроване середовище розробки для програмування під iOS, але воно єдине, за допомогою якого можна повністю створити додаток будь-якого рівня складності, без використання сторонніх засобів і бібліотек. Xcode регулярно оновлюється, офіційно підтримується компанією Apple і має багату документацію, саме тому я вибрав його в якості середовища розробки для мого мобільного додатку.

2 Аналіз предметної області

2.1 Характеристика предметної області

Обраною темою є створення мобільного додатку на базі операційної системи iOS для редагування графічних зображень. Основну частину вікна додатку займає робоча область для малювання, зверху знаходиться панель, в якій присутні інструменти для контролю поточного стану робочої області, додавання фону з бібліотеки або щойно зробленого за допомогою камери та збереження графічного зображення, а знизу — панель інструментів. Користувач має змогу взаємодіяти з робочою областю додатку за допомогою різних інструментів, масштабувати робочу область, зберігати створене власноруч графічне зображення, завантажувати фон з власної бібліотеки. Кожен елемент користувацького інтерфейсу розроблений таким чином, щоб бути інтуїтивно зрозумілим без будь-якої сторонньої допомоги. Додаток містить прості інструменти для малювання різних фігур, пензлі трьох типів, інструмент для заливки вибраної області та інструмент для стирання вибраної області. Також в додатку присутня можливість змінювати товщину пензля і фігури, включно з їх кольором і прозорістю. На панелі інструментів, у правому кутку, розміщений індикатор, який показує поточний вибраний колір і одночасно слугує зручним знаряддям для швидкої зміни кольору. Варто зазначити, що кожна дія, яка відбувається в робочій області графічного редактору, запам’ятовується і за допомогою кнопок Undo та Redo може бути скасована або відновлена. Довжина історії змін обмежується лише кількістю дій, які були зроблені під час поточного сеансу. В разі потреби користувач може повністю очистити робочу область.

2.2 Функціональні вимоги

В попередніх розділах було розглянуто мету моєї роботи, тому доречно було б зазначити функціональні вимоги, які були поставлені переді мною при розробці мобільного додатку, а саме:

- Можливість малювати різними типами пензлів;
- Можливість будувати різні види фігур;
- Зміна кольору пензля та фігури;
- Регулювання товщини пензля і його прозорості;
- Можливість малювати як на зображеннях з власної бібліотеки, так і на щойно зроблених за допомогою камери;
- Наявність різноманітних штампів (стікерів);
- Збереження поточного стану робочої області, в якому кожна дія може бути скасована та відновлена;
- Можливість повність очищати робочу область;
- Масштабування робочої області;
- Збереження створення графічного зображення у бібліотеку;
- Адаптація додатку під розміри екрану різних мобільних пристроїв.

3 Проектування

3.1 Архітектура програмної системи

Створення кожного додатку, який розробляється більш-менш досвідченою і професійною командою, повинне містити етап проектування. Цей етап є чи не найважливішим етапом, бо під час нього закладається фундамент майбутнього додатку, його хребет. Як відомо «хребет» додатку — це його архітектура. Невірно розроблена архітектура може перетворити життя розробників на суцільне пекло й негативно впливати на кінцеву якість продукту. Такі додатки дуже важко підтримувати в робочому стані, а про розширення їх функціоналу можна взагалі забути, тому що змінювати архітектуру дуже складно навіть на застосунках, де ця архітектура була правильно розроблена. Хороша архітектура робить процес розробки і обслуговування додатку більш ефективним і простим, дозволяючи розробником краще орієнтуватися в написаному коді, легше розширювати додаток додатковими можливостями і функціями, змінювати окремі компоненти й тестувати його.

Продумана архітектура програмної системи повинна відповідати таким критеріям:

- Збалансований розподіл функцій між діючими об’єктами та їх ефективність;
- Можливість розширюватися;
- Зручність використання.

Тому необхідно детально продумати архітектуру додатку на початковому етапі проекту. В цьому допомагають шаблони проектування, які містять сукупність загальноприйнятих правил і норм при розробці архітектури програмної системи.

3.2 Опис шаблону проектування

Сьогодні існує багато шаблонів проектування , але я хочу звернути увагу на Model-View-Controller (або Модель-Представлення-Контролер)[6], який є одним з найпопулярніших і найпоширеніших з них.

MVC — схема проектування архітектури програмної системи, де модель, користувацький інтерфейс та взаємодія з користувачем розділені на три окремих рівні: рівень моделі, представлення і контролеру, кожен з яких виконує свою функцію, як показано на рисунку 3.2.1.

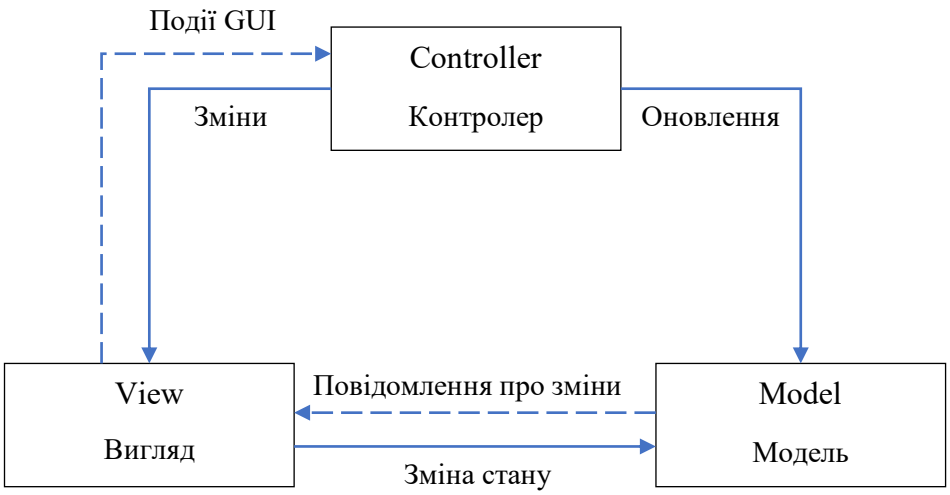

Рисунок 3.2.1 Схема взаємодії в MVC

Модель — це частина архітектури додатку, яка відповідає за керування даними. Контролер реагує на дії користувача й оновлює вигляд, використовуючи зміни в моделі. Вигляд відповідальний за правильне представлення даних з моделі за допомогою графічного користувацького інтерфейсу або іншими словами — GUI.

Традиційна архітектура MVC, відрізняється від архітектури, яка використовується при сучасній розробці iOS-додатків. У традиційній архітектурі MVC всі три сутності тісно пов’язані між собою, що суттєво знижує можливість повторного використання кожного з цих компонентів. Через цей чинник реалізовувати її у середовищі операційної системи iOS немає ніякого сенсу.

В офіційній документації Apple зазначений шаблон MVC, який був названий розробниками як Cocoa MVC[7]. Його схема проектування зображена на рисунку 3.2.2. В цьому шаблоні контролер являється посередником між рівнем представлення і моделлю, які нічого не знають про існування один одного і не можуть взаємодіяти між собою. Також особливістю цього шаблона проектування архітектури є те, що контролер UIViewController настільки залучений у життєвий цикл представлення UIView, що іноді важко сказати, що ці два рівні дійсно розділені між собою.

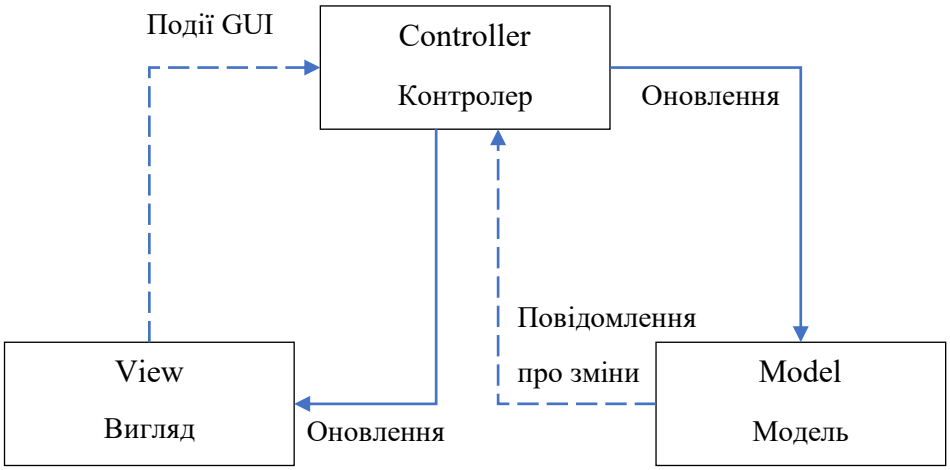

Рисунок 3.2.2. Схема взаємодії в Cocoa MVC

Cocoa MVC є найкращим шаблоном проектування з точки зору швидкості реалізації і простоти використання. Він незамінний в більшості мобільних додатків невеликої складності. Тому даний шаблон був використаний під час розробки архітектури для мого мобільного застосунку.

4 Структура iOS-додатку

4.1 Графічний інтерфейс користувача

Графічний інтерфейс є невідмінною складовою кожного додатку, який не тільки забезпечує презентабельний вигляд, від нього ще й залежить рівень сприйняття візуальної інформації користувачем. Гарно продуманий і зручний інтерфейс є запорукою збільшення рівня залучення користувача у функціонал програмного продукту, для реалізації якого він і був створений.

Я намагався розробити користувацький інтерфейс якнайбільш зручним. Для цього, в більшості випадків, використовувалися стандартні елементи інтерфейсу, які є перевіреними часом, гармонійно вписуються в загальний стиль iOS 13 та активно використовуються у всіх офіційних додатках Apple.

При запуску додатку користувач потрапляє на головний екран (див. рис. 4.1.1), на якому є три області. Найголовнішим компонентом є робоча область додатку (полотно для малювання), яка розташовується по центру екрану. Вона дозволяє користувачеві взаємодіяти з нею за допомогою різних інструментів. Зверху знаходиться панель навігації UINavigationBar, в якій зліва розташовані інструменти контролю поточного стану та очищення робочої області, а справа — інструменти для зміни фону та збереження графічного зображення. Знизу розташована панель наявних інструментів UIToolbar для взаємодії з робочою областю та індикатор поточного кольору в правому куті. Варто зазначити, що кожен елемент в панелі інструментів має своє додаткове вікно UIView, яке відкривається при натисненні на нього на певну висоту екрану і в якому розташований весь його власний функціонал.

Важливо зауважити, що мій додаток підтримує одне з найбільших нововведень iOS 13 — темну тему оформлення. Це означає, що користувач має можливість перемикатися на світлу і темну теми, які автоматично змінюють кольорову гаму всього інтерфейсу операційної системи, а також усіх системних програм. Як виглядає інтерфейс мого додатку з темною темою можна переконатися на рисунку 4.1.1. Але в подальшому, я буду показувати інтерфейс на прикладі світлої теми, щоб не перевантажувати цю роботу надлишковою графічною інформацією.

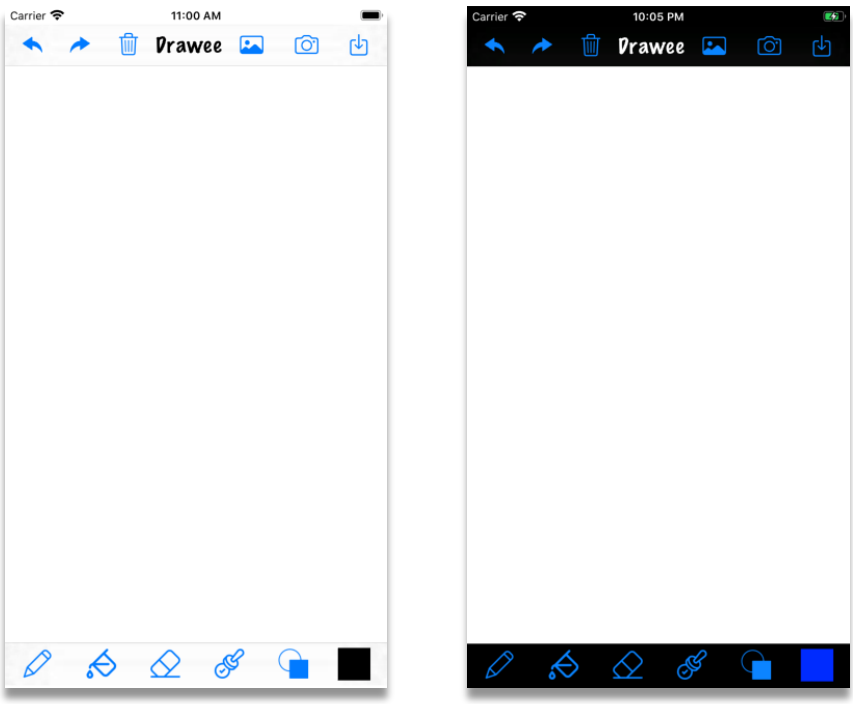

Рисунок 4.1.1 Головний екран додатку (світла і темна теми)

Натиснувши на інструмент «Пензлик», користувачу відкривається додатковий екран, зображений на рисунку 4.1.2, де можна змінювати тип пензля, який присутній у трьох видах, а саме: звичайний, розмивчастий та неоновий. Також можна регулювати товщину, прозорість даного інструменту за допомогою двох повзунків UISlider та його колір за допомогою UIPickerView.

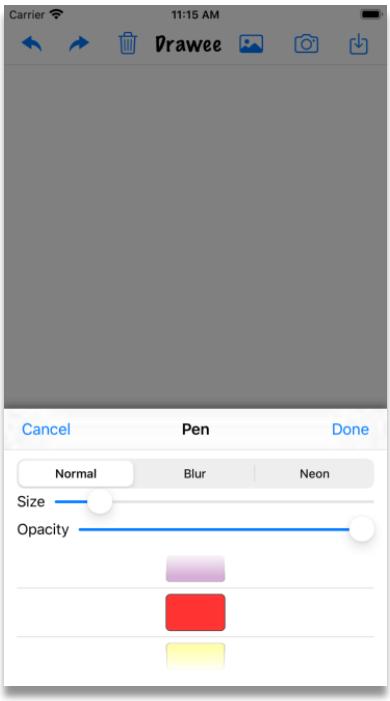

Рисунок 4.1.2 Екран, що відображає функціонал інструменту «Пензлик»

Можливості інструменту «Заповнення» більш обмежені ніж можливості пензля, тому користувачеві відкривається набагато менший додатковий екран (див. рис. 4.1.3), розмір якого залежить від кількості елементів у UIView. Заповнення заливає простір вибраним кольором. Його вибір реалізований за допомогою UIPickerView. Також даний екран використовується при зміні кольору через розташований у правому куті індикатор кольорів. Достатньо зробити довге або подвійне натискання на нього, і відкриється той самий екран, який використовується в інструменті «Заповнення». Але є лише одна відмінність — ми не змінюємо поточний інструмент на заповнення, а лише колір цього інструмента.

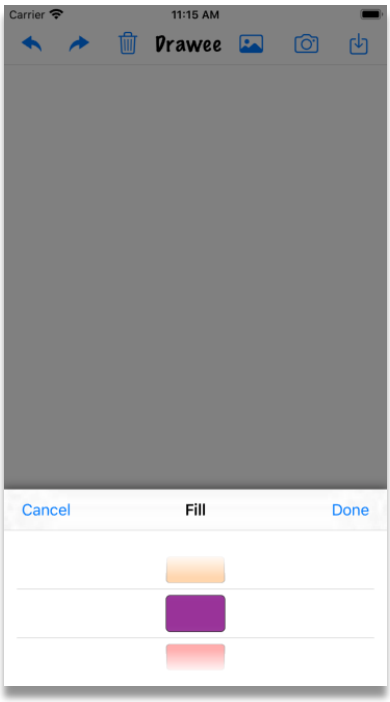

Рисунок 4.1.3 Екран, що відображає функціонал інструменту «Заповнення» та індикатору кольорів

Повзунок UISlider інструменту «Гумка», розташований на додатковому екрані (див. рис. 4.1.4), має здатність збільшувати або зменшувати площу стирання. Це дуже зручно, коли потрібно стерти або дуже велику, або дуже малу площу графічного зображення, не очищуючи повністю робочу область.

Рисунок 4.1.4 Екран, що відображає функціонал інструменту «Гумка»

Додатковий екран інструменту «Штамп» (див. рис. 4.1.5) містить зручний UICollectionView з однією горизонтальною колонкою, який відображає усі штампи, які присутні в моєму додатку. Достатньо провести пальцем праворуч або ліворуч, щоб вибрати потрібний штамп з наявного переліку.

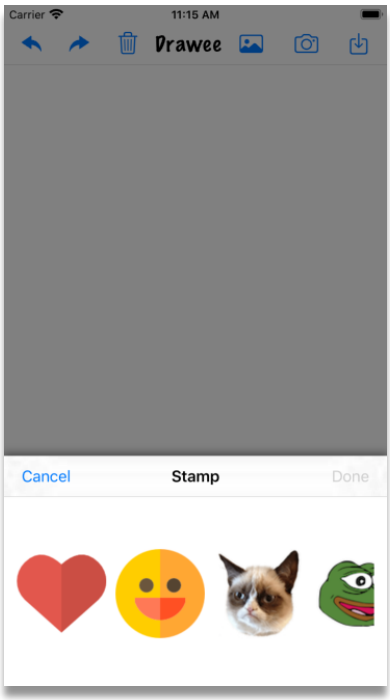

Рисунок 4.1.5 Екран, що відображає функціонал інструменту «Штамп»

Додатковий екран інструменту «Фігура», показаний на рисунку 4.1.6, містить повзунки для регулювання товщини фігури і ступеня її прозорості, нижче розташований UIPickerView для зміни кольору фігури, ще нижче — UICollectionView з однією горизонтальною колонкою, який містить такі фігури, як: коло, заповнене коло, квадрат, заповнений квадрат, зірка, стрілочка.

Важливо зауважити, якщо фігура не була вибрана, то кнопка підтвердження залишиться неактивною, і при виході з цього додаткового екрану за допомогою кнопки відміни, залишиться попередній інструмент, що використовувався. Так само працює і додатковий екран інструменту «Штамп». Всі інші додаткові екрани з панелі інструментів не потребують даного функціоналу, бо модель робочої області завжди зберігає поточний колір, товщину і прозорість, і немає сенсу змінювати один з цих елементів, для того щоб змінити інструмент малювання.

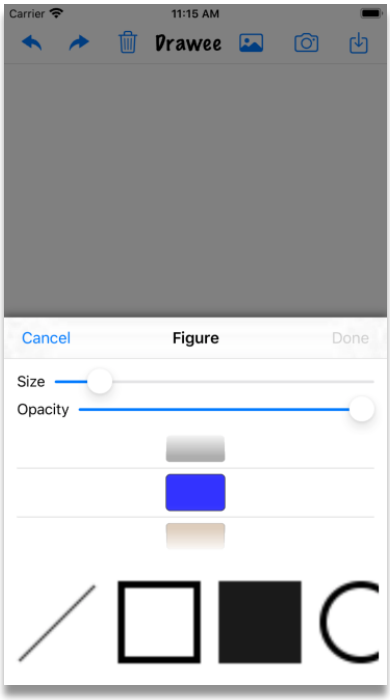

Рисунок 4.1.6 Екран, що відображає функціонал інструменту «Фігура»

Хотів би додати, що мобільний додаток адаптований під різні розміри екранів телефонів Apple (див. рис. 4.1.7) за допомогою технології Auto Layout[8], яка динамічно вираховує розмір і положення графічних об’єктів в ієрархії UIView, ґрунтуючись на обмеженнях, зазначених для кожного з цих об’єктів.

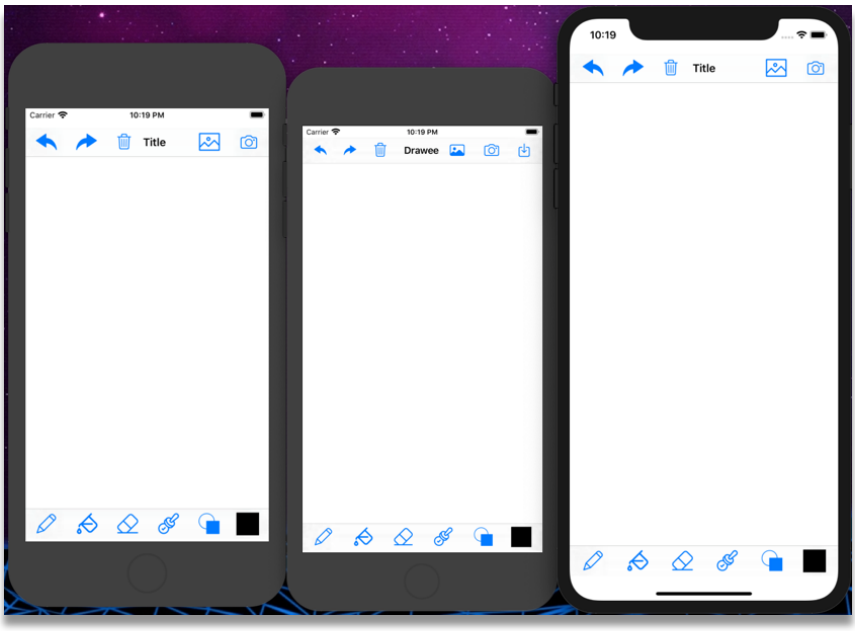

Рисунок 4.1.7 Головний екран на смартфонах iPhone 8, iPhone 8 Plus, iPhone 11 Pro (зліва направо)

4.2 Структура класів

Структура додатку складається з чотирьох допоміжних класів, шести класів-контролерів, головного storyboard[9] з Interface Builder, в якому знаходяться всі шість інтерфейсів UIView, одного додаткового інтерфейсу, що створюється прямо з коду, чотирьох моделей, двох бібліотек з кодом у відкритому доступі, не враховуючі класів в цих в бібліотеках, а також стандартної графічної бібліотеки UIKit.

Короткий опис усіх класів додатку:

- AppDelegate, SceneDelegate

  Класи, відповідальні за загальний стан додатку, управління екранами — рівень ядра нашого додатку.

- MainViewController

  Клас контролю головного вікна UIView і робочої області застосунку, в якій використовується спеціальне вікно UISketchView з бібліотеки для малювання Sketch для взаємодії з інструментарієм додатку.

- PenViewController

  Клас, відповідальний за взаємодію користувача з додатковим вікном інструменту «Пензлик».

- FillViewController

  Клас, відповідальний за взаємодію користувача з додатковим вікном інструменту «Заповнення» та індикатором поточного кольору.

- EraserViewController

  Клас, відповідальний за взаємодію користувача з додатковим вікном інструменту «Гумка».

- StampViewController

Клас, відповідальний за взаємодію користувача з додатковим екраном інструменту «Штамп». Містить UICollectionView, який дозволяє відображати на екрані колекцію довільних елементів. Він використовує клас CustomCell для відображення наявних штампів.

- FigureViewController

Клас, відповідальний за взаємодію користувача з додатковим екраном інструменту «Фігура». Містить UICollectionView який використовує клас CustomCell для відображення наявних фігур та Color-PickerView для вибору кольору.

- ColorPickerView

Клас, який містить реалізацію власного UIPickerView, колеса прокрутки для вибору кольору поточного інструменту.

- Figure, Stamp, Color

Класи, що уособлюють в собі моделі сутностей фігури, штампу, кольору.

- CustomCell

Клас відображає сутність клітинки штампу та фігури, взаємодіючі з UICollectionView та класами Figure та Stamp.

- PhotoAuthorization, RequestPhotoManager

Класи, за допомогою яких реалізовується доступ до камери і галереї зображень. Вони знаходяться у відкритому доступі.

4.3 Використані бібліотеки

У процесі розробки я використав менеджер залежностей CocoaPods[10]. Він допомагає iOS-розробникам легко додавати сторонні бібліотеки, код яких знаходиться у відкритому доступі. CocoaPods створює єдину централізовану систему керування залежностями в проекті. Після додавання до нього нової бібліотеки, назва якої вказується в єдиному текстовому файлі «Podfile» (див. рис. 4.3), менеджер автоматично слідкує за її подальшою підтримкою і оновленнями.

```
target 'MyPaint' do
  use_frameworks!
  # Pods for MyPaint
 pod 'Sketch'
 pod 'SemiModalViewController'
end
```

Рисунок 4.3 Podfile мого проекту

Бібліотеки, які я використав:

- Sketch

  Бібліотека з кодом у відкритому доступі, яка має основні функції для малювання і редагування графічних зображень. Весь її функціонал, в повному обсязі був впроваджений і повністю реалізований в моєму додатку.

- SemiModalViewController

  Розширення UIViewController, яке дозволяє відкривати новий UIView не на весь екран, а на задану висоту з плавними анімаціями переходу. Воно використовується при відкритті вікна з додатковим функціоналом кожного елемента з панелі інструментів.

Висновки

Результатом виконаної роботи стало створення мобільного додатку для редагування зображень на платформі iOS. Програмний продукт був розроблений на мові програмування Swift в інтегрованому середовищі розробки Xcode.

Було досліджено і проаналізовано сучасні методи і підходи до розробки мобільних застосунків.

При створенні цього програмного продукту, були успішно виконані всі функціональні вимоги, які ставились переді мною. Розроблений додаток надає змогу в повному обсязі використовувати всі його технічні можливості та виконувати поставлені користувачами задачі.

Також було розроблено зручний і інтуїтивно зрозумілий інтерфейс з адаптацією під різні розміри екранів, що дозволяє використовувати цей застосунок на будь-якому мобільному телефоні компанії Apple, який підтримує операційну систему iOS 13.

Список літератури

1. Inside the multitouch FingerWorks tech in Apple's tablet [Електронний ресурс]. – 2010. – Режим доступу до ресурсу: https://appleinsider.com/articles/10/01/23/inside_the_multitouch_fingerworks_tech_in_apples_tablet/page/3.

2. About Objective-C [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC.

3. Neuburg M. iOS 13 Programming Fundamentals with Swift: Swift, Xcode, and Cocoa Basics / Matt Neuburg. – O'Reilly Media, 2019. – 680 с.

4. TIOBE Index [Електронний ресурс] – Режим доступу до ресурсу: https://www.tiobe.com/tiobe-index/.

5. Xcode Overview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/index.html#//apple_ref/doc/uid/TP40010215-CH24-SW1.

6. Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: https://patterns.arc42.org/patterns/mvc/#mvc-triads.

7. Model-View-Controller (Cocoa MVC) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html#//apple_ref/doc/uid/TP40008195-CH32-SW1.

8. Working with Xcode Auto Layout in Swift and iOS Projects [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.twilio.com/blog/2018/05/xcode-auto-layout-swift-ios.html.

9. iOS Storyboards: Getting Started [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.raywenderlich.com/5055364-ios-storyboards-getting-started.

10. CocoaPods Guides – Using CocoaPods [Електронний ресурс] – Режим доступу до ресурсу: https://guides.cocoapods.org/using/using-cocoapods.html.

Додаток А (обов'язковий). Лістінг коду, який відповідає за взаємодію користувача з додатковим вікном інструменту «Пензлик».

```
import Foundation
import UIKit
import SemiModalViewController
import Sketch


class PenViewController: UIViewController {

    var penType: PenType = .normal
    var penColor: UIColor = .black
    var widthSliderValue = CGFloat(10)
    var opacitySliderValue = CGFloat(1)

    weak var delegate: MainViewControllerDelegate?

    @IBOutlet weak var segmentPenType: UISegmentedControl!
    @IBOutlet weak var widthSlider: UISlider!
    @IBOutlet weak var opacitySlider: UISlider!
    @IBOutlet weak var pickerView: ColorPickerView!

    override func viewDidLoad() {
        super.viewDidLoad()
        pickerView.delegate = pickerView
        pickerView.dataSource = pickerView
        pickerView.pickedColor = penColor
        changeSegmentIndex()
        changeSelectedColor()
        changeSliderValues()
    }


     @IBAction func changeDataInFirstVC() {
            delegate?.updateDrawTool(.pen)
            delegate?.updatePenType(type: penType)
            delegate?.updateColor(pickerView.pickedColor)
            delegate?.widthSliderMoved(widthSlider)
            delegate?.opasitySliderMoved(opacitySlider)
         dismissSemiModalView()
     }


    @IBAction func indexChanged(_ sender: Any) {
        switch segmentPenType.selectedSegmentIndex
        {
        case 0:
            penType = .normal
        case 1:
            penType = .blur
        case 2:
            penType = .neon
```

```
            default:
                break
            }
        }

        @IBAction func dismiss(_ sender: Any) {
            dismissSemiModalView()
        }


        private func changeSliderValues(){
            let widthValue =  Float(widthSliderValue).rounded()
            widthSlider.value = widthValue
            opacitySlider.value = Float(opacitySliderValue)
        }

        private func changeSelectedColor(){
            let row =  getColorIndex(pickerView.pickedColor)
            pickerView.selectRow(row, inComponent: 0, animated: true)
        }

        private func getColorIndex( _ penColor: UIColor) -> Int {
            let colorsArr = fixedColors.map{ $0.color }
            return colorsArr.firstIndex(of: pickerView.pickedColor) ?? 0
        }

        private func changeSegmentIndex(){
            switch penType
                   {
                   case .normal:
                       segmentPenType.selectedSegmentIndex = 0
                   case .blur:
                       segmentPenType.selectedSegmentIndex = 1
                   case .neon:
                      segmentPenType.selectedSegmentIndex = 2
                   }
        }
}
```

Додаток Б (обов'язковий). Лістінг коду, який відповідає за відображення колеса прокрутки для вибору кольору

```
import UIKit

class ColorPickerView: UIPickerView{

       var pickedColor: UIColor = .black

}

extension ColorPickerView: UIPickerViewDataSource, UIPickerViewDelegate {

          func numberOfComponents(in pickerView: UIPickerView) -> Int {
                return 1
          }

         func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent
component: Int) -> Int {
                return fixedColors.count
         }

         func pickerView(_ pickerView: UIPickerView, rowHeightForComponent
component: Int) -> CGFloat {
                return 50.0
         }

         func pickerView(_ pickerView: UIPickerView, viewForRow row: Int,
forComponent component: Int, reusing view: UIView?) -> UIView {
            let myView = UIView(frame: CGRect(x: 0, y: 0, width:
pickerView.bounds.width/6, height: 40))
            let colorCell = colorCellView(row)
                myView.addSubview(colorCell)
                colorCell.topAnchor.constraint(equalTo:
myView.topAnchor).isActive = true
                colorCell.leftAnchor.constraint(equalTo:
myView.leftAnchor).isActive = true
                colorCell.rightAnchor.constraint(equalTo:
myView.rightAnchor).isActive = true
                colorCell.bottomAnchor.constraint(equalTo:
myView.bottomAnchor).isActive = true
            return myView
         }

         func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
inComponent component: Int) {
               pickedColor = fixedColors[row].color
         }

        private func colorCellView(_ row: Int) -> UIView {
            let colorView = UIView(frame: .zero)
                colorView.backgroundColor = fixedColors[row].color
                colorView.translatesAutoresizingMaskIntoConstraints = false
                colorView.contentMode = .scaleAspectFill
                colorView.clipsToBounds = true
```

```
                colorView.layer.cornerRadius = 5
                colorView.layer.borderWidth = 0.75
                colorView.layer.borderColor =  colorLiteral(red: 0, green:
0, blue: 0, alpha: 1)
            return colorView
        }

}
```