

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

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ЛЮДЕЙ З ВАДАМИ ЗОРУ
НА ОСНОВІ ТЕХНОЛОГІЇ КОМП'ЮТЕРНОГО БАЧЕННЯ
Текстова частина до курсової роботи
за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи
с.в. Борозенний С.О.
(прізвище та ініціали)

(підпис)

“ ____ ” ____ 2020 р.

Виконав студент БП ІПЗ-3
Сабадишин М.О.
(прізвище та ініціали)

(підпис)

“ ____ ” ____ 2020 р.

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЇВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. Кафедри мультимедійних систем,
доцент, к.ф-м.н.

_____ О.П. Жежерун

(підпис)

“ ____ ” _____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

Студенту Сабадишина Максима Олександровича факультету інформатики 3 курсу

ТЕМА розробка мобільного застосунку для людей з вадами зору на основі технології комп'ютерного бачення

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Аналіз предметної області та постановка завдання

Теоретичні відомості щодо використаних технологій

Опис реалізації застосунку

Висновки по роботі та аналіз можливостей подальшого розвитку застосунку

Список використаної літератури

Додатки

Дата видачі “ ____ ” _____ 2020 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання роботи:

Тема: розробка мобільного застосунку для людей з вадами зору на основі технології комп'ютерного бачення

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової	05.10.2019	
2.	Ознайомлення з предметною областю	18.10.2019	
3.	Пошук корисної літератури	25.10.2019	
4.	Ознайомлення з літературою	02.01.2020	
5.	Проведення дослідження	10.01.2020	
6.	Аналіз отриманих результатів	15.01.2020	
7.	Планування структури та архітектури практичної частини роботи	15.02.2020	
8.	Планування та створення зручного дизайну та керування застосунком для людей з вадами зору	25.02.2020	
9.	Створення функціоналу з розпізнавання тексту	07.03.2020	
10.	Створення функціоналу з розпізнавання кольорів	20.03.2020	
11.	Створення бази даних та написання функціоналу збереження даних до неї	24.03.2020	
12.	Реалізація голосових команд	02.04.2020	
13.	Рефакторинг коду	10.04.2020	
14.	Написання текстової частини роботи	08.05.2020	
15.	Створення презентації та доповіді до неї	09.05.2020	
16.	Здача курсової роботи на перевірку	11.05.2020	

Студент Сабадишин М.О.

Керівник Борозенний С.О.

“ ”

ЗМІСТ

Анотація	6
Вступ.....	7
Актуальність теми та її практичне значення	7
Структура роботи	8
Розділ 1. Аналіз предметної області та постановка завдання.....	9
1.1 Аналіз проблем в побуті для людей з вадами зору	9
1.2 Аналіз існуючих застосунків, що вирішують проблеми людей з вадами зору	10
1.3 Основні поняття технології комп'ютерного бачення	12
1.4 Особливості створення застосунку для людей з вадами зору.....	13
1.5 Висновки до розділу 1.....	18
Розділ 2. Теоретичні відомості щодо використаних технологій.....	19
2.1 Аналіз особливостей розробки під iOS.....	19
2.2 Загальна інформація про Swift як мову програмування	22
2.3 Використання фреймворку Vision для застосування технології комп'ютерного бачення в iOS застосунках	24
2.4 Використання фреймворку Speech для розпізнавання голосових команд користувачем ..	26
2.5 Загальні відомості про SQLite	28
2.6 Висновки до розділу 2.....	29
Розділ 3. Опис реалізації застосунку	30
3.1 Аналіз технічного завдання	30
3.2 Пояснення вибору використаних технологій	30
3.3 Опис та пояснення вибору середовища розробки.....	33
3.4 Опис та обґрунтування архітектури проєкту	36
3.5 Обґрунтування дизайну та взаємодії з користувачем	36
3.6 Опис розробки застосунку	39
3.7 Принципи роботи готового застосунку.....	48
3.8 Висновки до розділу 3	49

Висновки по роботі та аналіз можливостей для подальшого розвитку застосунку..... 50

Список джерел 52

Додаток А. Функції перетворення зображень 54

Анотація

Метою даної курсової роботи є розробка мобільного додатку, що допоміг би людям з вадами зору інтегруватись в сучасний соціум та став би альтернативою людському зору в певних випадках.

Проведено аналіз користування мобільними пристроями людьми з вадами зору та вивчено методи щодо полегшення управління застосунком. Також було вивчено основні проблеми сучасного суспільства, пов'язані з доступністю, та детальніше розглянуто ті, які можна було б вирішити з використанням мобільних технологій та технології комп'ютерного бачення.

Закінчивши аналіз, було обрано 2 конкретні проблеми та складено план розробки застосунку, після чого його було розроблено з урахуванням того, що цільова аудиторія – це люди з вадами зору.

Ключові слова: мобільні технології, комп'ютерне бачення, розробка додатку, люди з вадами зору, доступність, iOS.

Вступ

Актуальність теми та її практичне значення

Смартфони вже багато років відіграють важливу роль в житті кожної людини. Вони надають постійний доступ до актуальної інформації, забезпечують контакт з близькими, та розважають користувачів медіа контентом та іграми. Сьогодні їх роль тільки збільшилась, а корисність набуває небачених висот. Популярність смартфонів сприяє їх прогресу та позитивно впливає на появу нових технологій.

До прикладу, технологія комп'ютерного бачення. Вона набула швидкого розвитку та популярності в останні роки, здебільшого через значний прогрес у сфері штучного інтелекту та машинного навчання.

Комп'ютерне бачення вирішує багато проблем, включаючи ідентифікацію об'єктів, модерацію контенту, та розпізнавання тексту. Google, наприклад, використовує технологію для розпізнавання та класифікації об'єктів в галереї, а Adobe використовує його в Lightroom для покращення приближення зображень. Напевне, найбільший прогрес з розвитком комп'ютерного бачення отримала функція розпізнавання обличчя, яка стала провідним методом в розблокуванні смартфонів. Все це підтверджує, що подальший розвиток цієї технології неминучий, адже провідні компанії вкладають свої кошти в дослідження в цій галузі, а нові фреймворки для роботи з комп'ютерним баченням та додатки з використанням технології з'являються кожного дня.

Все це разом відкриває нові горизонти та дозволяє задумуватись про вирішення проблем абсолютно іншого рівня. Мобільні пристрої перестають бути лише засобами для розваг чи отримання інформації – вони стають інструментами для розв'язування проблем на індивідуальному рівні, що до цього були недоступними через дороговизну або ж недостатній розвиток технологій. Відслідковування циклів сну, збирання статистики про активність впродовж дня, можливість перевіряти локацію дитини в екстрених ситуаціях, та багато інших – все це корисні речі, що лише укріплюють роль мобільних пристроїв як незамінних допоміжних інструментів.

Та це теж не використовує їх потенціал повною мірою. За даними дослідження, проведеного “Georgia Tech’s Wireless Engineering Rehabilitation Research Center”, 92% людей з різними вадами здоров’я користуються смартфонами чи планшетами.[1] Для них кожен день несе в собі додаткові труднощі, адже велика кількість повсякденних речей зроблена з думкою лише про сценарії роботи для повністю здорової людини. Така орієнтованість ускладнює життя для людей з вадами здоров’я і ніяк не допомагає їм. А тому маючи таку базу користувачів, розробники мають втілювати проекти, мета яких – допомогти цим людям виконувати різні повсякденні дії та таким чином полегшувати їх життя. На сьогоднішній день, найпопулярніші мобільні ОС Android та iOS мають вбудовані механізми для роботи людей з вадами як от збільшення розмірів вмісту екрану, озвучення вмісту екрану, інверсія кольорів екрану, різні кольорові схеми, та інші.[2] Проте їх користь закінчується, коли користувач перестає користуватись функціоналом девайсу та переходить в реальне життя.

Виходячи з потреби підтримки людей з обмеженими можливостями, а також високого потенціалу сьогоднішніх мобільних пристроїв для виконання задач такого типу, за мету даної роботи була поставлена реалізація програмного застосунку для мобільних пристроїв з функціоналом для допомоги людям з вадами зору на базі iOS.

Структура роботи

Дана робота складається з трьох основних розділів.

Перший розділ містить аналіз предметної області, дослідження основних потреб для людей з вадами зору та розгляд можливостей щодо їх вирішення за допомогою мобільних технологій та комп’ютерного бачення.

У другому розділі описані ті технології, способи роботи з ними, які було використано під час розробки iOS додатку.

Третій розділ містить деталі розробки, а також опис роботи з готовим програмним продуктом, його особливості та зручності.

Розділ 1. Аналіз предметної області та постановка завдання

1.1 Аналіз проблем в побуті для людей з вадами зору

Проблеми з якими кожного дня стикаються люди з вадами зору все ще не до кінця добре вивчені. Загалом, зрозуміло, що обмеження або відсутність зору впливає на сприйняття навколишнього середовища та орієнтацію в ньому. Але необхідно проаналізувати декілька аспектів з життя для того, щоб краще дізнатись про існуючі конкретні проблеми.

Однією з найбільших проблем для людей з вадами зору є складнощі повноцінного соціального життя. Крім того факту, що втрата зору або проблеми з ним є самі по собі складністю, яку необхідно прийняти і з якою необхідно навчитись жити, труднощі з пошуком занять або можливостей соціалізації тільки погіршують становище. Кількість інклюзивних занять дуже мала і сконцентрована переважно у великих містах, а типові місця, як от музеї, найчастіше просто не адаптовані для відвідувачів з вадами здоров'я.

Здавалось би, давно відомо, що люди з вадами зору не мають можливості читати книги та інші друковані матеріали, і ця проблема має поступово вирішуватись. На жаль, ситуація дещо гірша. За даними WBU, на сьогоднішній день більше, ніж 90% всіх друкованих матеріалів не є доступними для людей з повною або частковою відсутністю зору.[3] Тобто відсутні альтернативні варіанти з шрифтом Брайля або в аудіо форматі.

Для прикладного розуміння наступних проблем відтворимо похід в магазин.

Всі люди так чи інакше ходять за покупками, це тривіальна задача з виконанням якої у повністю здорової людини не виникає жодних проблем. Та коли це завдання постає перед людиною з обмеженим зором, воно перестає бути таким простим.

Навігація до бажаного місця – перша проблема, яка виникає перед людиною з вадами зору. Вона не так гостро стоїть коли мова йде про знайоме оточення, але якщо необхідно дійти до нового незнайомого місця, то подолання

маршруту може принести складнощі та вимагати пошуку допомоги від незнайомих оточуючих.

Подолавши маршрут і опинившись в бажаному магазині, людина з вадами зору стикається з наступною проблемою – проблемою пошуку та ідентифікації бажаних товарів. Інформація у всіх магазинах подана в текстовому форматі без альтернатив для людей з вадами зору. Навіть запам'ятавши форму та місце, де шукати необхідну річ, можна помилитись з певною специфікацією товару. Наприклад, прийшовши в магазин за дієтичною колою, навіть знаючи форму пляшки неможливо зрозуміти, яку саме ти обрав, або ж прийшовши в магазин іграшок за подарунком і маючи бажання обрати товар білого кольору.

Проблема ідентифікації та пошуку – найглобальніша для людей з вадами зору і виходить далеко за рамки купівлі товарів. Більшість інформації про річ ми отримуємо подивившись на неї – склад, термін придатності, колір, та іншу додаткову інформацію. Сьогодні більшість виробників не надають можливості альтернативно її отримати, тому людям з вадами зору залишається тільки просити допомогу у інших людей, сподіваючись, що хтось буде поруч. Але як вже було згадано, ця проблема трапляється не тільки в магазинах і може виникнути будь-де. Це так само може трапитись вдома, де не буде нікого з можливістю допомогти і в залежності від критичності ситуації, наслідки можуть бути серйозними.

Сьогоднішнє суспільство зробило занадто високу ставку на візуальне сприйняття речей, абсолютно забувши про випадки коли воно не є оптимальним, а враховуючи, що у світі близько 39 мільйонів людей з повністю відсутнім зором[4], це абсолютно недопустима помилка.

1.2 Аналіз існуючих застосунків, що вирішують проблеми людей з вадами зору

Для кращого розуміння шляхів вирішення поточних проблем, важливо провести аналіз вже існуючих рішень у цьому сегменті. Враховуючи популярність та доступність смартфонів, а також той факт, що багато людей з вадами зору мають мобільні пристрої[1], багато компаній починають

реалізовувати елементи доступності у своїх застосунках. Смартфони компактні і більшість користувачів тримає їх при собі увесь день. Така автономність робить їх корисними у різних ситуаціях та дозволяє слугувати помічниками для людей з вадами зору.

Проблема соціального життя також значно полегшується смартфонами. Найпопулярніші мобільні ОС підтримують різні режими для зручного керування людям з вадами здоров'я без додаткового програмного забезпечення, а тому вони можуть проводити час слухаючи книги, подкасти, або ж контактуючи з близькими. Проте проблемою залишається той факт, що все ще далеко не всі веб-сайти є пристосованими для відвідувачів з вадами зору, що обмежує їх у споживанні контенту.

Іншою згаданою раніше проблемою була навігація. Сьогодні на ринку присутні навігаційні застосунки, які містять підтримку роботи з користувачами, що мають порушення зору. Одним з таких прикладів є Google Maps[5]. Google Maps має звукову навігацію не тільки під час поїздок машиною, а і під час ходьби, що суттєво полегшує завдання дійти до необхідної точки людям з вадами зору.

Проблема читання та недоступності друкованих матеріалів має одне коріння з проблемою пошуку та ідентифікації. Обидві можна звести до однієї спільної – неможливість прочитати текст і відсутність альтернативного способу отримати дану інформацію. Ця проблема є складнішою з точки зору реалізації і не має як таких популярних рішень. Проте є один приклад, що заслуговує на увагу. Застосунок Be My Eyes[6] використовує інший підхід і допомагає людям з порушеннями зору отримувати інформацію через волонтерів. Коли користувачеві необхідна допомога, здійснюється дзвінок і волонтер через відеозв'язок відповідає на запитання. Звичайно, такий підхід вимагає великої кількості волонтерів та не гарантує, що людина з вадами зору отримає допомогу саме тоді, коли вона буде їй необхідна.

1.3 Основні поняття технології комп'ютерного бачення

На сьогоднішній день кожен смартфон має камеру, і мільйони фотографій надсилаються по мережі кожен день. Щоб отримати більше користі з зображень, мати змогу їх індексувати та шукати за певними параметрами, алгоритмам необхідно розуміти що саме зображено на вхідних даних. На перший погляд, ця проблема може здатись тривіальною. Так вважали і на початку досліджень в галузі комп'ютерного бачення в 1960-х роках.[7] Але десятки років потому дуже багато задач все ще не вирішено та багато досліджень продовжують проводитися без успішних результатів.

Довгий час проблема опису зображень для індексації вирішувалась ручним додаванням метаопису до них, але це незграбне та неповне вирішення питання, що потребує додаткових ресурсів людини. Тому щоб отримати повну інформацію про зображення ми маємо навчити комп'ютер сприймати його та розуміти його зміст. Цю проблему і намагається вирішити комп'ютерне бачення.

Відповідно до визначення даного професоркою Стенфордського університету Фей-Фей Лі, комп'ютерне бачення – це “підрозділ штучного інтелекту, що взаємодіє з наукою того, як наділити машини можливістю “бачити”, тобто аналізувати та розуміти зображення”.[8] Воно також має прямий зв'язок з машинним навчанням і у разі необхідності може використовувати загальні навчальні алгоритми. (див рис. 1.1)

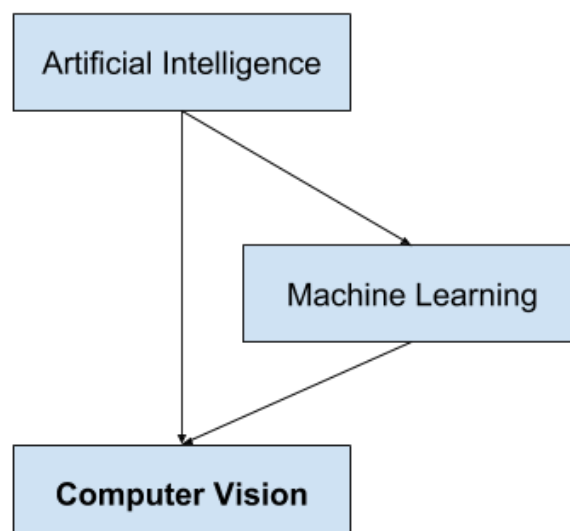


Рисунок 1.1 Зв'язок комп'ютерного бачення з іншими галузями[9]

По суті, задачею комп'ютерного бачення є створення алгоритмів, що аналізують вхідні зображення та витягують корисну інформацію про них.

Сьогодні ця технологія успішно ідентифікує об'єкти та класифікує зображення, але в її основі все ж лежить алгоритм, що порівнює шаблони пікселів, а тому без достатньої кількості даних для тренування, він не буде здатен успішно розпізнати об'єкт.

Розуміння та правильна інтерпретація інформації про зображення також вимагає додаткових знань щодо контексту. Людина сприймає зображення через призму своїх знань про світ та ситуацію в ньому і чітко розуміє різницю між, наприклад, документальним фільмом про нацизм та його пропагандою. В цей самий час існує великий шанс того, що комп'ютер, не розуміючи контексту, заблокує обидва матеріали як пропаганду.

На сьогоднішній день єдине рішення цієї проблеми, це тренувати алгоритми на все більшій та більшій вибірці даних, проте навіть це не захищає від окремих випадків, де алгоритм помилятиметься. Кінцеве вирішення цієї проблеми, скоріше всього, наступить після створення загального штучного інтелекту, який зможе розглядати проблеми таким же чином, як це роблять і люди.

Це створює певні обмеження для застосування та вимагає зусиль щодо підготовки даних для тренування. Проте незважаючи на цю проблему, при правильному використанні, комп'ютерне бачення відкриває багато нових шляхів ефективного вирішення існуючих проблем.

1.4 Особливості створення застосунку для людей з вадами зору

Враховуючи велику кількість користувачів смартфонів з вадами зору сьогодні, створювати додаток з їх врахуванням – це необхідність, а не опція. Може здаватись, що для такої підтримки необхідно витратити багато зусиль розробників, але насправді користуючись загальними гайдлайнами, можна безболісно зробити застосунок таким, що в комбінації з VoiceOver[10] або TalkBack[11] буде доступним і для людей з вадами зору.

Одним із перших кроків при створенні такого застосунку має бути підтримка Dynamic type. Ця функція дозволяє системі автоматично змінювати розміри шрифтів у застосунку. (див. рис. 1.2) Тобто, якщо користувач має поганий зір та потребує більшого шрифту при користуванні смартфоном та додатками, він збільшує розміри шрифту лише у налаштуваннях девайсу. Шрифт також автоматично збільшуватиметься у всіх застосунках, що підтримують Dynamic type. Користувач, що таким чином налаштовує собі систему, буде спантеличений незадовільним розміром шрифтів у застосунку, який не підтримує технологію, і може просто перестати ним користуватися.

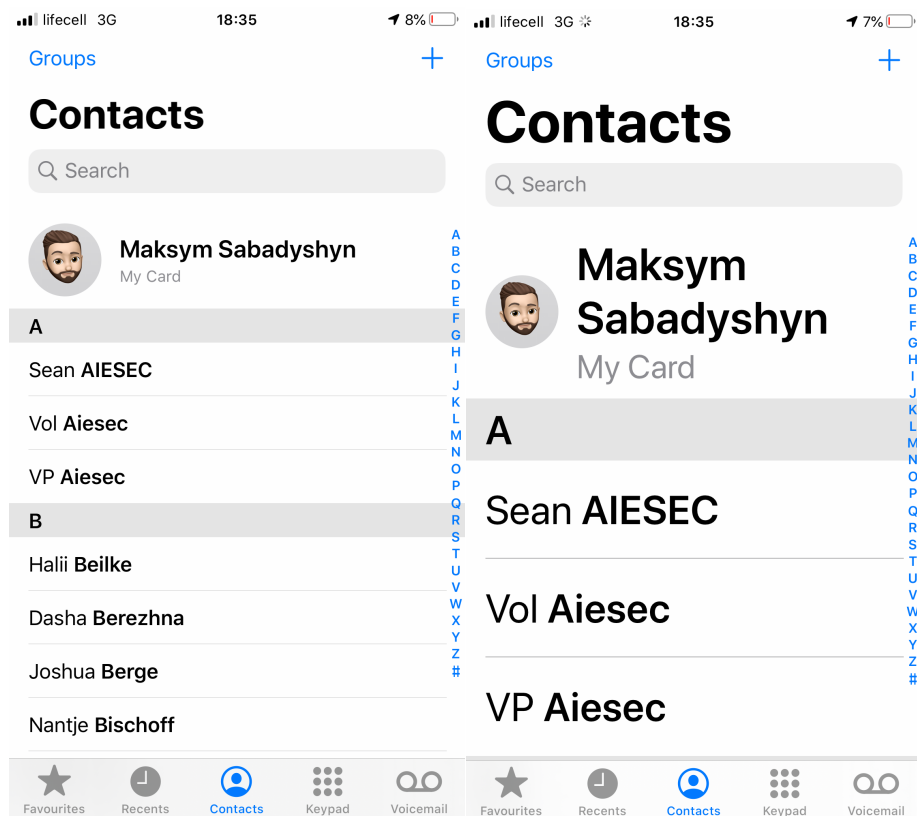


Рисунок 1.2 Порівняння розміру шрифтів з та без Dynamic Type

При розробці додатку для людей з вадами зору також варто звернути увагу на текстові елементи екрану. Необхідно пам'ятати, що вони сприймають контент на екрані через технології такі як VoiceOver[10] або TalkBack[11], що зчитують вміст з екрану, а тому написані тексти мають бути чіткими та зрозумілими. Також візуальні елементи мають супроводжуватися текстовими підказками, що пояснюють їх призначення. Хорошим прикладом є стартовий

екран iPhone, де поле для вводу пароля додатково супроводжується текстом “Touch ID or Enter Passcode” (див. рис. 1.3). Під час використання VoiceOver користувач зможе навести на текст та в аудіо форматі отримати інформацію щодо того, яка дія від нього очікується. За відсутності такого поля висока ймовірність того, що він просто не зрозумів би контексту ситуації.

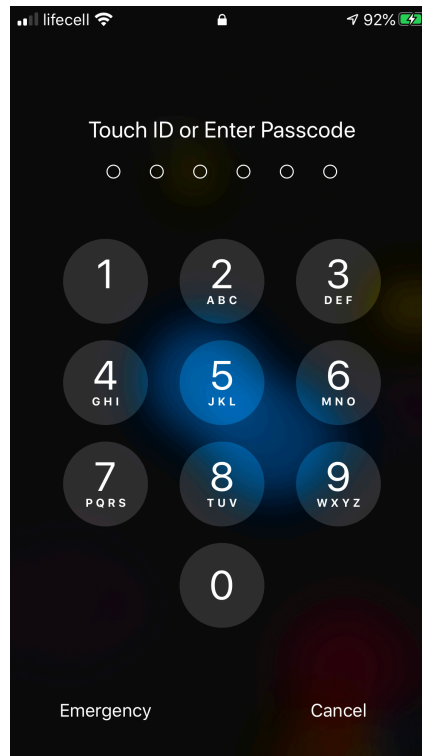


Рисунок 1.3 Приклад правильного супроводу візуального елемента

Також варто звернути увагу на використання заголовків та підзаголовків на сторінках застосунку задля кращої структуризації вмісту та, знову ж таки, для надання контексту людям з порушеннями зору. До прикладу, сторінка з застосунку Apple TV на iOS (див. рис. 1.4), що дає доступ користувачам до ексклюзивної бібліотеки серіалів від Apple.[12] Так як сторінка містить багато елементів з різними категоріями та типами інформації, людям з вадами зору використовуючи VoiceOver було б доволі важко орієнтуватись на ній за відсутності заголовків.

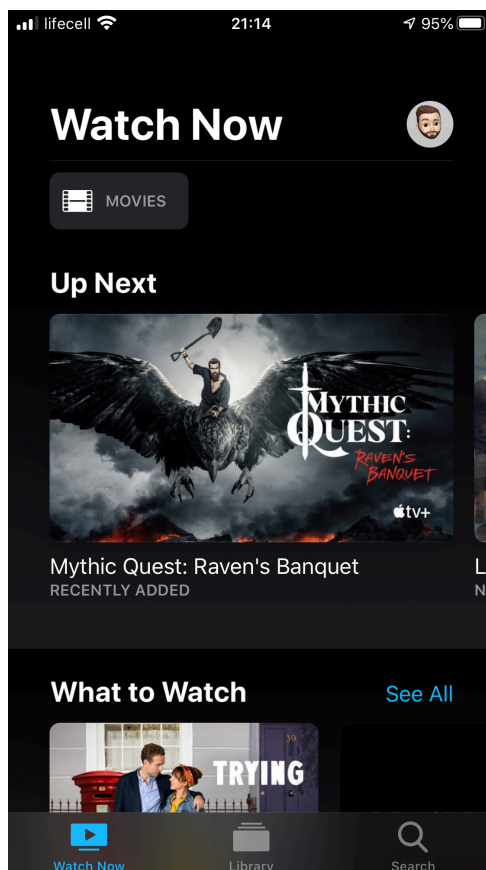


Рисунок 1.4 Основна сторінка Apple TV

Також при створенні додатку з думкою про людей з вадами зору необхідно додавати опис до зображень, або інакшими словами alt-text. Коли користувач з читачем екрану наводить на зображення у якого відсутній опис, він отримує інформацію лише про те, що це зображення, і жодної інформації про те, що на ньому міститься. Саме для того щоб прирівняти досвід користувачів з порушеннями зору до користувачів, які напряду споживають візуальний контент, і необхідно додавати опис. Тоді незалежно від ситуації всі будуть розуміти що саме знаходиться на зображенні, і відповідно загальний контекст.

Також не варто використовувати лише кольори задля того, щоб показати результат якоїсь дії. Наприклад, при введенні неправильного паролю підкреслювати поле червоним, або при успішній операції змінювати колір кнопки на зелений. Така поведінка в додатку не надає жодної корисної інформації не тільки людям з втратою зору, а також і людям з кольоровою сліпотою. Для виправлення такої поведінки необхідно доповнювати кольорові позначення результату дії текстом. Яскравий приклад – Google з їх поштовим клієнтом Gmail. Вони використовують кольорове підсвічування, але додають

альтернативний варіант у вигляді тексту під полем для вводу. (див. рис. 1.5) Таким чином користувач з порушеннями зору зможе отримати інформацію про неправильно введений пароль за допомогою технологій зчитування тексту з екрану.

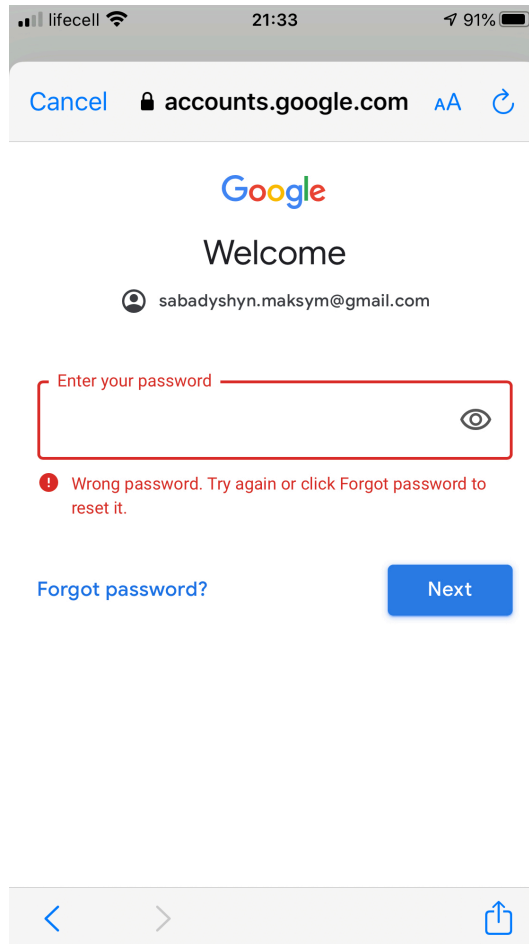


Рисунок 1.5 Правильна ідентифікація результату дії на прикладі Gmail

Ще один важливий елемент додатку – це звукова комунікація з користувачем. Такий підхід є дуже корисним для застосунків, які розроблені спеціально для користування людьми з вадами зору. Він передбачає окремі сценарії взаємодії, і вони не завжди будуть зручними людям без порушень зору, що сприймають інформацію переважно візуально. Звуком позначають перехід на нову сторінку, успішну автентифікацію, оновлення інтерфейсу. Така взаємодія з користувачем, що має проблеми з зором, значно полегшить йому роботу з додатком та зробить окремі моменти роботи простими та зрозумілими.

1.5 Висновки до розділу 1

У даному розділі було розглянуто основні проблеми людей з вадами зору у побуті. Також було знайдено та проаналізовано існуючі застосунки для вирішення деяких з цих проблем. Крім того, було розглянуто поняття комп'ютерного бачення, зв'язки технології з галузями машинного навчання та штучного інтелекту, а також основні проблеми над вирішенням яких зосереджені фахівці сьогодні. В кінці розділу було розглянуто кращі методики щодо створення застосунку, яким було б зручно користуватись людям з вадами зору.

Розділ 2. Теоретичні відомості щодо використаних технологій

2.1 Аналіз особливостей розробки під iOS

З шаленим розвитком смартфонів і зі збільшенням їх потужності, можливості програмного забезпечення також зростають. Сьогодні архітектура програмного забезпечення для мобільних пристроїв – комплексна, та дозволяє створювати масштабні застосунки, які попри велику кількість процесів швидко працюють на смартфонах та планшетах. Для того, щоб застосунок під мобільні пристрої оптимально працював в умовах пікових навантажень, а також задля легкої підтримки коду та відсутності критичних помилок, варто дотримуватись певних патернів та рекомендацій під час створення застосунку, а також продумувати загальну структуру проєкту ще до початку його написання.

Для різних платформ рекомендації можуть відрізнятись, навіть iOS і Android попри те, що працюють на смартфонах, мають дещо різні підходи до розробки. Наприклад, кількість різних видів девайсів під управлінням iOS набагато менша, ніж під Android, а тому адаптувати дизайн та роботу застосунку під iOS значно легше. Також на iOS суттєво краща ситуація з кількістю пристроїв на нових версіях ОС. Це напряду впливає на розробників, адже певні програмні нововведення стають доступними лише на нових версіях. Для порівняння, останні дві версії iOS встановлені на 94% сумісних пристроїв[13], в той час як останні дві версії Android мають 54.5% користувачів.[14]

Програмуючи під iOS необхідно визначитись з загальною структурою програми. Apple вказує, що рекомендований патерн дизайну iOS застосунку – MVC.[15]

MVC, де М – це модель (Model), V – це представлення (View), а С – це контролер (Controller) – це патерн дизайну, що розділяє класи програми на попередньо вказані 3 категорії. Він вказує не лише ролі об'єктів у застосунку, а також і те, як вони взаємодіють між собою.[15] (див. рис. 2.1)

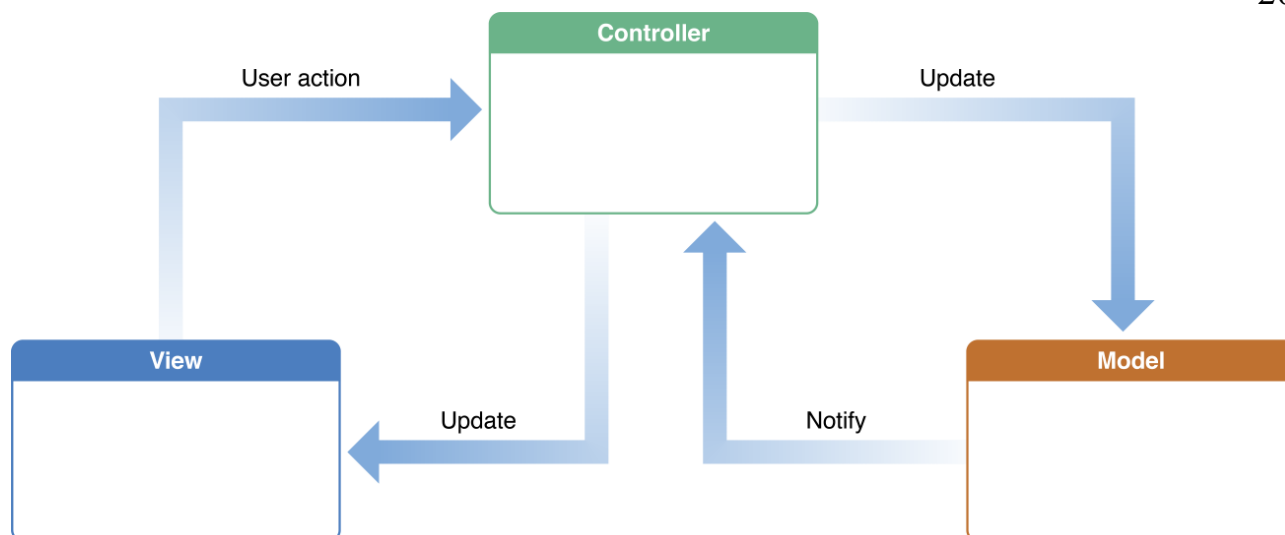


Рисунок 2.1 Візуалізація патерна MVC[15]

Він доволі простий в розумінні і не вимагає особливих знань для використання.

Модель відповідає за зберігання даних та операції над даними. Наприклад, вона може представляти студента. В такому випадку в ній будуть зберігатись дані про нього, як от ім'я, масив дисциплін, оцінки, а також можливі методи для роботи з ними. Наприклад, обрахування середнього балу, або знаходження максимальної чи мінімальної оцінки. Модель не має жодних зв'язків з представленням, і загалом не знає про нього та його елементи нічого. У разі змін в моделі, вона повідомляє про них контролер.

Представлення відповідає за взаємодію з користувачем. Воно містить елементи користувацького інтерфейсу та реагує на натиснення клавіш або ж інші дії користувача. Інформацію щодо його дій воно передає до контролера. Так як і модель нічого не знає про представлення, саме представлення також не знає нічого про модель і має зв'язок лише з контролером.

Контролер – це об'єкт, що з'єднує між собою представлення та модель і знаходиться, по суті, посередині між ними. Він отримує інформацію по діям користувача та приймає рішення щодо того як на них реагувати. Разом з цим він налаштовує та оновлює інтерфейс, використовуючи дані отримані з моделі. Також контролер викликає окремі методи з моделі за потреби та отримує повідомлення про зміни в ній. Сам контролер не повинен містити жодної логіки щодо керування даними, як от вивантаження даних з серверу чи обрахування інформації, так як це є обов'язком моделі.

Така незалежність об'єктів один від одного дозволяє легко розширювати окремі елементи застосунку без пошкоджень інших та підтримувати існуючий функціонал.

Звичайно, крім MVC є ще і інші патерни, як от MVVM, MVP, та VIPER. Навіть MVC, який рекомендований Apple, відрізняється від традиційної моделі яка навряд прижилася б в мобільній розробці (див. рис. 2.2). кожен з цих патернів має свої переваги та недоліки, та MVC все ще залишається основним та рекомендованим при розробці iOS застосунку.

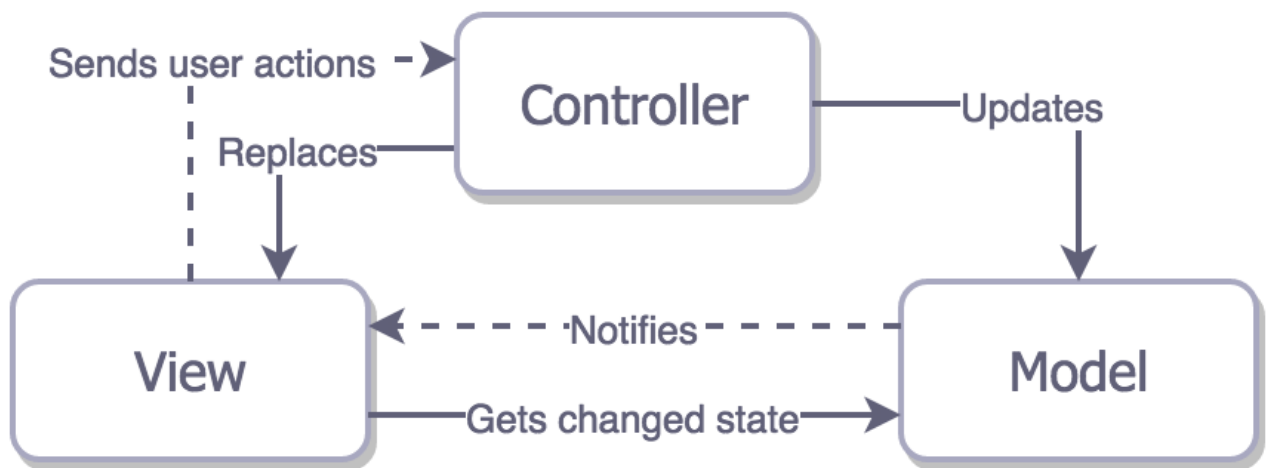


Рисунок 2.2 Традиційний MVC

Ще одним інструментом, що заслуговує на увагу та постійно використовується при розробці iOS застосунків, є делегування. Воно забезпечує комунікацію між об'єктами в зручний та безпечний спосіб. При делегуванні клас віддає керування певною своєю логікою іншому класу, який і буде делегатом. При використанні цього патерну делегат може викликати методи делегатора без жодної створеної події.

При розробці під iOS за допомогою мови програмування Swift це досягається за використання протоколів, що надають список необхідних методів, властивостей, та будь-яких інших вимог, які має реалізувати потенційний делегат, та про які має інформацію в разі бажання виклику делегатора. (див. рис. 2.3)

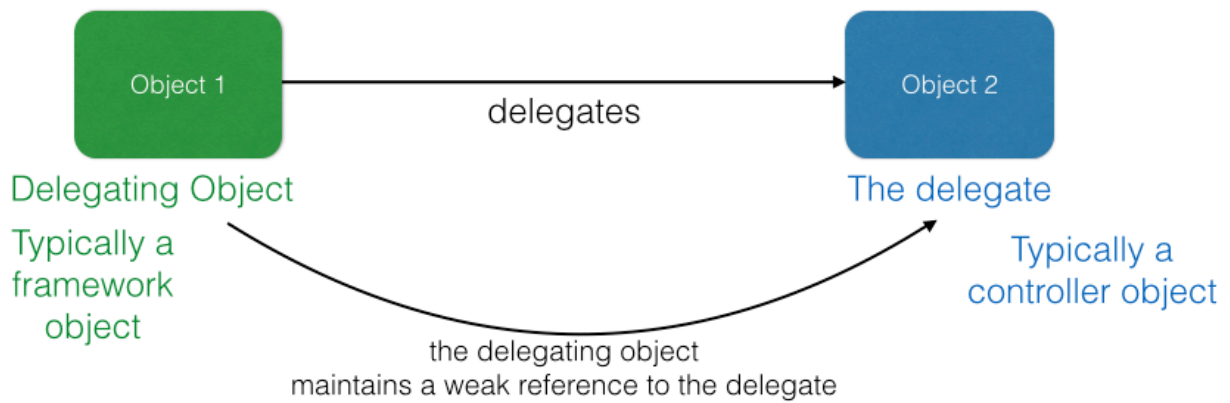


Рисунок 2.3 Принцип патерну делегування[16]

Маючи такий спосіб зв'язку між класами досягається хороша архітектура та уникається багато зайвого коду при підтримці різних сценаріїв роботи застосунку.

2.2 Загальна інформація про Swift як мову програмування

Не звертаючи увагу на різні можливості в кросплатформенній розробці, які, звичайно, хоч і мають певні переваги, та мають і доволі багато суттєвих недоліків, на сьогоднішній день існує 2 нативні шляхи розробки iOS застосунків – використовуючи Objective-C як мову програмування, або ж Swift. Objective-C був, як можна зрозуміти з назви, створений на основі C, а свої об'єктно-орієнтовані риси він отримав від мови програмування SmallTalk. Він був розроблений набагато раніше, ніж Swift, в 1984 році, та має дещо перевантажений та застарілий на вигляд синтаксис. На сьогоднішній день випуск нових версій Objective-C не відбувається.

Swift, на відміну від Objective-C – це сучасна мова програмування, представлена Apple на щорічній конференції для розробників WWDC в 2014 році як поступова заміна Objective-C. Синтаксис, представлений в Swift, набагато легший та сучасніший, аніж в свого попередника. (див. рис. 2.4) Swift потребує менше зайвого коду та читається майже як звичайний текст англійською мовою, в той час як Objective-C відлякує дещо заплутаним синтаксисом.

```

1 @interface MyClass
2
3 - (void)sayHello;
4
5 @end
6
7 @implementation MyClass
8
9 - (void)sayHello{
10     NSLog(@"Hello Swift,
11     Goodbye Obj-C!");
12 }
13 @end
  
```

```

1 class MyClass {
2     func sayHello() {
3         print("Hello Swift,
4         Goodbye Obj-C!")
5     }
6 }
  
```

Рисунок 2.4 Порівняння синтаксису Objective-C(зліва) та Swift(справа) [17]

Swift – об’єктно-орієнтована мова програмування, тобто в ньому присутні всі типові для ООП властивості: поліморфізм, інкапсуляція, наслідування, та абстракція. Також варто звернути увагу на те, що Swift – статично типізований, а тому помилки невідповідності типів виявляються ще на етапі компіляції.

Крім того, Swift має автоматичне керування пам’яттю, яке реалізоване через ARC – Automatic Reference Counter. Він автоматично вивільнює пам’ять тоді, коли об’єкти перестають використовуватись і в більшості випадків розробникам не потрібно думати за це. Загально кажучи, система працює на основі того, що кожен раз при створенні об’єкту рахунок посилань на нього збільшується на 1. В момент, коли рахунок посилань знову стане 0, об’єкт буде вивільнено з пам’яті, адже він не буде нікому потрібен. Звичайно, є випадки, коли система не працює так як треба. Найчастіше зустрічається так званий strong reference cycle, коли два об’єкта посилаються один на одного і таким чином жоден з них не буде звільнено з пам’яті. Розробнику необхідно передбачувати такі ситуації та писати з код з оглядом на їх уникнення.

Також варто згадати і про опціонали, реалізація яких є у Swift. Опціонали (optionals) – являють собою інструмент обробки ситуацій, коли у змінної може не бути значення. Реалізовано це як перелік з двома можливими випадками – none, та some(Wrapped), де Wrapped – тип, що зберігається в опціоналі. У випадку none – значення просто немає. За допомогою опціоналів

уникається багато помилок в рантаймі, а синтаксис програми стає значно лаконічнішим, ніж якби він містив перевірки на nil.

Swift, крім всього, мова програмування з відкритим вихідним кодом з 2015 року. Це дає йому ще більшу підтримку спільноти, пришвидшений пошук помилок та багів, а також інтенсивніший розвиток, адже кожен зацікавлений фахівець може зробити свій внесок у розвиток мови. Цією можливістю користуються і великі компанії, зацікавлені у розвитку Swift. Google, до прикладу, має свою гілку в репозиторії проєкту, де працює над розвитком Swift у галузі диференційованого програмування.

2.3 Використання фреймворку Vision для застосування технології комп'ютерного бачення в iOS застосунках

Враховуючи бурхливий розвиток галузі комп'ютерного бачення не дивно, що створюється багато фреймворків для зручної роботи з технологією. Не залишилась в стороні і Apple, яка для розробників під свою платформу створила фреймворк Vision.

Vision був представлений на WWDC в 2017 році. Після цього кожного року він оновлювався та доповнювався новими функціями. Тому на сьогоднішній день - це сучасний фреймворк зі зручним синтаксисом.

Vision дозволяє застосовувати алгоритми комп'ютерного бачення, щоб виконувати різні завдання на вхідних зображеннях та відео.[18] Це, по суті, надбудова над більш низькорівневим фреймворком CoreML, яка зосереджується лише на виконанні завдань пов'язаних з комп'ютерним баченням. Його функціонал покриває більшість типових задач, що потребують використання комп'ютерного бачення, а саме: знаходження обличчя, розпізнавання тексту, слідкування за об'єктом по кадрах відео або фото, знаходження штрих-кодів. Також Vision має функціонал знаходження рис обличчя. Для задач таких як класифікація або ж розпізнавання об'єктів у Vision є можливість використовувати власні CoreML моделі.

Загалом робота з Vision складається з використання трьох основних елементів: запиту (Request), обробника запиту (RequestHandler), та результату виконання запиту (Observation) (див. рис. 2.5).

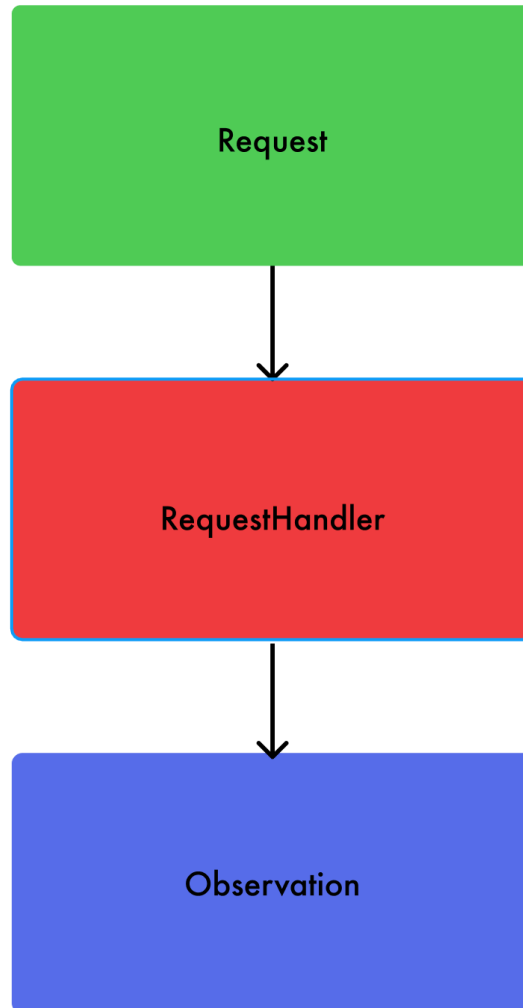


Рисунок 2.5 Загальна структура взаємодії з Vision

Request використовується для налаштування запиту, а також для вказування інструкцій по роботі після отримання результатів. На початку створюється екземпляр класу, що наслідується від базового абстрактного класу VNRequest. Існує багато видів запитів, що відрізняються задачами, які вони виконують. До прикладу, VNDetectBarcodeRequest, що являє собою запит для знаходження штрих-кодів, або ж VNRecognizeTextRequest, що відповідає за запит для знаходження тексту.

RequestHandler відповідає за виконання запитів. Загалом, є два типи RequestHandler – VNImageRequestHandler та VNSequenceRequestHandler. Їх відмінність полягає в тому, що VNImageRequestHandler на вхід отримує запит з

окремим зображенням, в той час як `VNSequenceRequestHandler` отримує запит з серією зображень. `VNSequenceRequestHandler` використовується коли необхідно, наприклад, відслідковувати об'єкт після його ідентифікації.

Після додавання запиту до проекту створюється об'єкт, що наслідується від базового `VNRequestHandler`. У обробник запитів ми передаємо масив з наслідниками `VNRequest`. Після цього у методі, що відповідає за отримання результатів виконання ми обробляємо отримані дані як об'єкти наслідників `VNObservation` та продовжуємо роботу з ними.

2.4 Використання фреймворку *Speech* для розпізнавання голосових команд користувачем

`Speech` – ще один фреймворк від Apple, мета якого виконувати розпізнавання в реальному часі або ж попередньо записаного людського мовлення. [19] Він був представлений в 2016 році і після цього оптимізовувався та отримував новий функціонал. На даний момент, фреймворк підтримує та розпізнає більше, ніж 50 різних мов та діалектів. В основі `Speech` лежать ті ж технології, що використовуються в Siri[20] або Dictation[21].

Варто також зауважити, що офлайн розпізнавання доступне лише для основних мов, в той час як робота з іншими мовами відбувається після надсилання запиту на сервери Apple та отримання результату. Тому якщо програма підтримує декілька мов, то розробникам необхідно забезпечити перевірку на підключення до інтернету.

Для більшої точності отриманих результатів фреймворк підлаштовує результати індивідуально до користувача, використовуючи для цього дані зі списку контактів, список встановлених застосунків, та іншу потенційно корисну інформацію. При цьому самому розробникові не потрібно писати жодної додаткової стрічки коду, це працює відразу.

Загалом, для розпізнавання мови в реальному часі спочатку створюється запит (`SFSpeechAudioBufferRecognitionRequest`), який потім віддається розпізнавачеві (`SFSpeechRecognizer`). Від розпізнавача повертається

задача, і у завершальному блоці задачі ми обробляємо отриманий нами результат. (див. рис. 2.6)

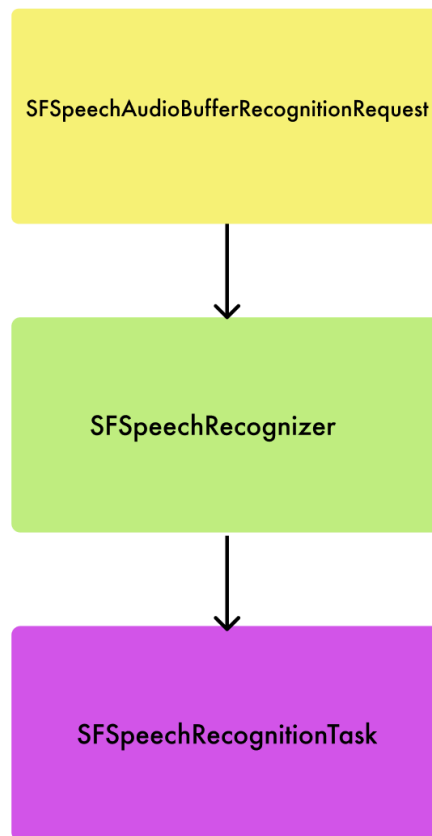


Рисунок 2.6 Схема роботи з фреймворком Speech

На жаль, на сьогоднішній день Speech не має способу роботи з мікрофоном напряду, а тому для імплементації розпізнавання мовлення в реальному часі необхідно також використовувати AVAudioEngine, відповідальністю якого якраз таки і є робота з вводом аудіо від мікрофона. Якщо ж розробник вже має аудіо файл, який необхідно розпізнати, то замість SFSpeechAudioBufferRecognitionRequest необхідно використати SFSpeechURLRecognitionRequest.

Розробникам з поганими намірами не вдасться використовувати функціонал Speech на свою користь без відома користувача, так як перед використанням його необхідно запитати про 2 дозволи: дозвіл на зчитування вводу з мікрофону, а також дозвіл на розпізнавання мовлення. Це, звичайно, не унеможливилює всі способи зловживання технологією, але гарантує те, що користувач матиме повну інформацію про те, які права отримує застосунок і чим він може користуватися.

2.5 Загальні відомості про SQLite

Необхідність зберігати дані користувача була завжди, але зараз вона як ніколи актуальна. Даних стає все більше, і вони займають все більше пам'яті, а на плечі розробників лягає відповідальність щодо прийняття рішень, які стосуються де зберігати потрібні дані та як саме організувати доступ до них.

Існує дуже багато різних СКБД, популярними також стають нереляційні бази даних, де інформація зберігається не у форматі таблиця-відношення. Але пальму першості у більшості випадків все ще займають типові реляційні бази даних. Однією з таких і є SQLite.

SQLite – це вбудована система керування базами даних, яка сьогодні користується високим попитом, особливо у мобільних розробників. Ця СКБД дуже зручна та не потребує сервера для зберігання – вона займає лише один файл на диску. Тому при налаштуванні розробникові не потрібно перейматись за налаштування з'єднань до сервера, лише вказати шлях до файлу з базою даних та працювати з запитам до неї.

Також SQLite – це СКБД з відкритим вихідним кодом, а тому спільнота користувачів може надавати свої пропозиції та вирішувати поточні проблеми в коді так само, як це робить і основна команда проєкту. В такому випадку набагато менша ймовірність натрапити на помилки чи баги при роботі з СКБД, аніж у випадку з СКБД, де кодова частина є приватною, адже там тільки команда проєкту займається його просуванням. Також така підтримка спільноти виражається в тому, що для SQLite написано багато допоміжних застосунків для керування нею. Ще одним аргументом за надійність SQLite є те, що перед випуском нової версії СКБД проходить через приблизно 1.7 мільйонів тестів, а покриття коду ними складає 100%.[22]

Відомо, що SQL має багато стандартів, повне дотримання яких не є обов'язковим. Тому багато розробників СКБД не реалізують всі необхідні по стандарту вимоги. Це тільки додає проблем користувачам СКБД, адже виникає потреба додатково шукати, які саме функції доступні, а які – ні. На відміну від них, SQLite, що також використовує SQL, дотримується стандартів, та

намагається мати повний набір функцій, хоч і не додаючи при цьому окремих своїх. Тобто користуючись SQLite не треба розраховувати, що будуть доступні складні механізми роботи з СКБД, які зроблені поза стандартом, але можна бути впевненим в тому, що всі функції стандарту SQL 92 є доступними в СКБД.

2.6 Висновки до розділу 2

В другому розділі було розглянуто теоретичну складову використаних у практичній частині проекту технологій. Було проаналізовано особливості розробки застосунків під iOS, кращі техніки та важливі моменти в реалізації. Також було розглянуто Swift, сучасну мову програмування створену Apple на заміну Objective – C, її особливості та переваги.

Окремо були розглянуті два важливі фреймворки – Speech та Vision і їх задачі, проаналізовано алгоритми їх застосування та переваги використання. В кінці розділу було також розглянуто питання зберігання інформації в бази даних та надано відомості про реляційну СКБД SQLite, її переваги та особливості.

Розділ 3. Опис реалізації застосунку

3.1 Аналіз технічного завдання

Технічне завдання складається з розробки мобільного застосунку під iOS для допомоги людям з вадами зору. Як складова додатку, має бути використана технологія комп'ютерного бачення.

Після аналізу предметної області та пошуку ключових проблем, з якими стикаються люди з вадами зору, було виділено дві ключові – проблеми з читанням написаного, а також відсутність можливості розпізнавати кольори. Вирішення цих двох проблем і лягло в основу розробки застосунку, являючись основним функціоналом.

Необхідно було також проаналізувати та створити інклюзивний дизайн, зручний для користування людям з вадами зору, які і є цільовою аудиторією застосунку. З цього випливає, що він не потребував складних візуальних рішень, а мав бути простим та зрозумілим, і в парі з VoiceOver мав створювати зручну взаємодію з елементами інтерфейсу для користувачів.

Додатковим завданням було створення основних голосових команд, які дозволили б швидко переходити до основних елементів меню у випадку, якщо користувач не хоче скролити в пошуках потрібного елемента.

Доповнене технічне завдання також включало завдання реалізувати збереження прочитаних текстів та збережених кольорів в базу даних за бажанням користувача. Крім того, база даних мала б містити основні кольори, з якими порівнювався б колір, що знаходиться на процесі розпізнавання.

3.2 Пояснення вибору використаних технологій

В даному проєкті було багато задач, що вимагали використання різних технологій, вибір яких не завжди був очевидним.

Для розпізнавання тексту було обрано фреймворк Vision, адже це власний фреймворк від Apple. Він надає розробникам дійсно повний інструментарій роботи з технологією комп'ютерного бачення через зрозумілий високорівневий API. Окремо варто зазначити, що API, який надається для

користування, створено з оглядом на те, як побудований синтаксис Swift, а тому всі виклики та механізми роботи не мають лишнього коду та повністю зрозумілі для розробників, що працюють з цією мовою програмування.

Ще однією перевагою є те, що розпізнавання елементів відбувається швидко. Починаючи з ЦП A11 Bionic, всі мобільні девайси від Apple мають на кристалі ще один додатковий процесор. Він був названий Neural Engine та задіюється в операціях, які стосуються машинного навчання, що пришвидшує роботу алгоритмів. Також необхідно пам'ятати, що Vision – це власний фреймворк Apple, а тому він повністю оптимізований під роботу на iOS/iPadOS та має повну підтримку і оновлення від компанії.

Очевидним плюсом Vision є також його безпека. Як вже було сказано раніше, це надбудова над CoreML, однією з особливостей якої є те, що всі запити відбуваються локально на девайсі. Тобто весь процес від створення запитів до отримання результатів ізольований від зовнішнього середовища, а інформація користувачів нікуди не передається. Це, крім того, додає приріст у швидкості, адже не потрібно робити запитів по мережі.

Крім того, плюсом локальності процесу є те, що програмою можна користуватись офлайн. Це може значно зекономити трафік користувачам та кошти розробникам, адже немає необхідності зберігати модель на хмарному сховищі або використовувати один з вже готових сервісів в хмарі.

Для реалізації розпізнавання голосових команд було обрано фреймворк Speech. Загалом, аргументація вибору досить схожа до аргументації вибору Vision. Speech – це також фреймворк від Apple, його API лаконічне та не створює непорозумінь у розробників. Вся інформація, навіть якщо вона передається на сервери Apple, передається зашифрованою, а тому гарантується особистість користувачів.

Для зберігання інформації було обрано СКБД SQLite. Загалом, у проєкті не було складних сценаріїв використання баз даних, а тому SQLite було обрано за її компактність та відсутність лишніх додаткових налаштувань. Вона повністю влаштовувала по вимогам, а присутність на ринку додаткового програмного забезпечення для керування базою даних SQLite, велика база

користувачів та популярність використання SQLite у якості рішення для мобільних застосунків тільки підкріпило впевненість у виборі цієї СКБД.

Так як SQLite не входить до стандартного пакету бібліотек для розробки під девайси Apple, його необхідно було додатково підключити. Для підключення SQLite я використав Swift Package Manager.

Це – відносно новий менеджер залежностей, що розроблений компанією Apple і підтримка якого вбудована напряду в XCode.[23] А тому підключення в такий спосіб має свої переваги над конкурентами типу CocoaPods чи Carthage.

Для підключення за допомогою технологій конкурентів необхідно було б використовувати консоль та робити багато лишніх дій. Наприклад, для того щоб підключити SQLite, та і взагалі будь-яку іншу залежність через CocoaPods необхідно окремо підключати CocoaPods до проєкту, у створеному pod файлі прописувати бажану залежність, а потім виконувати команди в консолі для синхронізації поточного стану файлу з встановленими залежностями або ж робити це все окремо тільки через консоль. Сценарій встановлення залежностей через Carthage теж є доволі схожим.

У випадку ж з Swift Package Manager все набагато простіше. У інтерфейсі XCode необхідно увімкнути застосування Swift Package Manager, а потім через графічний інтерфейс пошуку знайти та додати бажаний пакет. Всі варіанти додавання пакету у SPM прописані у графічному інтерфейсі, а тому не вимагають запам'ятовування і здаються набагато простішими для використання. У подальшому використанні, слідкування за версіями доданих пакетів також відбувається через XCode та не містить в собі жодних складнощів.(див. рис. 3.1)

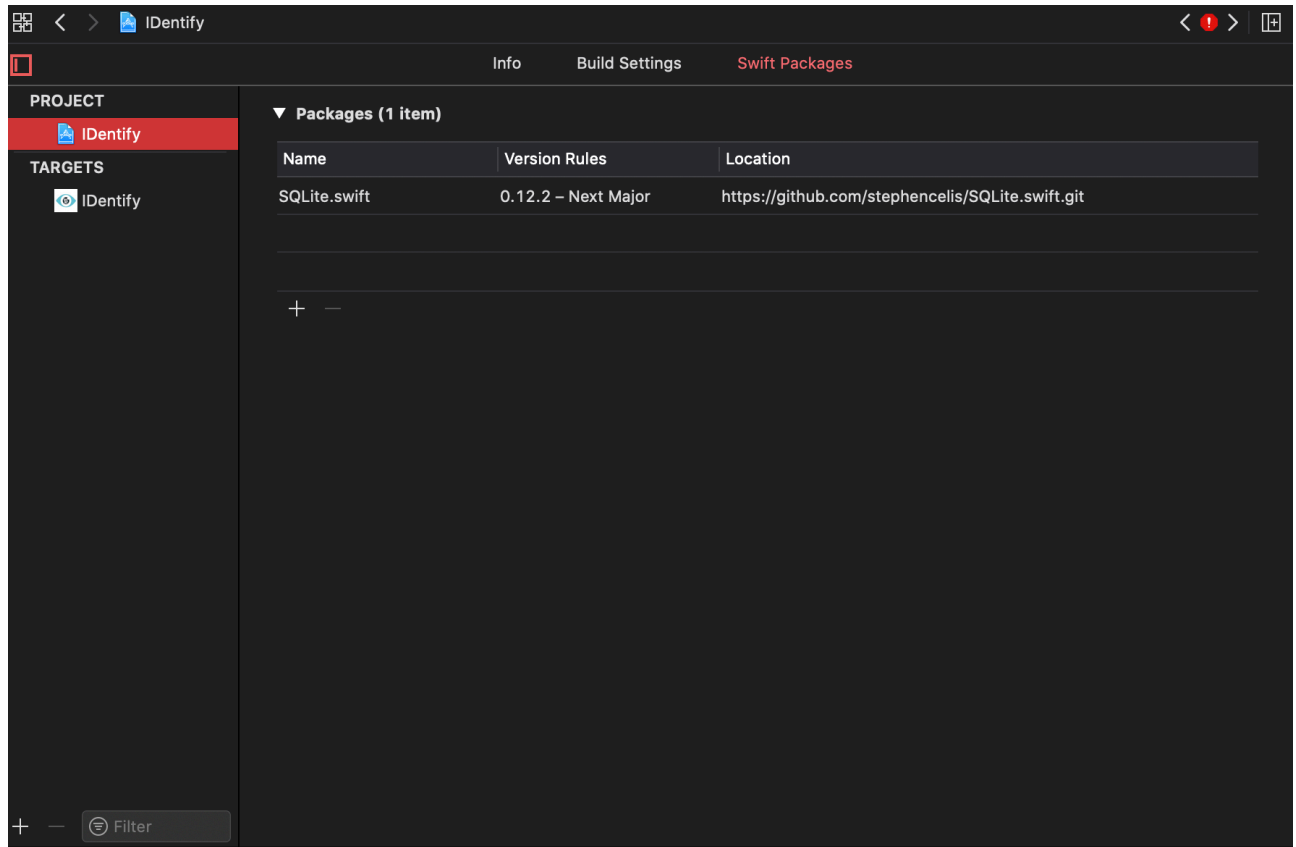


Рисунок 3.1 Демонстрація переліку доданих пакетів SPM у Xcode

Враховуючи таку простоту використання Swift Package Manager та його повну інтеграцію в XCode, було обрано саме цей інструмент для подальшого керування залежностями в проєкті.

3.3 Опис та пояснення вибору середовища розробки

Як середовище розробки було вибрано XCode – IDE створену Apple. XCode – це інтегроване середовище розробки для використання для розробки застосунків для продуктів Apple.[24] Це комплексне середовище, що містить всі необхідні інструменти для всього циклу розробки, від створення проєкту і до завантаження його у фірмовий магазин додатків App Store.

Дизайн XCode слідує принципу одновіконного інтерфейсу, а тому простір з кодом, консоль дебагу, структура проєкту, а також детальна інформація про обраний елемент знаходяться в одному вікні. (див. рис. 3.2)

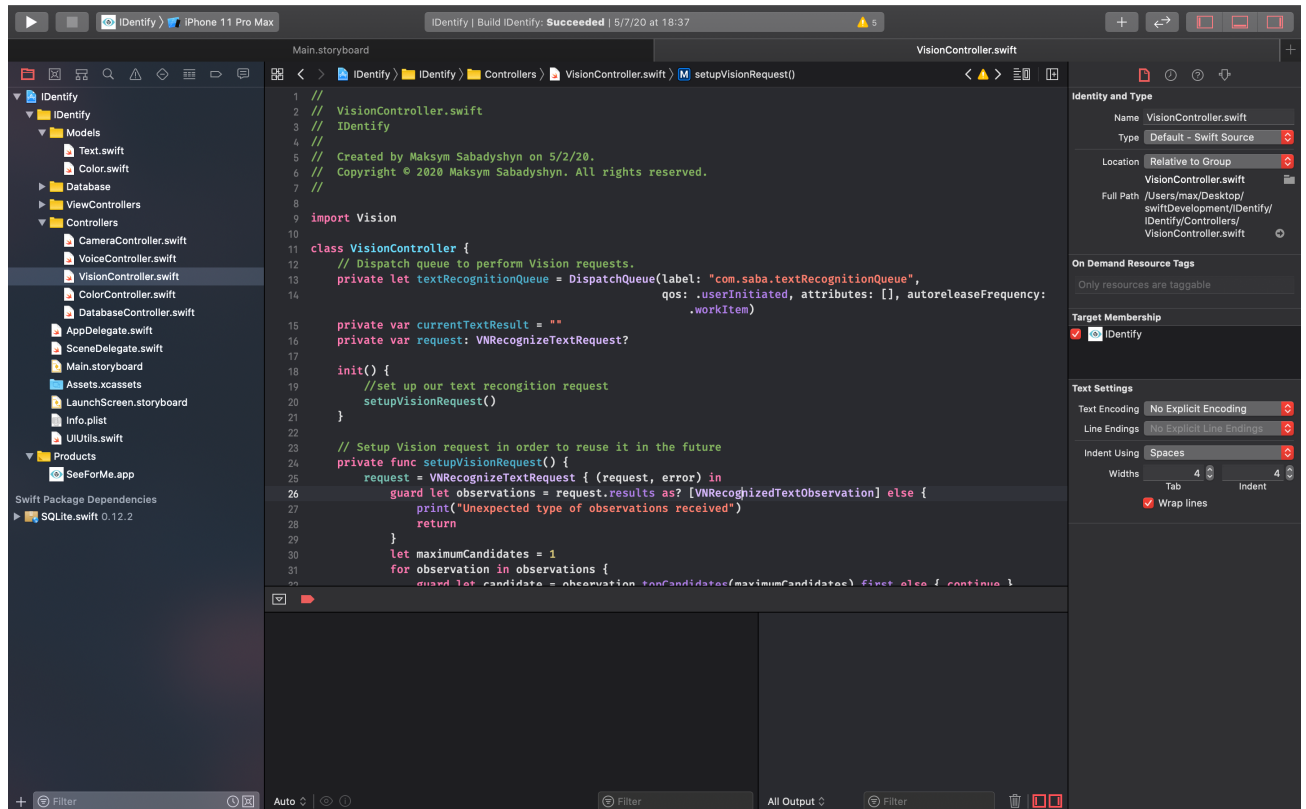


Рисунок 3.2 Вікно XCode

Це надає швидкий доступ до всіх потрібних елементів. Звичайно, за бажанням, певні вікна можна згортати і таким чином збільшувати площу інших.

Як і всі сучасні редактори, XCode має підтримку Git, але розробники з Apple додатково створили зручний користувацький інтерфейс для роботи з ним. XCode має окреме вікно, де вказуються всі існуючі гілки проєкту, та коміти, що зроблені в них. Також існує зручна навігація та інформація по тому, де саме в проєкті та у файлі були зроблені ці зміни. Крім того, робити Git-пов'язані дії можна прямо з XCode без використання консолі чи сторонніх застосунків. Всі вище зазначені факти значно полегшують роботу з системами контролю версій та надають змогу відмовитись від використання додаткових програм.

XCode також має вбудований редактор storyboard файлів. Storyboard файл відповідає за зберігання та створення інтерфейсу користувача. Завдяки простоті інтерфейсу, в XCode дуже просто створювати нові інтерфейси та редагувати вже існуючі.(див. рис. 3.3)

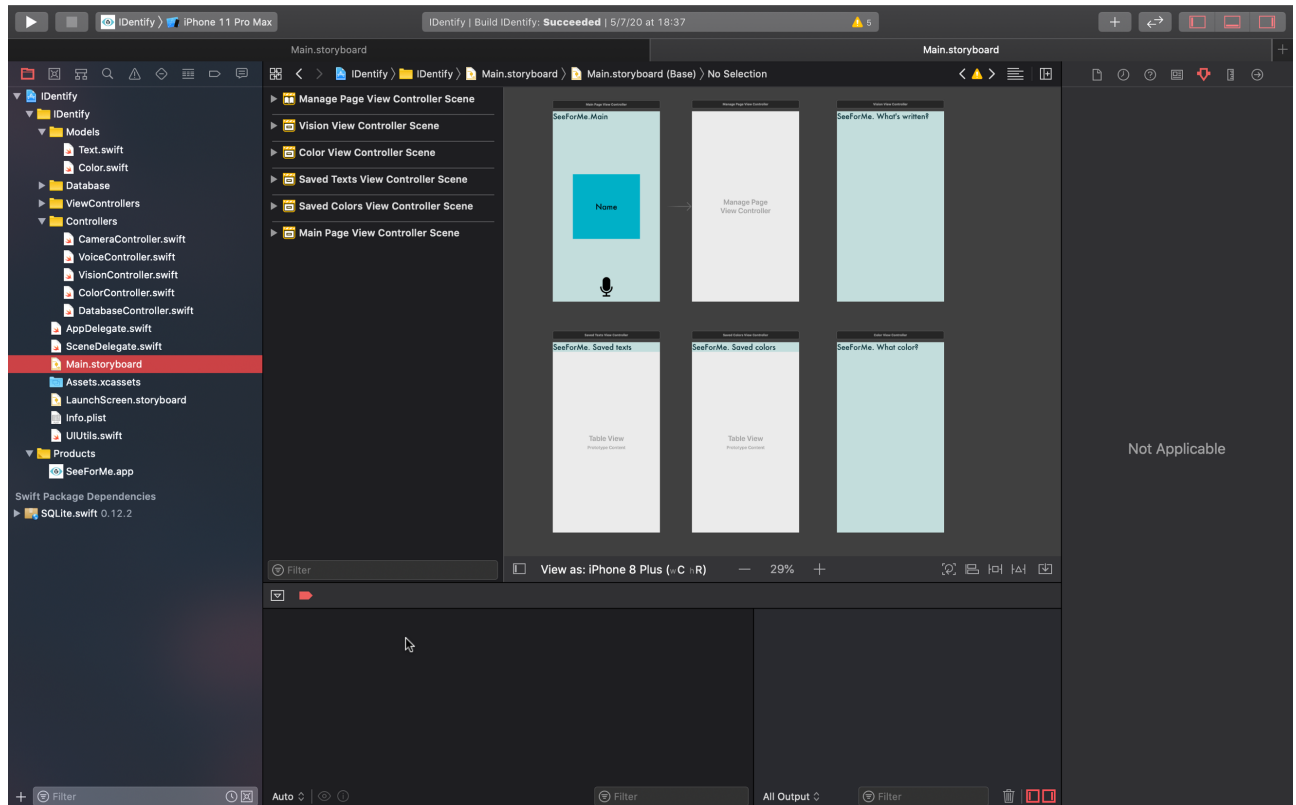


Рисунок 3.3 Інтерфейс редактору Storyboard в XCode

Одним з найважливіших етапів розробки застосунку є його тестування. Для цього в XCode є напевне найбільший інструментарій. Можна тестувати додаток на реальному девайсі, просто підключивши його до комп'ютера з проектом, але також можна використати симулятор. Симулятор – це емулятор девайсів Apple. В симуляторі доступні всі можливі актуальні девайси, а тому можна протестувати працездатність свого застосунку на багатьох різних апаратах, фактично не володіючи жодним. Кожен емульований девайс практично нічим не відрізняється від реального, а також дає можливості тестування різних ситуацій. До прикладу, таких як от обертання девайсу, тряска, або ж симулювання певної локації.

Враховуючи таку велику кількість переваг, а також той факт, що XCode розроблений Apple, має постійну підтримку та отримує оновлення, це середовище розробки стає чудовим варіантом для вибору під час створення iOS застосунку.

3.4 Опис та обґрунтування архітектури проекту

Для розробки застосунку був використаний рекомендований Apple патерн MVC. Він був обраний через гнучкість архітектури, можливість створювати нові моделі та вносити зміни в існуючі ізольовано від контролерів та не впливаючи на загальну працездатність застосунку. Також перевагу було надано MVC через відсутність складних архітектурних рішень при розробці додатку опираючись на цей патерн. А його популярність призвела до великої кількості документацій та інформації щодо того, які типові помилки не варто допускати та як саме правильно розробити застосунок використовуючи такий підхід.

У застосункові Storyboard файли є файлами представлення, вони містять в собі елементи інтерфейсу та зв'язані з контролерами, ViewController файли мають роль контролерів та зв'язують представлення з моделями, які оперують даними та повертають результати назад. За аналогією рисунку 2.2 наведено схему відповідностей класів ролям в патерні MVC (див. рис. 3.4)

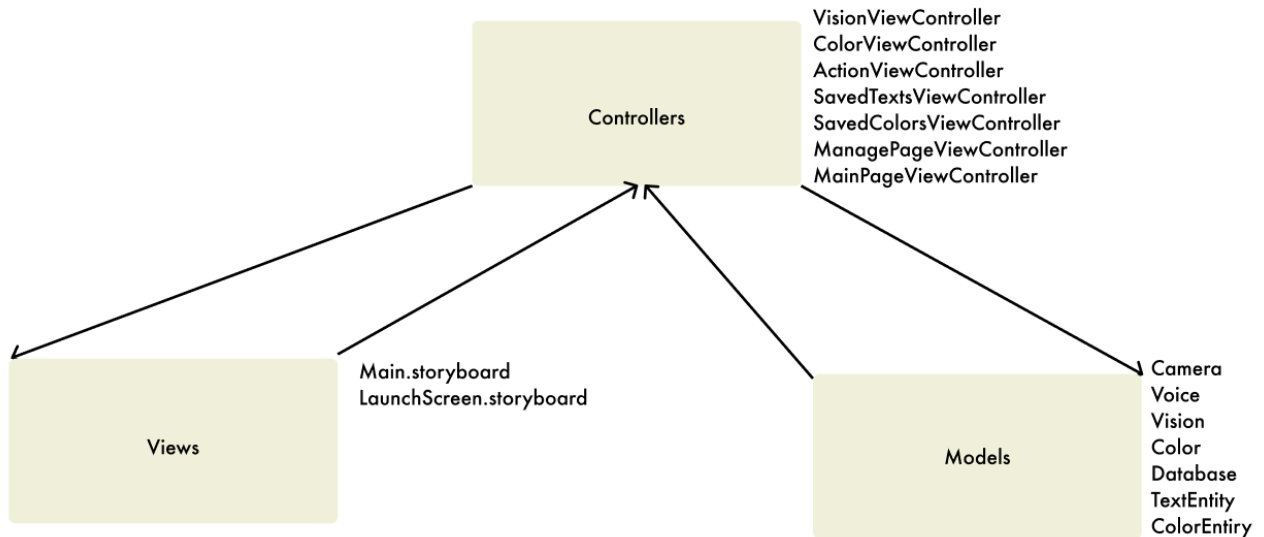


Рисунок 3.4 Розподіл класів щодо патерну MVC в розробленому застосунку

3.5 Обґрунтування дизайну та взаємодії з користувачем

Як вже аналізувалось раніше, при створенні додатку завжди необхідно продумувати сценарії роботи з ним для людей з вадами зору. Але дещо інша ситуація виникає тоді, коли застосунок розробляється конкретно для людей з ними. Необхідно відкинути стандартний процес створення дизайну з нахилом до

візуальної складової, сучасних кольорів та форм, і проаналізувати як сприймають застосунок люди з проблемами зору. Для них не є важливим чи material design було використано, чи flat design. Вони не піклуються про те, які саме анімації було реалізовано в застосунку. Для цих людей важливо, щоб було просто отримати необхідну інформацію, щоб навігація програмою була інтуїтивною та зрозумілою і не вимагала багато часу на звикання до неї. Людям з кольоровою сліпотою також важливо, щоб текст був чітко читабельним і не зливався по кольорам з фоновими малюнками.

З думкою про це і було створено дизайн та продумані сценарії взаємодії застосунку SeeForMe. Головний екран містить інформацію та можливість переходу лише на одну з функцій. (див. рис. 3.5)

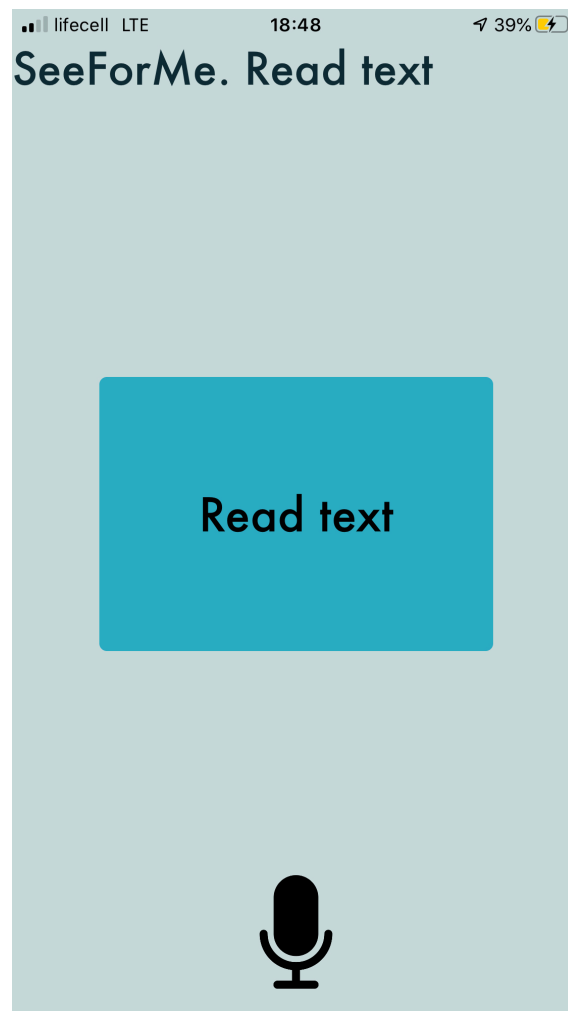


Рисунок 3.5 Головний екран

При включенні – це функція розпізнавання тексту. VoiceOver захоплює та автоматично озвучує перший текстовий елемент на екрані, а саме тому вгорі кожного вікна присутній текстовий елемент з інформацією про те, де саме зараз

перебуває користувач. Посередині екрану розташована одна велика кнопка з назвою команди. Розміри кнопки обумовлені тим, щоб її легше було знайти на екрані. Також у меню вибору знизу присутня кнопка активації голосової команди. Для переходу до наступного пункту меню користувачеві необхідно просто зробити свайп вгору і з'явиться наступна опція в меню.

Рішення про одну кнопку на екран було прийнято для того, щоб не перенавантажувати меню і дати змогу користувачеві отримувати можливості дозовано та вручну переходити до наступної, в разі невідповідності поточної його запиту.

У випадку переходу на сторінку розпізнавання тексту або ж кольору, зверху все рівно присутнє текстове поле з назвою екрану, а от весь інший простір вже займає відображення камери. (див. рис. 3.6)

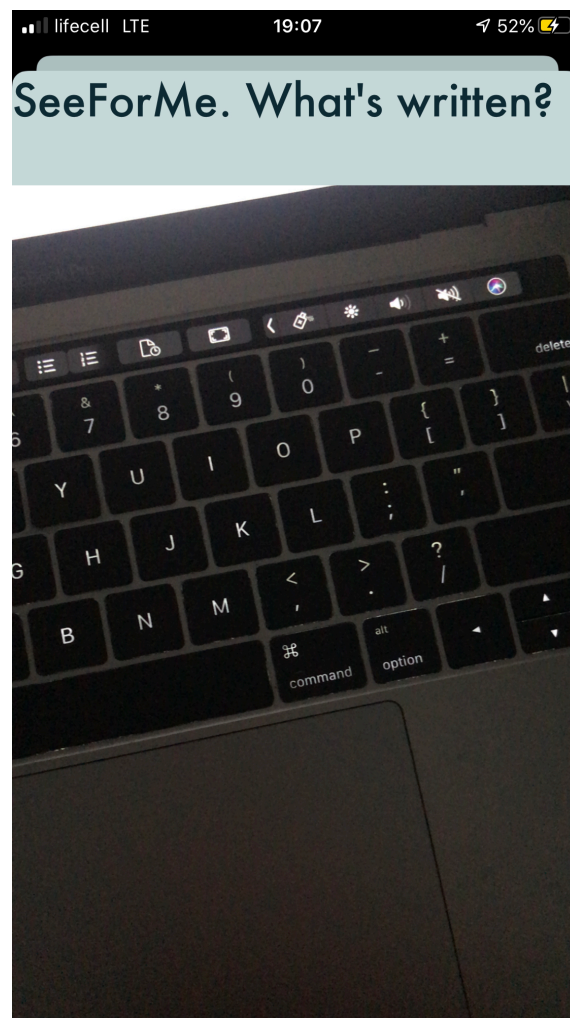


Рисунок 3.6 Екран розпізнавання тексту

Для управління застосунком також була розроблена зручна система жестів. Для початку розпізнавання тексту необхідно потримати палець на

зображенні, після чого воно почнеться. По завершенню застосунк в голосовому форматі дасть підказки щодо можливої подальшої дії, а саме: два натиснення на екран збережуть поточний запис чи колір до бази даних. Повернення назад з будь-якого екрану реалізовано через горизонтальний свайп від краю екрану або ж вертикальний згори. Також будь-які помилки в ході користування програмою озвучуються користувачеві, а при першому запускові програми він отримує інструкцію щодо управління застосунком.

Всі ці особливості користувацького інтерфейсу та взаємодії з користувачем в об'єднанні з VoiceOver створюють по-справжньому зручний процес користування.

3.6 Опис розробки застосунку

Робота над застосунком почалася з створення назви, розробки лого, та вибору кольорової гами. Було вирішено створити таку назву, в ідею якої вклався б основний зміст та ціль застосунку. Після аналізу предметної області було створено поточну назву проєкту – SeeForMe, що з англійської перекладається як «Дивись для мене». Фінальна назва містить в собі основний запит користувача застосунку – запит на надання певної візуальної підтримки.

Після створення назви, був проведений аналіз можливого логотипа. Він також мав би мати певний зв'язок з основною ідеєю застосунку та бути його символом. Додаток в певному сенсі надає зір користувачеві, а око є органом, що відповідає за бачення. Тому було прийнято рішення побудувати концепцію логотипа застосунку навколо символу ока. Серед безлічі створених прототипів, було обрано три найімовірніших (див. рис. 3.7.) та серед них використано один фінальний (див. рис. 3.8.)



Рисунок 3.7 Три фінальні варіанти логотипу



Рисунок 3.8 Фінальний логотип SeeForMe

Після визначення з логотипом та назвою було вирішено витримати застосунок в кольорах, що з'являються на лого, а саме #5FC2D4 та #032E35.

Визначившись з візуальною складовою застосунку, було почато планування архітектури та основної структури програми. Як вже було описано в попередніх розділах, загалом застосунок розроблявся опираючись на патерн MVC.

Головний екран та можливість свайпу на наступні сторінки було реалізовано через PageViewController. ManagePageViewController, що наслідується від UIPageViewController, контролює поточний стан сторінок та перехід на сторінку в разі свайпу (див. лістинг 3.1)

```
extension ManagePageViewController: UIPageViewControllerDataSource {
    func pageViewController(_ pageViewController: UIPageViewController, viewControllerBefore viewController: UIViewController) → UIViewController? {
        if let viewController = viewController as? MainPageViewController, let index = viewController.buttonIndex,
            index > 0 {
            return viewMainPageController(index - 1)
        }
        return nil
    }

    func pageViewController(_ pageViewController: UIPageViewController, viewControllerAfter viewController: UIViewController) → UIViewController? {
        if let viewController = viewController as? MainPageViewController, let index = viewController.buttonIndex,
            (index + 1) < viewController.buttonNames.count {
            return viewMainPageController(index + 1)
        }
        return nil
    }
}
```

Лістинг 3.1 Додатки ManagePageViewController

В ході тестування була знайдена проблема, за якої у користувача при ввімкненому VoiceOver не працювали свайпи. Це виникало через специфіку роботи VoiceOver, який спочатку перехоплює жести і вже потім віддає їх застосунку. Так трапилось і в цьому випадку. VoiceOver просто перехоплював

свайп і код застосунку ніколи не отримував сигналу перейти на іншу сторінку. Дана проблема була вирішена шляхом переписування системної функції, що перехоплює жести, та ручного вказання інструкцій щодо подальшої роботи. (див. лістинг 3.2)

```
//code to support scrolling with accessibility and enable three finger swipe
override func accessibilityScroll(_ direction: UIAccessibilityScrollDirection) -> Bool {

    //down means literally down
    if direction == .down {
        guard let currentController = viewMainPageController(currentIndex ?? 0), let pageViewController = pageViewController(self, viewControllerAfter: currentController) else {
            return false
        }

        //manually incrementing current index
        self.currentIndex = self.currentIndex == nil ? 0 : self.currentIndex! + 1
        //setting vcs manually, too
        self.setViewControllers([pageViewController], direction: .forward, animated: true, completion: nil)
    } else if direction == .up {
        guard let currentController = viewMainPageController(currentIndex ?? 0), let pageViewController = pageViewController(self, viewControllerBefore: currentController) else {
            return false
        }
        self.currentIndex = self.currentIndex == nil ? 0 : self.currentIndex! - 1
        self.setViewControllers([pageViewController], direction: .reverse, animated: true, completion: nil)
    }

    UIAccessibility.post(notification: .pageScrolled,
        argument: nil)

    return true
}
```

Лістинг 3.2 Вирішення проблеми зі свайпами

ManagePageViewController переключає інстанси класу MainPageViewController та налаштовує їх відповідно до того, яка сторінка зараз має відображатись.

Контролери, що відповідають за зв'язок між представленням сторінок з розпізнаванням тексту та кольору містять багато ідентичного коду. Задля уникнення непотрібних дублікатів був створений клас ActionController, що містить спільні для обох класів властивості та методи, як от обробка довгого натискання на екран, або ж двох швидких. Відповідно до такої архітектури, VisionViewController та ColorViewController наслідуються не напряду від UIViewController, а від ActionController, який вже сам є підкласом UIViewController. (див. рис. 3.9)

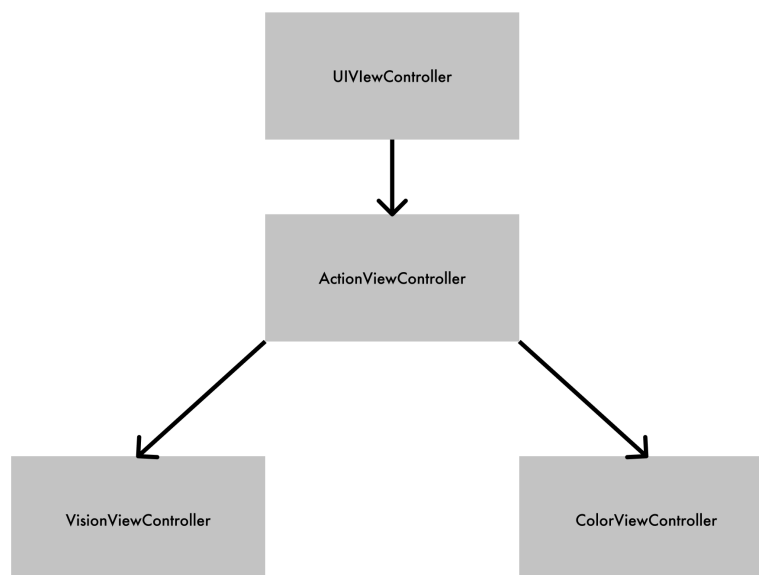


Рисунок 3.9 Демонстрація наслідування задля уникнення дублікатів коду

MainPageViewController також містить активацію голосової команди для можливості швидкого голосового переходу на бажану сторінку. (див. лістинг 3.3) При натисненні на іконку мікрофона запускається recordAndRecognizeSpeech метод з моделі Voice. Також у метод передається completion. Це функція, що буде викликана по завершенню ідентифікації вимовленого. В ній результат розпізнавання буде порівняно з існуючими шаблонами і в разі не знаходження збігу – застосунок виконує функцію read з вхідним параметром – текстом, який необхідно вимовити. Також використовується змінна awaitingForResult, яка визначає, чи чекає функція на нові результати.

```

@IBAction func voiceCommandButtonPressed(_ sender: UIButton) {
    awaitingForResult = true
    voice.recordAndRecognizeSpeech(completion: { command in
        if self.awaitingForResult {
            self.awaitingForResult = false
            if command.lowercased() == "read" {
                self.presentViewController(with: "Read text")
            } else if command.lowercased() == "identify" {
                self.presentViewController(with: "Identify color")
            } else if command.lowercased() == "text" {
                self.presentViewController(with: "Saved texts")
            } else if command.lowercased() == "color" {
                self.presentViewController(with: "Saved colors")
            } else {
                self.voice.read(text: "Command is not recognized. Please, retry your voice command")
            }
        }
    })
}
}

```

Лістинг 3.3 Запуск розпізнавання голосу

`recordAndRecognizeSpeech` спочатку зупиняє всі попередньо запущені процеси (якщо такі є), потім робить налаштування мікрофону, і створює новий `recognitionRequest`. `RecognitionTask` вказує інструкції по завершенню розпізнавання та виконує `completion`, який був переданий з `MainPageViewController`. Після повернення результату всі запити окремо закриваються і припиняється робота мікрофона. (див. лістинг 3.4)

```
func recordAndRecognizeSpeech(completion: @escaping (String) → Void) {
    let audioEngine = AVAudioEngine()

    guard let speechRecognizer = speechRecognizer else {
        read(text: "Couldn't create speech recognizer. Please restart your app and try again")
        return
    }

    recognitionTask?.cancel()
    self.recognitionTask = nil
    do {
        try audioSession.setCategory(.playAndRecord, options: .defaultToSpeaker)
        try audioSession.setActive(true, options: .notifyOthersOnDeactivation)
    } catch {
        read(text: "Unable to process your request")
        return
    }

    let inputNode = audioEngine.inputNode

    let recordingFormat = inputNode.outputFormat(forBus: 0)
    inputNode.installTap(onBus: 0, bufferSize: 1024, format: recordingFormat) {
        (buffer: AVAudioPCMBuffer, when: AVAudioTime) in
        self.recognitionRequest?.append(buffer)
    }

    audioEngine.prepare()
    do {
        try audioEngine.start()
    } catch {
        read(text: "Couldn't start audio engine. Please restart your app and try again")
        return
    }

    recognitionRequest = SFSpeechAudioBufferRecognitionRequest()
    guard let recognitionRequest = recognitionRequest else {
        read(text: "Couldn't create recognition request. Please restart your app and try again")
        return
    }

    if #available(iOS 13, *) {
        if speechRecognizer.supportsOnDeviceRecognition {
            recognitionRequest.requiresOnDeviceRecognition = true
        }
    }

    recognitionTask = speechRecognizer.recognitionTask(with: recognitionRequest) { result, error in
        if let result = result {
            DispatchQueue.main.async {
                completion(result.bestTranscription.formattedString)
            }
        }

        self.recognitionTask?.cancel()
        self.recognitionTask?.finish()
        audioEngine.stop()
        self.recognitionRequest?.endAudio()
        inputNode.removeTap(onBus: 0)
        if error != nil {
            self.recognitionRequest = nil
            self.recognitionTask = nil
        }
    }
}
```

Лістинг 3.4 Реалізація функції розпізнавання команд

За голосове озвучування заданого тексту у всій програмі відповідає функція `read` з моделі `Voice`. Запустившись, вона зупиняє поточний процес

відтворення, якщо такий є, потім налаштовує вимову та виконує метод `speak` з бібліотеки `AVFoundation` (див. лістинг 3.5)

```
func read(text input: String) {
    synthesizer.stopSpeaking(at: AVSpeechBoundary.immediate)
    let utterance = AVSpeechUtterance(string: input)
    utterance.voice = AVSpeechSynthesisVoice(language: "en-US")
    utterance.rate = utteranceRate
    synthesizer.speak(utterance)
}
```

Лістинг 3.5 Реалізація функції вимови тексту

Як вже було сказано в попередніх розділах, для розпізнавання тексту було використано фреймворк `Vision`, всі операції з ним відбуваються в окремому виділеному класі `Vision`.

В ініціалізації класу ми викликаємо функцію, що налаштовує `VNRecognizeTextRequest` та вказуємо необхідні параметри, як от точність розпізнавання, що у випадку `SeeForMe` стоїть точною, та використання корекції мови, що також використовується в додатку. (див. лістинг 3.6)

```
init() {
    //set up our text recognition request
    setupVisionRequest()
}

// Setup Vision request in order to reuse it in the future
private func setupVisionRequest() {
    request = VNRecognizeTextRequest { (request, error) in
        guard let observations = request.results as? [VNRecognizedTextObservation] else {
            print("Unexpected type of observations received")
            return
        }
        let maximumCandidates = 1
        for observation in observations {
            guard let candidate = observation.topCandidates(maximumCandidates).first else { continue }
            self.currentTextResult += candidate.string + "\n"
        }
    }

    // implicitly unwrapping request because we just initialized it above
    request!.recognitionLevel = .accurate
    request!.usesLanguageCorrection = true
}
```

Лістинг 3.6 Ініціалізація моделі Vision

За саме розпізнавання відповідає функція `processReceived`, що приймає на вхід зображення та функцію, яку необхідно викликати по завершенню. Сам запит виконується асинхронно, а `completion` функція викликається по отриманню результатів в головному потоці. (див. лістинг 3.7)


```

func processReceived(cgImage: CGImage, completion: @escaping (String, Error?) → Void) {
    guard let request = request else {
        completion("", VisionError.requestInvalid)
        return
    }

    //perform an async call for text recognition
    self.textRecognitionQueue.async {
        self.currentTextResult = ""
        let requestHandler = VNImageRequestHandler(cgImage: cgImage, orientation: .right, options: [:])
        do {
            try requestHandler.perform([request])
        } catch {
            print(error)
        }
        DispatchQueue.main.async(execute: {
            completion(self.currentTextResult, nil)
        })
    }
}
}
}

```

Лістинг 3.7 Виконання розпізнавання зображення

На сьогоднішній день не існує простого фреймворка для iOS, що транслював би колір з цифрового значення у назву, яка була б зрозуміла людині. А тому постала задача розробити алгоритм, що виконував би таке перетворення.

Загальна схема алгоритму зображена на рисунку 3.10:

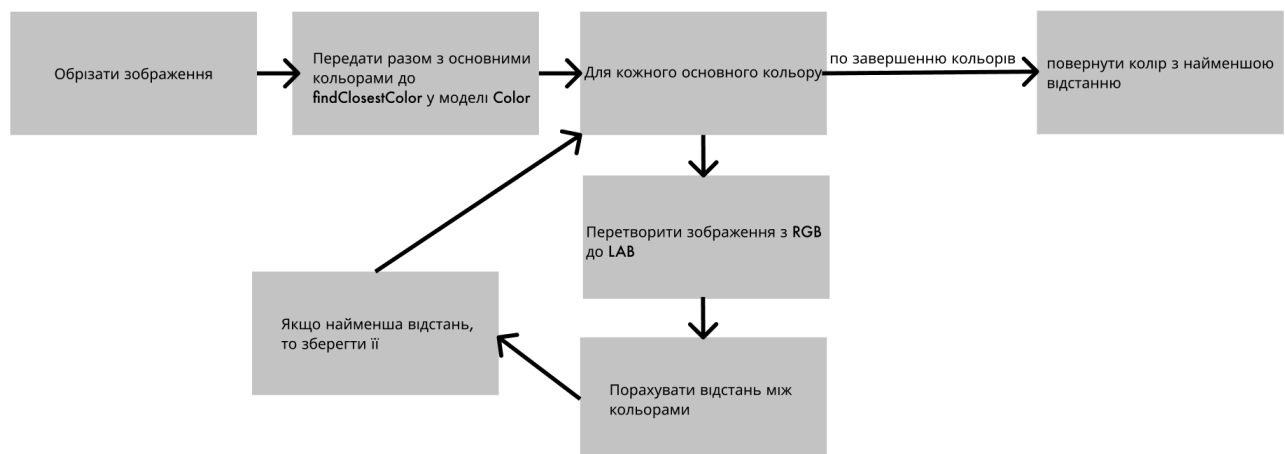


Рисунок 3.10 Алгоритм пошуку найближчого кольору

Алгоритм засновується на порівнянні відстані між значенням вхідного кольору та основними кольорами і відповідь знаходиться базуючись на найменшій відстані. Є 20 основних кольорів, назви яких і використовуються у результаті. Сенсу в більшій кількості немає, адже необхідно забезпечити розуміння користувачем отриманого результату.

Для початку необхідно отримати той набір пікселів, який відповідав би центру зображення, адже ми рахуємо основний колір з об'єкта, що знаходиться в центрі об'єктива. Для цього була створена допоміжна функція `crop()`, що обрізає зображення до його центрального регіону. Після цього цей регіон разом

з колекцією основних зображень передається в функцію `findClosestColor`. Всередині функції вхідне зображення порівнюється з усіма базовими кольорами та повертає як результат назву кольору, з яким різниця є мінімальною. (див. лістинг 3.8)

```
func findClosestColor(for color: UIColor, among colorsCollection: [ColorEntity]) → String {
    var minDistance = Double.infinity
    var closestColor = ""

    colorsCollection.forEach({
        currentColor in
        let distance = distanceBetween(colorOne: color, and: currentColor.correspondingColor)
        if minDistance > distance {
            minDistance = distance
            closestColor = currentColor.name
        }
    })

    return closestColor
}
```

Лістинг 3.8 Функція знаходження назви найближчого кольору

Дистанція між кольорами означає їх візуальну різницю. Чим більша відстань між ними, тим менш схожими є кольори. Наприклад, відстань у чорного і білого буде більшою, ніж у чорного та темно-коричневого. Задача `distanceBetween` знаходити значення дистанції між двома кольорами. (див. лістинг 3.9) Отримана дистанція між двома RGB кольорами буде неправильною, а тому необхідно провести конвертацію з RGB до CIELAB. Для цього за формулами відбувається конвертація з RGB у формат XYZ, а потім з XYZ в CIELAB.

```
func distanceBetween(colorOne: UIColor, and colorTwo: UIColor) → Double {
    //converting both colors to LAB format
    let colorOneLAB = colorOne.lab
    let colorTwoLAB = colorTwo.lab

    //Delta E 76 color difference formula used
    let result = (pow((colorTwoLAB.CIEL - colorOneLAB.CIEL), 2) + pow((colorTwoLAB.CIEa - colorOneLAB.CIEa), 2) + pow((colorTwoLAB.CIEb - colorOneLAB.CIEb), 2)).squareRoot()
    return Double(result)
}
```

Лістинг 3.9 Функція обрахування дистанції між кольорами

Неможливо точно визначити, чи схожі між собою два кольори в представленні RGB, так як різниця в просторі RGB не відповідає тому як її бачить людське око. CIELAB, в свою чергу, була створена, для представлення кольорів в той самий спосіб, як їх розрізняють люди. Тому дистанція і рахується саме між кольорами в цьому просторі.

Для обрахування дистанції між кольорами використовується формула Delta E76. (див. рис. 3.11)

$$\Delta E_{ab}^* = \sqrt{(L_2^* - L_1^*)^2 + (a_2^* - a_1^*)^2 + (b_2^* - b_1^*)^2}.$$

Рисунок 3.11 Формула Delta E76 для обрахування відстані між кольорами[25]

В даній формулі результат – це дистанція між кольорами, L^* - світлість кольору, a^* – позиція кольору в діапазоні від зеленого до червоного, а b^* - від синього до жовтого.

Знайдений за таким алгоритмом колір і озвучується як результат користувачеві.

Знайдений текст та колір користувач може зберегти до бази даних, а тому необхідно було створити базу даних та реалізувати кодову частину взаємодії з нею.

Загалом, для виконання даного завдання не потрібно робити складних маніпуляцій з базами даних, а тому було створено дві таблиці: Data та Colors, і декілька запитів: отримати всі базові кольори, отримати всі збережені кольори, отримати весь збережений текст, зберегти новий текст та колір та видалити бажані елементи. Data містить унікальний ідентифікатор, атрибут content, що зберігає в залежності від задачі назву кольору або текст, атрибут is_Text який визначає чи це є збереженим текстом чи кольором, а також атрибут Date, який визначає дату збереження елемента. В таблиці Colors є атрибут id, який є зовнішнім ключем атрибута id з таблиці Data, значення кольорів red, green, blue, та атрибут is_Base, який позначає чи це один з основних кольорів, або ж збережений користувачем. (див. рис. 3.12).

▼ Colors	
red	INTEGER
green	INTEGER
blue	INTEGER
id	INTEGER
is_Base	INTEGER
▼ Data	
id	INTEGER
content	TEXT
is_Text	INTEGER
Date	TEXT

Рисунок 3.12 Схема бази даних

За замовчуванням SQLite не має типів Boolean та Date, а тому всі логічні значення представлені типом INTEGER та двома варіантами 0 або 1. За представлення дати відповідає програмна частина, що кодує її в стрічку та зберігає в базу даних типом TEXT.

3.7 Принципи роботи готового застосунку

Вперше заходячи в додаток користувач має прослухати невелику інструкцію щодо керування застосунком, а потім програма автоматично відкриє головне вікно з розділом розпізнавання тексту. Обравши дану категорію або ж категорію розпізнавання кольорів (візуальна відмінність лише у заголовках), користувач отримає вікно з попереднім переглядом камери. (див. рис. 3.13)

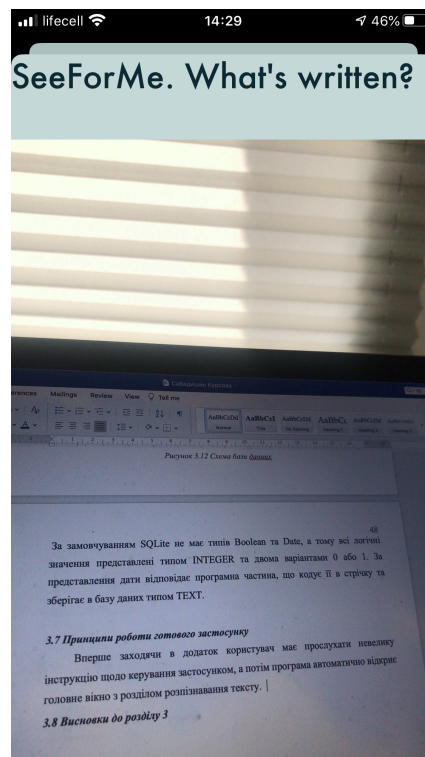


Рисунок 3.13 Дизайн вікна розпізнавання тексту

Після сканування програма нагадує про можливість збереження та надає результати.

У разі переходу в меню збережених текстів або ж кольорів, користувач побачить список збереженої інформації, при натисненні на яку йому буде озвучено всі деталі щодо елементу. (див. рис. 3.14)

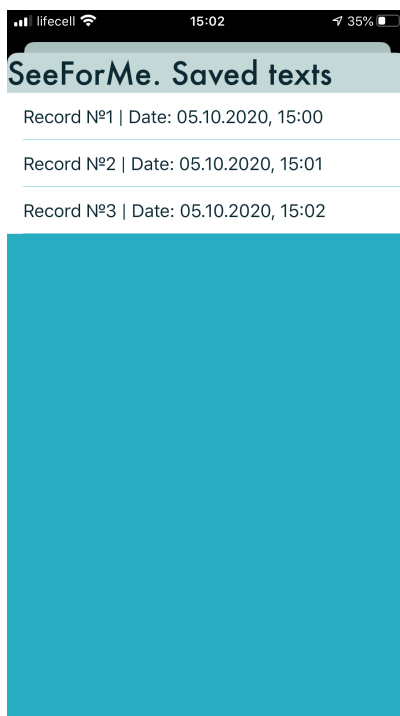


Рисунок 3.14 Інтерфейс збережених текстів

Задля виконання команди голосом необхідно просто натиснути на іконку мікрофона та промовити одну з команд: “Text”, “Color”, “Identify”, або ж “Recognize”. Після цього користувача буде переведено на сторінку з бажаною функціональністю або ж повідомлено, що вказана команда є невірною.

3.8 Висновки до розділу 3

В розділі 3 було описане технічне завдання, обґрунтовано причини вибору технологій при розробці застосунку, а також описано середовище розробки та його переваги.

Так як розробка застосунку для людей з вадами зору вимагає окремого підходу для створення дизайну, в розділі було описано та пояснено причини створення саме такого користувацького інтерфейсу та методів взаємодії користувача і застосунку.

В кінці розділу було описано процес створення застосунку, а також наведено скріншоти з роботою готового продукту.

Висновки по роботі та аналіз можливостей для подальшого розвитку застосунку

Отже, по закінченню роботи було створено застосунок SeeForMe для допомоги людям з вадами зору, в розробці було використано технології комп'ютерного бачення на базі фреймворку Vision. Поставлена задача була виконана повністю та продукт готовий для використання. До початкової постановки завдання були додані додаткові функціональні особливості, а саме: можливість збереження отриманих результатів в базу даних, а також функція голосового переходу в бажане меню. Крім того, був розроблений особливий дизайн та способи взаємодії, зручні для користування з урахуванням цільової аудиторії, що не сприйматиме візуальні елементи.

Під час розробки застосунку виникали певні труднощі, які були успішно подолані. До прикладу, відсутність фреймворку для Swift, що конвертував би колір з цифрового значення в зрозумілу для людей назву, була вирішена розробкою власного алгоритму для такого перетворення. Або ж перехоплення жестів користувача VoiceOver, а тому унеможливлення використовувати свайпи в застосунку, які є ключовим жестом в переході на наступне вікно. Це питання було вирішено перезаписом системної функції отримання жестів та ручним прописуванням необхідного сценарію роботи застосунку.

Даний проєкт має багато можливостей для подальшого розвитку і може стати центральним елементом хабу застосунків для людей з різними вадами здоров'я. На поточному етапі розглядається декілька можливостей подальшого покращення застосунку, а саме додавання функції «Listen for me», що допомагала б людям з вадами слуху, трансліюючи мовлення в текст на екрані та «Speak for me», що допомагала б людям з порушеннями слуху озвучувати те, що вони бажають сказати співрозмовникові без розуміння мови жестів по мережі або ж в реальному світі. Також варіантом розвитку є

функціонал розпізнавання об'єктів, але на сьогоднішній день проблематика складається в створенні такого універсального розпізнавача.

Список джерел

1. SUNspot – Use of Wireless Devices by People with Disabilities [Електронний ресурс]
http://www.wirelessrerc.org/sites/default/files/publications/sunspot_2013-01_wireless_devices_and_people_with_disabilities_final.pdf
2. Blind and Deaf Consumer Preferences for Android and iOS Smartphones by J. Morris and J. Mueller [Електронний ресурс]
http://www.wirelessrerc.gatech.edu/sites/default/files/publications/morris_et_al_-_blind_and_deaf_consumer_preferences_for_android_and_ios_smartphones.pdf
3. Marrakesh Treaty Ratification and Implementation Campaign [Електронний ресурс]
<http://www.worldblindunion.org/English/our-work/our-priorities/Pages/right-2-read-campaign.aspx>
4. Global Data on Visual Impairments 2010, ст. 5 [Електронний ресурс]
<https://www.who.int/blindness/GLOBALDATAFINALforweb.pdf>
5. Google Maps [Електронний ресурс]
<https://apps.apple.com/us/app/google-maps-transit-food/id585027354>
6. Be My Eyes [Електронний ресурс]
<https://apps.apple.com/us/app/be-my-eyes/id905177575>
7. R. Hartley, A. Zisserman – Multiple View Geometry in Computer Vision – Cambridge University Press, ст. 11
8. Computer Vision in Artificial Intelligence [Електронний ресурс]
<https://blogs.oracle.com/datascience/computer-vision-in-artificial-intelligence>
9. A Gentle Introduction to Computer Vision [Електронний ресурс]
<https://machinelearningmastery.com/what-is-computer-vision/>
10. Apple Introducing VoiceOver [Електронний ресурс]
https://www.apple.com/voiceover/info/guide/_1121.html
11. What is Google TalkBack? [Електронний ресурс]
<https://www.androidcentral.com/what-google-talk-back>

12. AppleTV [Электронный ресурс]
<https://apps.apple.com/us/app/apple-tv/id1174078549>
13. App Store [Электронный ресурс]
<https://developer.apple.com/support/app-store/>
14. Mobile and Tablet Android Version Market Share Worldwide [Электронный ресурс]
<https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide>
15. Model-View-Controller [Электронный ресурс]
<https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>
16. Understanding Delegate Pattern in Swift [Электронный ресурс]
<https://en.proft.me/2018/08/16/understanding-delegate-pattern-swift/>
17. Objective-C to Swift Code XCode Plugin [Электронный ресурс]
<https://maniacdev.com/2016/04/objc2swift-xcswiftr-objective-c-to-swift-code-converter-tool-and-xcode-plugin>
18. Vision Framework [Электронный ресурс]
<https://developer.apple.com/documentation/vision#overview>
19. Speech Framework [Электронный ресурс]
<https://developer.apple.com/documentation/speech>
20. Siri [Электронный ресурс]
<https://www.apple.com/siri/>
21. Dictation [Электронный ресурс]
<https://support.apple.com/en-us/HT208343>
22. How SQLite Is Tested [Электронный ресурс]
<https://www.sqlite.org/testing.html>
23. Swift Package Manager [Электронный ресурс]
<https://swift.org/package-manager/>
24. XCode Overview [Электронный ресурс]
https://developer.apple.com/library/archive/documentation/ToolsLanguages/Conceptual/Xcode_Overview/index.html
25. Delta E101 [Электронный ресурс]
<http://zschuessler.github.io/DeltaE/learn/>

Додаток А. Функції перетворення зображень

```
extension UIColor {

    var ciColor: CIColor {
        return CIColor(color: self)
    }

    var rgb: (red: CGFloat, green: CGFloat, blue: CGFloat) {
        let ciColor = self.ciColor
        return (ciColor.red, ciColor.green, ciColor.blue)
    }

    //returns image in LAB format in order to calculate distance between images on another level that's common to how
    //humans compare them
    //we will first convert RGB to XYZ and then XYZ to LAB
    var lab: (CIEL: CGFloat, CIEa: CGFloat, CIEb: CGFloat) {
        let rgbValues = self.rgb

        //RGB → XYZ
        var rgbRed = rgbValues.red
        var rgbGreen = rgbValues.green
        var rgbBlue = rgbValues.blue

        if rgbRed > 0.04045 {
            rgbRed = pow(((rgbRed + 0.055) / 1.055), 2.4)
        }
        else {
            rgbRed /= 12.92
        }

        if rgbGreen > 0.04045 {
            rgbGreen = pow(((rgbGreen + 0.055) / 1.055), 2.4)
        }
        else {
            rgbGreen /= 12.92
        }

        if rgbBlue > 0.04045 {
            rgbBlue = pow(((rgbBlue + 0.055) / 1.055), 2.4)
        }
        else {
            rgbBlue /= 12.92
        }

        rgbRed *= 100
        rgbGreen *= 100
        rgbBlue *= 100

        let X = rgbRed * 0.4124 + rgbGreen * 0.3576 + rgbBlue * 0.1805
        let Y = rgbRed * 0.2126 + rgbGreen * 0.7152 + rgbBlue * 0.0722
        let Z = rgbRed * 0.0193 + rgbGreen * 0.1192 + rgbBlue * 0.9505

        //XYZ → LAB
        var labX = X / 95.047
        var labY = Y / 100.0
        var labZ = Z / 108.883

        if labX > 0.008856 {
            labX = pow(labX, 1/3)
        }
        else {
            labX = (7.787 * labX) + (16 / 116)
        }

        if labY > 0.008856 {
            labY = pow(labY, 1/3)
        }
        else {
            labY = (7.787 * labY) + (16 / 116)
        }

        if labZ > 0.008856 {
            labZ = pow(labZ, 1/3)
        }
        else {
            labZ = (7.787 * labZ) + (16 / 116)
        }

        let CIEL = (116 * labY) - 16
        let CIEa = 500 * (labX - labY)
        let CIEb = 200 * (labY - labZ)

        return (CIEL, CIEa, CIEb)
    }
}
```