

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗПІЗНАВАННЯ
УПАКОВОК МЕДИКАМЕНТІВ НАЖИВО З ВИКОРИСТАННЯМ

ТЕХНОЛОГІЙ МАШИННОГО НАВЧАННЯ

RESEARCH AND SOFTWARE IMPLEMENTATION OF MEDICATION
PACKING RECOGNITION IN LIVE MODE USING MACHINE LEARNING
TECHNOLOGY

Виконав: студент(ка) 2 курсу, групи КНТ-110м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Харебашвілі І. Г.

(прізвище та ініціали)

Керівник Степаненко О. О.

(прізвище та ініціали)

Рецензент Казурова А. Є.

(прізвище та ініціали)

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
_____ **С. О. Субботін**
“ ____ ” _____ 20 ____ року

(прізвище, ім'я, по батькові)

керівник проєкту (роботи) Степаненко О.О. к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити 1. Аналіз предметної області 2. Характеристики об'єкта проектування

3. Вибір і обґрунтування структури та методів розпізнавання проектованої системи

4. Керівництво програміста 5 Керівництво користувача

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділи	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Степаненко О.О. к.т.н., доцент		
Нормоконтроль	Дейнега Л. Ю., ст. викладач		

7. Дата видачі завдання “ 02 ” вересня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-4 тижні	Розділ 1
3	Вибір засобів розробки програмного продукту	5-6 тижні	Розділ 2
	Розробка архітектури програми	7-10 тижні	Розділ 3
4	Розробка програмного продукту	11-13 тижні	Розділ 4-5
5	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	14-15 тижні	Додатки
6	Захист роботи	16 тиждень	

Студент(ка)

(підпис)

Керівник проєкту (роботи)

(підпис)

Харєбашвілі І. Г.

(прізвище та ініціали)

Степаненко О. О.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: - 163 с., 17 табл., 41 рис., 2 дод., 32 джерел.

МАШИННЕ НАВЧАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ДОПОВНЕНА РЕАЛЬНІСТЬ, МЕДИЦИНА, ФАРМАЦЕВТ, МОБІЛЬНИЙ ЗАСТОСУНОК.

Об'єктом дослідження є розробка системи розпізнавання упаковок медикаментів наживо з використанням технологій машинного навчання.

Предмет дослідження – мобільний застосунок, з допомогою якого можна буде сканувати упаковки з медичними препаратами та дивитись інформацію про них.

Задачі – розробка програмного забезпечення, яке допоможе простим користувачам та фармацевтам оперативно шукати інформацію про той чи інший препарат.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мова програмування Swift, середовище розробки Xcode, менеджер залежностей Cocoapods, комп'ютер з шестидерним процесором Intel Core i7, оперативною пам'яттю на 16 Гб, монітором, клавіатурою та маніпулятором, який керує курсором.

Виконано аналіз предметної області, розглянуто аналоги на ринку, що зараз існують, на основі цього були виявлені переваги та недоліки кожного рішення та було розроблено модель, яка розв'язує найголовніші проблеми аналогів, що було розглянуто. Після чого за допомогою розробленої моделі було реалізовано програмне рішення для популярної мобільної платформи.

Також було розглянуто особливості обраної мобільної платформи для розробки, мови програмування, середовища розробки.

Основною платформою для розробки була обрана операційна система iOS з підтримкою IOS 14.0 і вище.

ABSTRACT

Explanatory note to the master's thesis: - 163 pages, 17 tables, 41 figures, 2 appendixes, 32 sources.

MACHINE LEARNING, SOFTWARE, AUXILIARY REALITY, MEDICINE, PHARMACIST, MOBILE APPLICATION.

The object of the research is the development of a system for recognizing live packaging of medicines using machine learning technologies.

The subject of the research is a mobile application, with the help of which it will be possible to scan packages with medical drugs and view information about them.

Tasks - the development of software that will help ordinary users and pharmacists to quickly search for information about a drug.

Materials, methods and technical means: structural and object-oriented programming, Swift programming language, Xcode development environment, Foundation library, UIKit library, Cocoapods dependency manager, computer with six-core Intel Core i7 processor, 16GB RAM , monitor, keyboard, and cursor control. The computer described is running the macOS Big Sur operating system.

The analysis of the subject area is performed, the analogues on the market that currently exist are considered, based on this the advantages and disadvantages of each solution are identified and a model is developed that solves the main problems of analogues that were considered. Then, with the help of the developed model, a software solution for a popular mobile platform was implemented.

The features of the selected mobile platform for development, programming language, development environment were also considered.

The main platform for development was the iOS operating system with support for IOS 14.0 and higher.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
.....	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Особливості предметної області	10
1.2 Сучасний метод контролю неефективних препаратів	12
1.3 Висновки по розділу 1	12
2 ХАРАКТЕРИСТИКИ ОБ'ЄКТА ПРОЕКТУВАННЯ.....	13
2.1 Існуючі аналоги системи.....	13
2.1.1 Ліки контроль	13
2.1.2 Чесний знак	20
2.2 Порівняння програмних засобів, що розглядались	22
2.3 Висновки по розділу 2.....	23
3 ВИБІР І ОБГРУНТУВАННЯ СТРУКТУРИ ПРОЕКТОВАНОЇ СИСТЕМИ.....	25
3.1 Вимоги до програмного забезпечення.....	25
3.2 Огляд особливостей мови програмування	25
3.3 Огляд особливостей обраної середовища розробки	29
3.4 Опис особливостей програмної реалізації	30
3.4.1 Архітектура програмного продукту.....	30
3.4.2 Патерни, що використовуються	33
3.4.3 База даних	34
3.4.4 Опис структури бази даних	35
3.5 Способи реалізації розпізнавання упаковок препаратів	36
3.6 Порівняння та вибір алгоритмів машинного навчання.....	39
3.6.1 Обмежена машина Больцмана.....	39
3.6.2 Автокодувальник	40
3.6.3 Згортова нейронна мережа.....	41
3.6.4 Fully-connected neural networks.....	45
3.7 Огляд програмних засобів глибокого навчання	46

	7
3.7.1 Бібліотека Torch	48
3.7.2 CoreML.....	49
3.8 Порівняння швидкості розпізнавання бібліотек Torch та CoreML	51
3.9 Модернізація алгоритму розпізнавання	55
3.10 Висновки по розділу 3	59
4 КЕРІВНИЦТВО ПРОГРАМІСТА	61
4.1 Інформація про призначення та функціонал програми	61
4.2 Характеристики програми	61
4.3 Середовище розробки та тестування	61
4.4 Функції програми.....	62
4.5 Менеджери залежностей	63
4.5.1 Cocoapods.....	63
4.5.2 Carthage та порівняння з Cocoapods.....	64
4.5.3 Swift Package Manager	65
4.6 Установка та запуск робочого місця.....	66
4.7 Особливості побудови дизайну	67
4.8 Блок схеми користуванням роботи додатку	69
4.8.1 Процес сканування	73
4.9 Діаграми класів	74
5 КЕРІВНИЦТВО КОРИСТУВАЧА.....	79
5.1 Головний екран	79
5.2 Режим фотографії	81
5.3 Режим сканування у реальному часі.....	83
5.4 Результат сканування	85
ВИСНОВКИ	86
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	88
Додаток А Текст програми	91
Додаток Б Слайди презентації.....	149

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API - (Application Programming Interface) набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення;

AR – (Augmented reality) доповнена реальність [29];

IOS – (iPhone operation system) мобільна операційна система компанії Apple [25];

HTTP - Hyper Text Transfer Protocol;

IDE – (Integrated Drive Electronics) інтегроване середовище розробки;

IP - Internet Protocol;

JSON – (JavaScript Object Notation) текстовий формат обміну даними;

MVC - Model-View-Controller;

MVP – Model-View-Presenter [32];

XCODE – середовище розробки компанії Apple;

ОС – операційна система.

ВСТУП

На сьогоднішній день фармацевтичне виробництво не стоїть на місці і щодня з'являються нові і нові препарати для лікування як звичайної застуди, які продаються без рецепта, так і складних захворювань.

У зв'язку з цим на ринку з'являються не дієві. Мета виробництва даних препаратів - не прибрати симптоми захворювання та вилікувати пацієнта, а заробити на пацієнтах, і деякі малопопулярні аптеки чи інтернет-магазини займаються продажем даних «фуфломіцинів» [1].

За статистикою, серед найпопулярніших в Україні препаратів 40% не мають наукового обґрунтування, а за 5 років українці витратили на не дієві ліки 3,6 мільярда доларів.

Як виявилось, з 54 млрд грн, витрачених на ліки в 2017 році, 14 млрд (26%) пішли на гомеопатію та фітотерапію. А препарати цих категорій МОЗ занесло до списку фуфломіцинів, тобто засобів без доведеної ефективності.

Фуфломіцини [1] — тема слизька: вимоги до реєстрації ліків у різних країнах свої. На те, щоб довести ефективність нової діючої речовини, виробнику потрібно витратити кілька років та мільярдів доларів.

Для того, щоб убезпечити простих людей від покупки неефективного препарату, - я вирішив зробити додаток, який видаватиме користувачеві інформацію про препарат із правдивого джерела, в якому знаходяться лише перевірені препарати.

Таким чином, метою даної випускної кваліфікаційної роботи є розробка мобільного програмного забезпечення, яке могло б допомогти простим людям або фармацевтам оперативно дивитися інформацію про препарати просто відсканувавши їх.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В першу чергу потрібно зрозуміти суть і особливості предметної області до системи, яка розробляється.

1.1 Особливості предметної області

Всі ми стикаємося з різними проблемами зі здоров'ям, починаючи від легкої застуди до хронічних захворювань. У всіх випадках нам допомагають медичні препарати, які підвищують імунітет і допомагають організму боротися із хворобами.

Але найчастіше при легкому захворюванні мало хто йде до лікаря за консультацією і починають лікуватися самотужки. Люди йдуть в аптеку та обирають собі препарати для лікування. Ось на цьому етапі головне не зіткнутися з недіючими препаратами, тобто - препарати які не впливають на організм і не допомагають боротися з хворобою.

Проінформовані цією проблемою люди знають, що існує єдина база даних (розділ 1.2), в якій знаходяться лише перевірені медикаменти і будь-яка людина може перевірити назву препарату через цю базу, після чого прийняти рішення, купувати йому таблетки чи ні.

У випадку, якщо людина потрапить на недоброякісний препарат, який медично ніяк не підтверджений, то і лікування залежатиме тільки від неї самої, ґрунтуючись на ефекті Плацебо.

Плацебо — речовина без явних лікувальних властивостей, яка використовується для імітації лікарського засобу в дослідженнях, де ефект, що оцінюється, може бути спотворений вірою самого пацієнта в дієвість препарату, або для поліпшення самопочуття пацієнта у випадках відсутності більш дієвого лікарського засобу [2]. Іноді капсулу або таблетку з плацебо називають пустушкою.

В якості плацебо рекомендується використовувати фізіологічно не активну речовину, яка зовні не відрізняється від експериментального лікарського засобу. Наприклад, при дослідженні ефекту речовини, що діє, в рослинних сумішах потрібно використовувати ту ж суміш без активного компонента. Залежно від постановки експерименту як плацебо можуть використовуватися інші ліки.

Терміном ефект плацебо називають поліпшення самопочуття людини завдяки тому, що він вірить у ефективність деякого впливу, насправді нейтрального [2].

Ефект плацебо ґрунтується на навіюванні [2]. Це навіювання не вимагає будь-яких спеціальних навичок, оскільки критичність свідомості («не вірю») долається шляхом прив'язування інформації до фактичного об'єкта, як правило, — таблетці або ін'єкції без будь-якого реального впливу на організм. Пацієнту повідомляють, що даний препарат має певну позитивну дію на організм, і, незважаючи на неефективність препарату, пацієнт відчуває суб'єктивне поліпшення самопочуття відповідно до очікуваної дії препарату [2].

Аналогічно, очікування негативної дії може спричинити суб'єктивне погіршення самопочуття — ефект ноцебо.

Плацебо може трохи полегшити страждання пацієнта, зменшити біль або нудоту, але в цілому застосування плацебо не може змінити перебігу хвороби, тому терапевтична цінність ефекту плацебо в медицині, за сучасними науковими уявленнями, вкрай мала. Доведено, що плацебо може мати знеболюючий ефект, оскільки сприяє виробленню ендорфінів - при цьому, якщо вироблення ендорфінів у пацієнта заблокувати, знеболюючий ефект плацебо не проявлятиметься. Дослідження ж дії плацебо на інші симптоми показали, що стан пацієнтів за тих чи інших хвороб покращується лише за оцінкою самих пацієнтів; іншими словами, змінюється суб'єктивне судження пацієнта про свій стан, об'єктивні ж показники (лабораторні аналізи і т.п.) в основному не демонструють поліпшення.

Однак деякі виробники випускають свої таблетки в масовий продаж і заявляють, що вони лікують ті чи інші симптоми, а насправді є «пустушка». А люди в свою чергу прийшовши в аптеку і розглядаючи вітрини з таблетками, можуть піддатися гарній упаковці з гарними написами «від головного болю», купити і навіть не підозрювати, що купили пустушку і ніякого ефекту дана таблетка не дасть.

1.2 Сучасний метод контролю неефективних препаратів

На даний момент існує єдине джерело правди, а саме портал «Ліки Контроль» [3], де зібрані тільки перевірені, безпечні та дієві препарати. Щоб подивитись інформацію – необхідно вписати в рядок пошуку назву препарату та подивитися на результат. Якщо нічого знайдено не буде - краще відмовитися від покупки даного препарату

Але не завжди зручно вводити вручну назви препаратів у пошук, особливо коли назви складні та довгі. Наприклад, якщо людина в рукавичках або з поганим зором.

Для вирішення цієї проблеми добре підходить використання камери. Для того щоб вважати назву препарату необхідно лише навести камеру на упаковку і додаток сам знайде назву препарату і виведе користувачеві необхідний результат. Цей спосіб у разі прискорює процес, особливо коли людина, яку хоче дізнатися інформацію про препарат стоїть у черзі і затримує інших людей.

1.3 Висновки по розділу 1

Проаналізовано предметну область мобільного застосунку, який розробляється. Було виявлено, що використання мобільного додатка з використанням технологій розпізнавання тексту та предметів у разі прискорить взаємодію користувача з додатком.

2 ХАРАКТЕРИСТИКИ ОБ'ЄКТА ПРОЕКТУВАННЯ

У цій сфері існує дуже мало аналогів, але слід зазначити два найпопулярніші та перевірені часом.

2.1 Існуючі аналоги системи

2.1.1 Ліки контроль

Найпопулярніший додаток, який рекомендує міністерство охорони здоров'я України – Ліки Контроль (рис. 2.1).

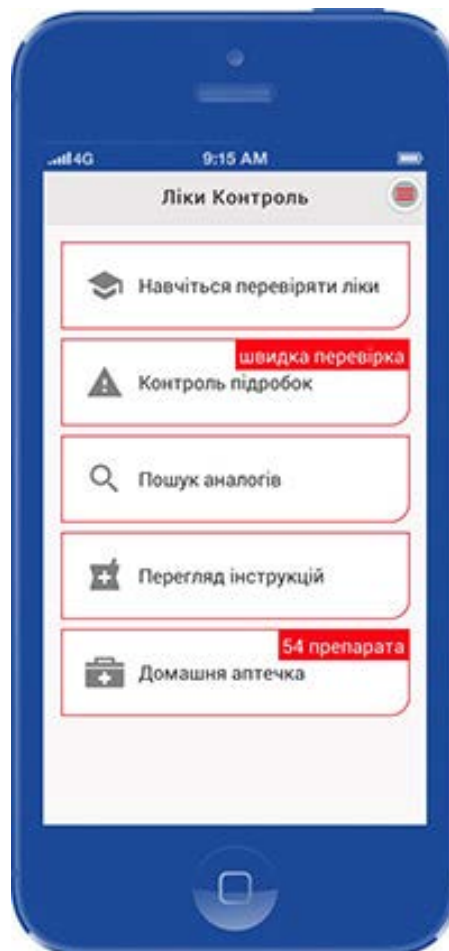


Рисунок 2.1 – Програма Ліки Контроль

Ліки Контроль – це:

- 100% зареєстрованих в Україні лікарських засобів [3];

- тільки офіційні оновлені інструкції без перекладів та скорочень [3];
- економія – знайдіть аналоги за кілька секунд (пошук аналогів за діючою речовиною та по тих виробниках і країнах, яким довіряємо. Виключаємо з пошуку компоненти-алергени) [3];
- захист від підробок – навчитися перевіряти ліки з покроковою інструкцією – не дайте фальсифікату жодного шансу [3];
- домашня аптечка з контролем термінів придатності (інструкції до препаратів можна завантажити на смартфон) [3];
- ліки, які можна отримати БЕЗКОШТОВНО позначені логотипами державних програм [3];
- контроль умов зберігання лікарських засобів [3];
- відчуття впевненості – база даних оновлюється щодня [3].

Має приємний і зручний у використанні інтерфейс, набір необхідних функцій.

Застосунок допомагає шукати інформацію про ліки та замовляти прямо через додаток. Також підтримує державну програму «Доступні ліки» через яку більшість громадян мають можливість замовляти ліки зі знижкою або безкоштовно.

- платформи: IOS і Android, WEB інтерфейс;
- необхідна версія Android 4.4 і вище;
- необхідна версія iOS: 10.0 і вище.

Розберемо докладніше дану систему.

Ця програма має кілька функцій (рисунки 2.2) такі як:

- «Навчіться перевіряти ліки» - навчальний матеріал щодо перевірки ліків на справжність;
- контроль підробок;
- пошук аналогів;
- інструкції;
- домашня аптечка.

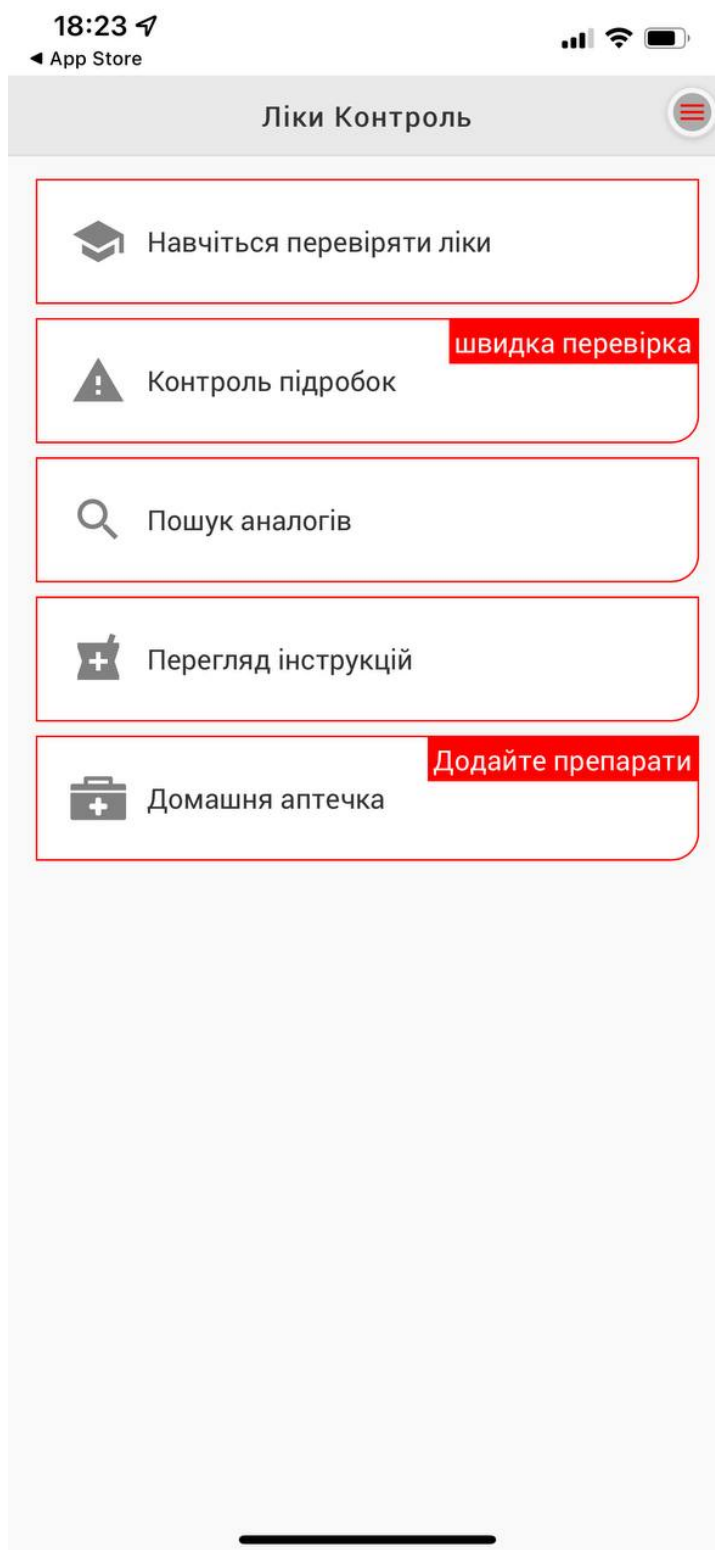


Рисунок 2.2 – Список функцій додатку Ліки контроль

Модуль 1 «Навчіться перевіряти ліки» (рис 2.3).



Рисунок 2.3 – Модуль 1

Містить в собі текст для ознайомлення як відрізнити підробку від оригінала.

Модуль 2 «Контроль підробок» (рис. 2.4).

18:24 17

Перевірка реєстрації

Дізнайтесь, чи зареєстровано препарат в Україні.
Дізнайтесь, чи існують заборонені серії.
Введіть реєстраційний номер.

UA/ / /

Рисунок 2.4 – Модуль 2

Перевіряє справжність препарату за реєстраційним кодом.

Модуль 3 «Пошук аналогів» (рис. 2.5).

18:32   

← Пошук аналогів 

Введіть назву препарату українською мовою або **МНН** англійською

ІБУПРОФЕН 

Знайдено 91 аналог
[Шукати серед аналогів](#)
 Показано аналоги за діючою речовиною.
 Співпадає АТС-код **M01AE01**

АРОФЕН ДЛЯ ДІТЕЙ **інструкція**

супозиторії ректальні, по 60 мг; по 5 супозиторіїв у алюмінієвому стрипі; по 2 стрипи у картонній пачці

Країна-виробник: Франція

Склад діючих речовин: 1 супозиторій містить ібупрофену 60 мг

 [картка препарата](#)

АРОФЕН ДЛЯ ДІТЕЙ **інструкція**

Супозиторії ректальні, по 60 мг № 10 (5x2) у стрипах

Країна-виробник: Франція

Склад діючих речовин: 1 супозиторій містить ібупрофену 60 мг

 [картка препарата](#)

АФФИДА ДЛЯ ДІТЕЙ **інструкція**

суспензія оральна з абрикосовим смаком, 100 мг/5 мл по 100 мл у флаконі; по 1 флакону в комплекті зі шприцом-дозатором у картонній коробці

Країна-виробник: Північна Македонія

Склад діючих речовин: 5 мл суспензії містять

Рисунок 2.5 – Модуль 3

Шукає аналоги до препарату, який користувач вводить у строку пошуку.
Пошук здійснюється за діючою речовиною у пошуковому препараті.

Модуль 4 «Домашня аптечка» (рис 2.6).

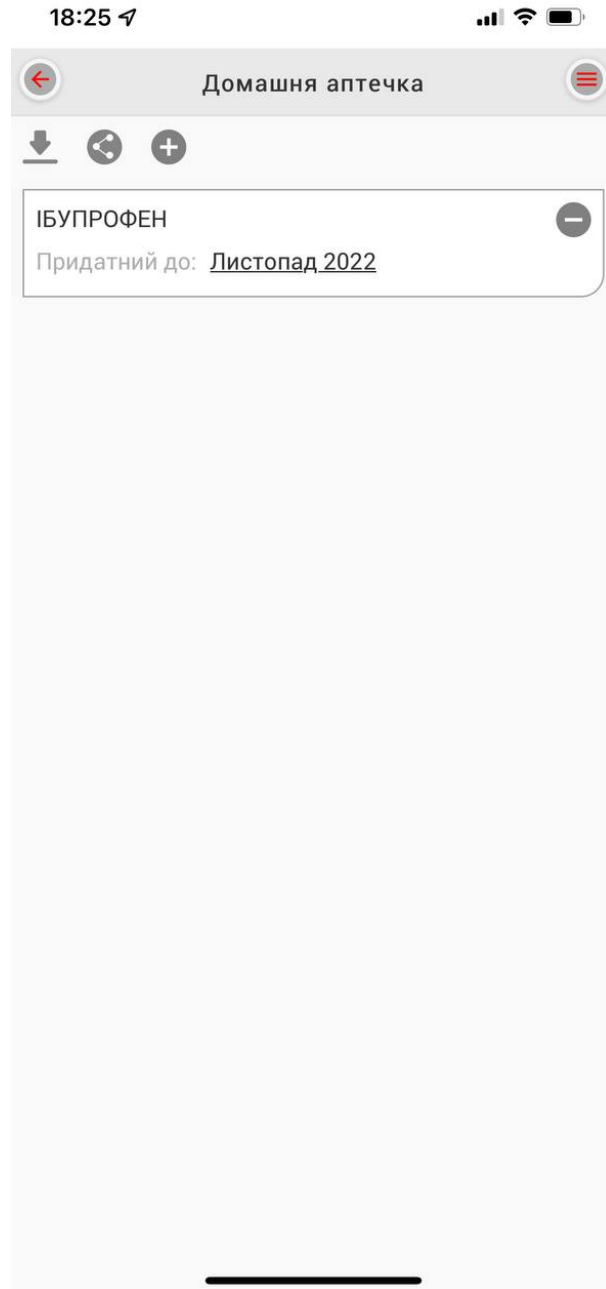


Рисунок 2.6 – Модуль 4

У цьому розділі користувач може додати препарати які купує і налаштувати термін придатності, після якого йому прийде повідомлення на пошту про те, що у такого препарату закінчується термін придатності.

Проаналізувавши цей додаток можна сказати що набір функцій - корисний, але дизайн і інтерфейс користувача незручний і не продуманий. Мобільний додаток більше схожий на мобільну версію сайту. Також до мінусів варто віднести те, що немає можливості сканування штрих-коду препарату.

Останнє оновлення додатка проводилось в 2018 році, це говорить про те, що підтримка більше не здійснюється і, можливо, додаток не буде підтримуватися новими версіями iOS.

2.1.2 Чесний знак

Чесний знак [4] (рис. 2.7) - це простий (але зрозумілий) додаток з «неперевантаженим інтерфейсом» який допомагає у два кроки дізнатися інформацію про потрібний товар. Нажаль функціонал додатка більше націлений не на медичні товари, але така можливість є.

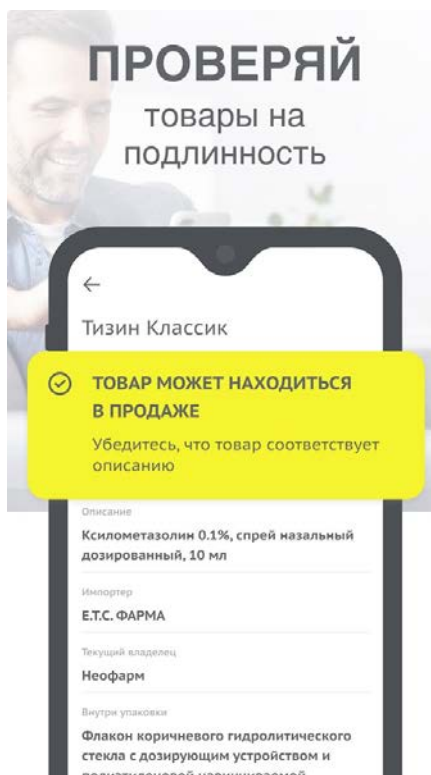


Рисунок 2.7 – Програма Чесний знак

Платформи: IOS, Android.

Дана система має набір корисних функцій:

- перевірка товарів на оригінальність за допомогою сканеру QR або штрих-коду;
- синхронізація. Всі відскановані товари синхронізуються в хмарі і доступні з будь-якого пристрою, за умови реєстрації;
- система повідомлень про порушення якості товарів або підробки, за які користувачу нараховуються бонуси (рис 2.8).

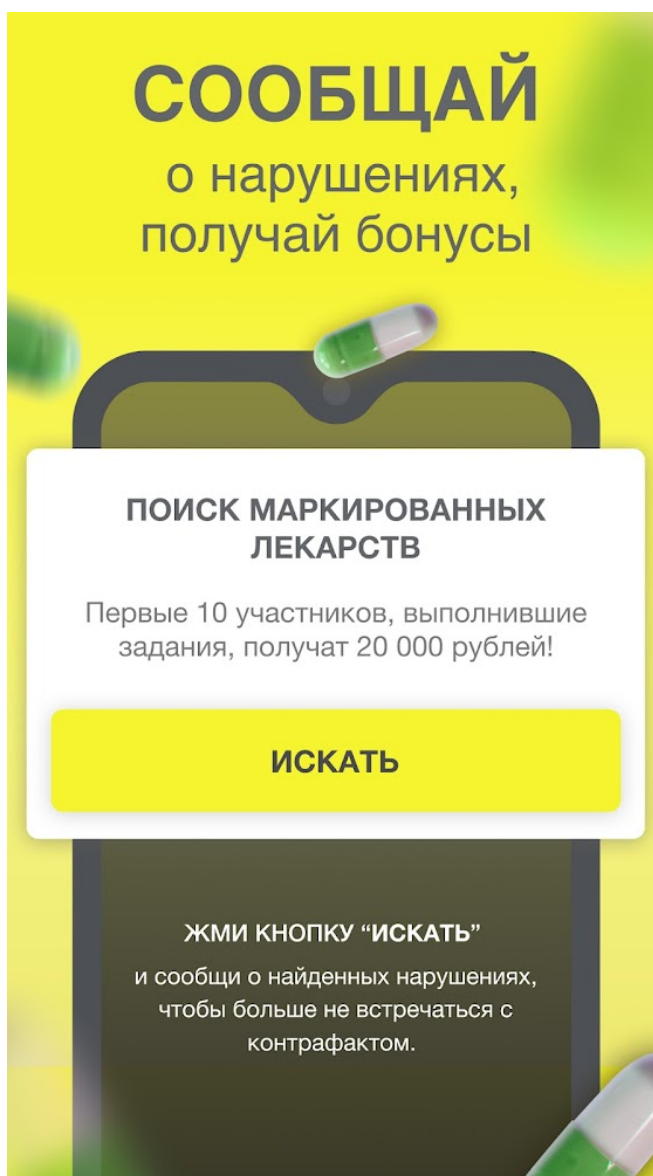


Рисунок 2.8 – Система повідомлень про порушення

До недоліків можна віднести:

- додаток не локалізований під різні мови;
- має обмежену кількість сканувань на місяць, а для розблокування необхідно оформити підписку за 28 грн.

2.2 Порівняння програмних засобів, що розглядалися

Після аналізу систем, що існують на сьогодні, можна зробити висновок, що, хоча вони мають дещо схожий функціонал та можливості, все ж таки користувач, при виборі додатка, віддає перевагу тому, який більше йому підходить по ряду факторів:

- функції додатку - набір дій, які виконує або автоматизує додаток;
- час на повне знайомство з структурою та інтерфейсом користувача - час, який затратив користувач в період розуміння структури та функціонування додатку;
- інтерфейс користувача - зручність виконання дій, доступ до функцій;
- вартість - скільки доведеться заплатити користувачу для того щоб користуватися додатком, або необхідним функціоналом.

Слід враховувати те, що користувач, який не вважає доцільним, у своєму випадку, використовувати весь функціонал додатку - має платити повну ціну, тобто він робить недоцільну покупку, від яких, начебто, цей додаток має допомогти позбутися.

Тому було прийнято рішення щодо створення системи, яка могла дати користувачу всі необхідні можливості за мінімальною ціною, не перевантажуючи інтерфейс користувача, а також була представлена за мінімальною ціною. При цьому щоб користувач мав би можливість спробувати повний функціонал додатку без внесення коштів.

Саме тому з'явилась необхідність у розробці додатку «Tabletki Info», порівняльна характеристика з аналогами якого представлена в таблиці 2.1.

Як можна побачити з порівняльної таблиці аналогів додатку, було враховано переваги та недоліки вже існуючих рішень і система буде вирішувати проблеми, які виникали перед користувачами при виборі додатку для своїх потреб.

Одна з яких – це, те що користувачу не треба переплачувати, якщо йому потрібен не надто широкий функціонал додатку.

При цьому користувач має право користуватись додатком повністю безкоштовно.

Таблиця 2.1 – Порівняльна характеристика додатку «Tabletki Info» з його аналогами.

	Ліки Контроль	Чесний Знак
Можливість шукати товари за допомогою сканеру	-	+
Історія пошуку	+	-
Можливість зберігати товари до «обране»	+	-
Зрозумілий та сучасний інтерфейс користувача	-	+
Можливість ділитися товаром, який знайшов	-	-
Можливість спробувати весь функціонал безкоштовно	+	-
Спільний доступ	+	+

2.3 Висновки по розділу 2

Розглянувши два аналога системи, яка розробляється, можна сказати, що головні критерії такі:

– платформа - пристрій, завдяки якому йому буде максимально комфортно взаємодіяти з програмним забезпеченням;

- функціональні можливості - набір необхідних дій, які «вміє» виконувати програмний продукт;
- синхронізація даних - зберігання даних на хмарі, що дозволяє користувачеві працювати зі своїми даними з різних пристроїв;
- інтерфейс - міст між пристроєм і користувачем. Чим зрозуміліше і в той же час незвичайний інтерфейс, тим більше людей зацікавить використовуваний програмний продукт;
- низька вартість впровадження з максимальною ефективністю системи.

Система, що розробляється, відрізняється пропрацьованим UX дизайном, чого на сьогоднішній день, немає у багатьох аналогів, і невеликою кількістю вікон, що спрощує розуміння програми при роботі з нею [5]. Також, має зрозумілий функціонал.

3 ВИБІР І ОБГРУНТУВАННЯ СТРУКТУРИ ПРОЕКТОВАНОЇ СИСТЕМИ

3.1 Вимоги до програмного забезпечення

Програмне забезпечення призначене для оперативного отримання інформації про таблетки шляхом сканування упаковки за допомогою камери.

Для того, щоб користувач отримав необхідну інформацію про потрібний препарат - йому потрібно відкрити додаток, навести камеру на упаковку і дочекатися поки додаток знайде назву препарату. Як тільки назва буде розпізнана - з'явиться спливаюче вікно з інформацією про препарат.

Також програма підтримує інші формати розпізнавання, такі як:

- шляхом завантаження фотографії з галереї. Такий метод зручний тим, що користувачу необов'язково знаходиться поруч із упаковкою таблеток для того, щоб дізнатися про інформацію. Наприклад, якщо вам відправили фотографію з упаковкою і просять відсканувати - ви можете зберегти її до себе в галерею і з легкістю відкрити через програму і вона розпізнає всю необхідну інформацію;
- шляхом фотографії прямо з програми. Даний метод корисний тим, що не завжди розпізнавання об'єктів та тексту добре працює в режимі реального часу, на це впливає безліч факторів, такі як освітленість, матеріал упаковки та інше. Для більш чіткої картинки можна сфотографувати упаковку прямо через додаток і далі механізму розпізнавання буде легше видати необхідний результат.

3.2 Огляд особливостей мови програмування

При розробці додатку було використано мову програмування Swift.

Swift - це мова програмування компанії Apple, яка була представлена на одній з презентацій компанії у 2014 році [30],[31].

Ця мова пришла на зміну мови програмування для пристроїв компанії Apple Objective-C [6]. Мова Swift відрізняється від свого попередника більш читабельним синтаксисом, більш автоматизованою роботою з пам'яттю та більш високою продуктивністю.

На даний момент мова стрімко розвивається. Самі засновники роблять все можливе, щоб розробники переходили зі старого Objective-C на новий Swift: впроваджують курси, випускають літературу та відеоуроки, спеціальні додатки для ознайомлення з мовою, а також створюють нові бібліотеки (як наприклад бібліотека для роботи з мережею) тільки під мову програмування Swift.

Отже, Swift займає менше часу на розробку додатків ніж Objective-C, але не на стільки, як, наприклад, гібридні способи розробки (ReactNative), що дозволяють виконувати код, який був написаний відразу на двох платформах (IOS та Android), але це за собою несе багато нюансів, бо треба враховувати аспекти обох платформ. Варто зауважити, що в даному випадку немає потреби у розробці додатку одразу для двох платформ.

Для розробки стояв вибір між Swift та Objective-C, вибір упав на Swift. Після того, як Apple оголосили Swift новою мовою програмування, реакція суспільства була неоднозначною. Створена для заміни Objective-C, ця мова iOS програмування вважається кращою за всіх своїх попередників. 7 основних переваг Swift над Objective-C:

- Apple [24][25] та IBM об'єднали свої зусилля і разом інвестують у Swift. IBM Cloud надає можливості не тільки розробляти та застосовувати, але також ділитися ресурсами та використовувати Swift Sandbox для швидких експериментів. Таким чином, Objective-C виявився поза грою. Практично всі зміни, які вносилися до Objective-C останнім часом, робилися для того, щоб спростити імпорт даних у Swift;

- Swift — компактніша мова програмування. Але це зовсім не означає, що код простий. Іноді він дуже складний. Але в той же час він забезпечує переваги, якими не може похвалитися Objective-C;

- синтаксис і будова мови виключають кілька типів помилок, які потенційно можливі Objective-C. Ця стабільність мови допомагає розробнику уникнути небажаних похибок;
- продуктивність Swift наближається до C++. Apple постійно працює над покращенням швидкості Свіфт. Objective-C працює повільніше;
- створений для платформ Apple, Swift потихеньку поширюється на інші платформи. В даний момент він досяг лише Linux. Але прогнозується, що ця мова стане ще портативнішою і переноситиметься на інші платформи. Кажуть, що у планах автора Кріса Латтнера є ідея зробити його сумісним із Android;
- інструмент Swift Playgrounds надав нові можливості розробникам. Він дозволяє тестувати код в режимі реального часу без складання великих шматків або всієї програми. Ця функція відмінно підходить для експериментів із кодом;
- сучасну мову програмування Apple легше зрозуміти розробникам, які не працюють з iOS. Це позитивно впливає на продуктивність та швидкість роботи. Swift надає унікальну можливість - він може бути використаний для всіх цілей програмного забезпечення iOS.

По таблиці порівняння можна помітити, що Swift виграє за багатьма параметрами, давайте розберемо кожен з них (табл. 3.1):

- типобезпека (англ. type safety) мови програмування означає безпеку (або надійність) його системи типів. Система типів називається безпечною (англ. safe) або надійною (англ. sound), якщо у програмах, що пройшли перевірку узгодження типів, (англ. well-typed programs або well-formed programs) виключено можливість виникнення помилок узгодження типів під час виконання. Помилка узгодження типів або помилка типізації (англ. type error) є неузгодженістю типів компонентів виразів у програмі, наприклад спробу використовувати ціле число в ролі функції. Пропущені помилки узгодження типів на етапі виконання можуть призводити до багів і навіть крахів програм.

Безпека мови не є синонімом повної відсутності багів, але щонайменше вони стають досліджуваними в рамках семантики мови (формальної або неформальної);

- run time компілятор дозволяє застосовувати технічну диспетчеризацію класів, методів та структур під час проектування, за рахунок чого час компіляції зменшується, а швидкість виконання програми збільшується;

- ARC (англ. Auto Reference Counter) - автоматичний підрахунок посилань передбачає, що розробнику не потрібно займатися ручним управлінням пам'яті і самому звільняти об'єкти, коли вони не використовуються. Це все виконує ARC.

Таблиця 3.1 – Порівняння Swift та Objective-C

	Swift	Objective-C
Типобезпечна мова програмування	+	-
Швидкий RunTime компілятор	+	-
Управління пам'яттю за допомогою ARC (автоматичний підрахунок ссилки)	+	+/-
Щорічні оновлення	+	-
Підтримка нових популярних фреймворків	+	-
Велика кількість навчального матеріалу у відкритих джерелах	+	+/-

3.3 Огляд особливостей обраної середи розробки

Якщо для розробки мобільного додатку було обрано мову програмування Swift, то список програмного забезпечення для створення рішення звужується до двох пунктів: Xcode [22][23]- офіційна середа розробки від компанії Apple; AppCode - середа розробки від JetBrains [8].

Враховуючи те, що AppCode коштує близько 200 доларів, та не є офіційною середою розробки, то було прийнято рішення використовувати середу розробки Xcode [23].

Xcode - середа розробки рішень для пристроїв компанії Apple на операційній системі:

- iOS;
- macOS;
- watchOS;
- tvOS.

Крім Objective-C та Swift, середа розробки має можливість використовувати такі відомі мови програмування:

- C;
- C++;
- Java [9];
- Ruby [10];
- Python [11];
- AppleScript [12];
- Objective-C [6].

Однією особливістю, яка прискорює розробку додатків в середі Xcode - це Interface Builder (рис 3.1). Це інструмент, який дає можливість створювати інтерфейси користувача вручну, за допомогою перетягування елементів.

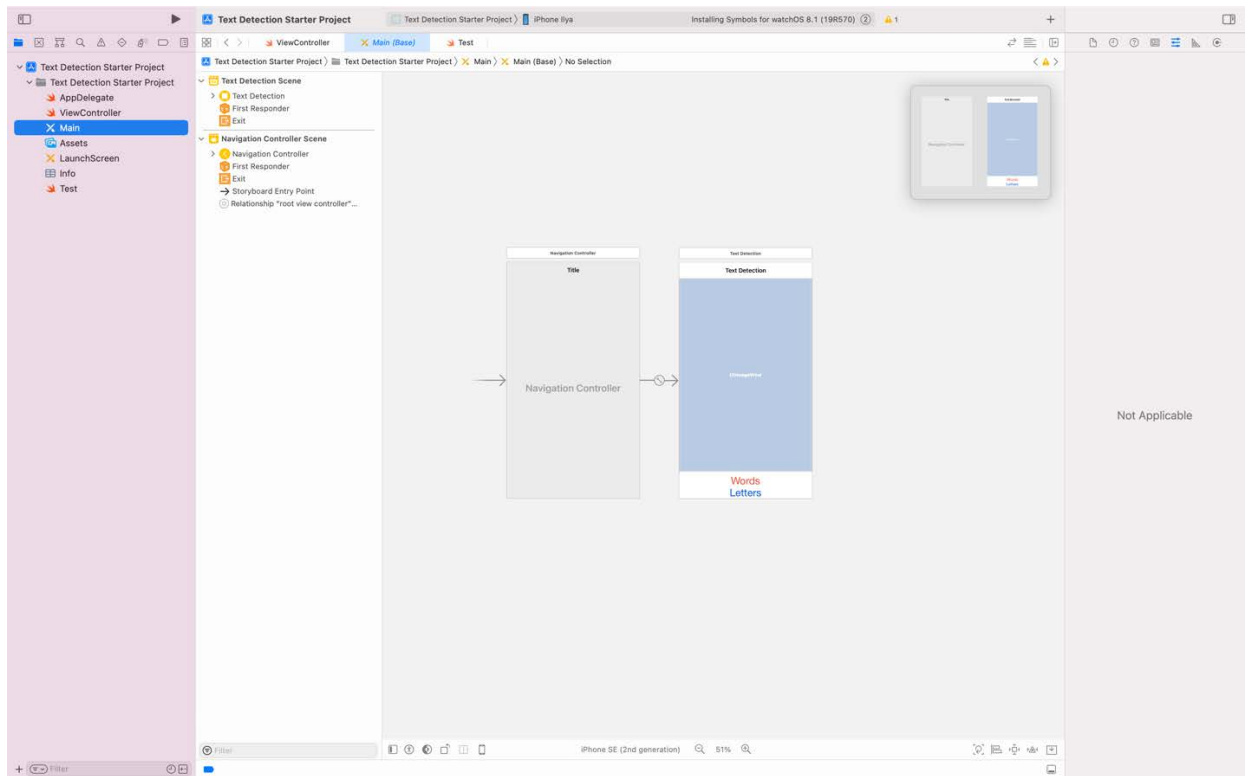


Рисунок 3.1 - Вигляд інструменту Interface Builder

3.4 Опис особливостей програмної реалізації

Для того щоб код був більш гнучкий і таким, що масштабується - було прийнято рішення використовувати патерни проектування, які буде розглянуто у даному підрозділі.

3.4.1 Архітектура програмного продукту

Для зручності навігації у додатку був реалізований паттерн «Координатор».

Координатор - це об'єкт, який відповідає за певний модуль, бере на себе функціонування певної області, яку реалізує. Головним обов'язком координатора - є прийняття рішення, на основі даних, наприклад, який далі показати екран для користувача.

Саме цей підхід було обрано, як головний при розробці додатку.

Також, було прийнято рішення замість відомого патерну MVC (рис. 3.2) використовувати патерн MVP [32](рис. 3.3), в якому акцент робиться вже не на певний контролер, а на презентер, що дає можливість на одному контролері використовувати декілька презентерів для відображення інформації. Більш детальне порівняння представлено у таблиці 3.2.

Слід також враховувати те, що патерн MVC був видозмінений, для проектування рішень у середі Xcode (рис. 3.4), в зв'язку з особливості платформи. Це призвело до певного злиття контролера та виду.

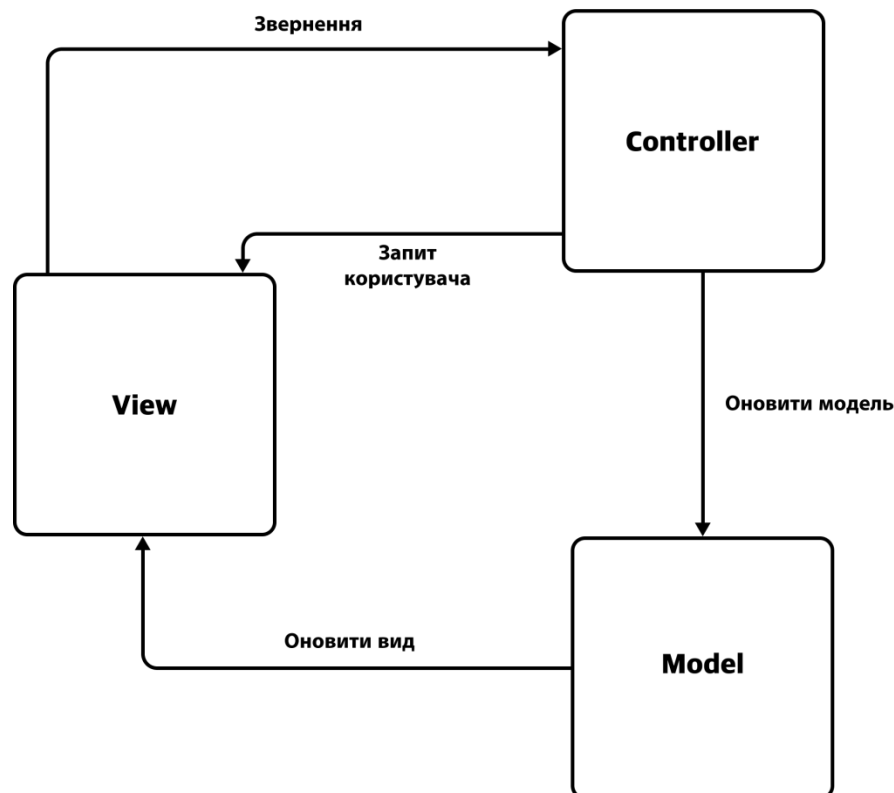


Рисунок 3.2 - Схема патерну MVC

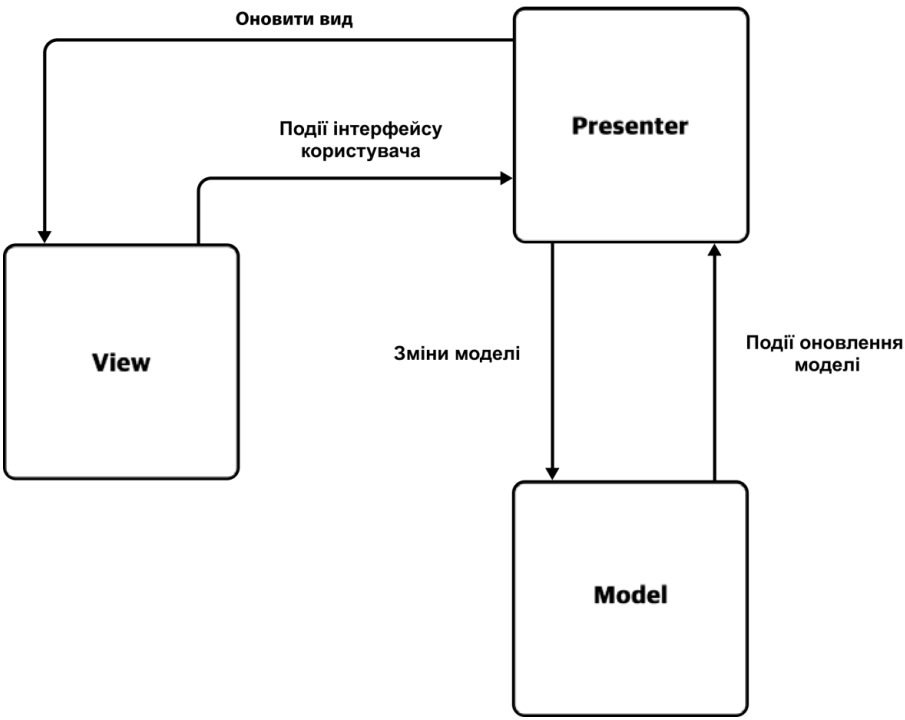


Рисунок 3.3 – Схема патерну MVP

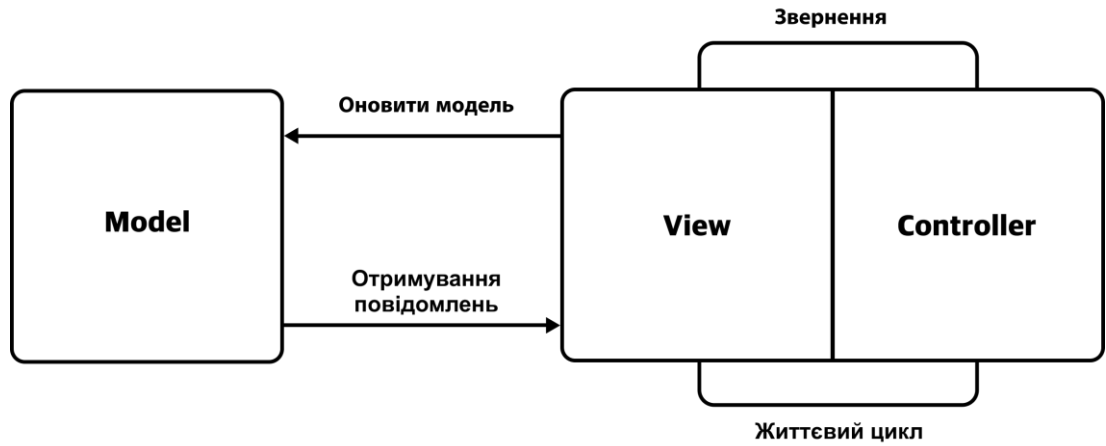


Рисунок 3.4 – Схема патерну MVC (від Apple)

Таблиця 3.2 – Порівняння патернів MVC та MVP

MVP	MVC
Вид не залежить від моделі	Контролер може використовуватись разом з видом
Легше тестувати компоненти	Несе відповідальність за відображення виду
Складні контролери можуть мати декілька презентерів	

3.4.2 Патерни, що використовуються

Патерни зменшують час на розробку систему та на її підтримку у майбутньому.

При роботі у середині розробки Xcode програміст може навіть не замислюватись, як часто він зустрічає патерни в період розробки додатку.

У таблиці 3.3 наведено список патернів, які було використано при розробці додатку.

Таблиця 3.3 – Патерни, що використовувались при розробці

Назва	Приклад використання
Adapter	При використанні Data source, а також класу «MoneyFormatter».
Coordinator [13]	Для управління логікою та послідовністю екранів.
Data source	Для заповнення даних для таких елементів, як таблиці чи колекція плиток.
Delegate	Для передачі інформації між об'єктами та їх взаємодії.
Fabric method	Для отримання певного значення за вказаним параметром («GoalsSorting»).
MVP	Для відображення та отримання інформації з контролерів.
Notification	При відстеженні події, коли з'являється або зникає клавіатура для вводу інформації у контролерах, що наслідуються від «InputViewController».
Observer	
Singleton	При взаємодії з NotificationCenter.

3.4.3 База даних

Для збереження історії знайдених препаратів необхідно використати базу даних.

Серед розробки Xcode пропонує реалізувати це за допомогою інструменту за замовченням Core Data, але було прийнято рішення використовувати бібліотеку Realm, яка має низку переваг (таблиця 3.4).

Таблиця 3.4 – Порівняння Realm та Core Data.

	Realm	Core Data
Легкість у використанні	+	+/-
Швидкодія	+	+/-
Кросплатформеність	+	-
Багато матеріалу для ознайомлення	+	+/-
ПЗ для перегляду бази даних	+	-

Бібліотека встановлюється за допомогою менеджера залежностей. Це може бути Carthage, але у даному випадку було обрано менеджер залежностей Cocoapods.

Realm - це NoSQL база даних, яка була розроблена спеціально для мобільних платформ і вирізняється своєю простотою і швидкодією [17].

3.4.4 Опис структури бази даних

Додаток база даних використовується для збереження історії знайдених препаратів.

У зв'язку з цим структура складається всього з однієї сутності Drug Entity (рис. 3.5).

Attributes

Attribute	Type
S desc	String
S name	String
O photo	Binary Data
N price	Decimal
+ —	

Relationships

Fetches Properties

Рисунок 3.5 – Сутність Drug Entity

Сутність Drug Entity містить основну інформацію про знайдений препарат. Опис властивостей приведений у таблиці 3.5.

Таблиця 3.5 –Властивості Drug Entity

Назва	Тип	Опис
desc	String	Опис препарату у вигляді тексту
name	String	Назва препарату
photo	Binary Data	Фотографія препарату у вигляді завантаженої бінарної картинки
price	Decimal	Ціна за препарат

На рисунку 3.6 зображена структурна схема використання бази даних у додатку

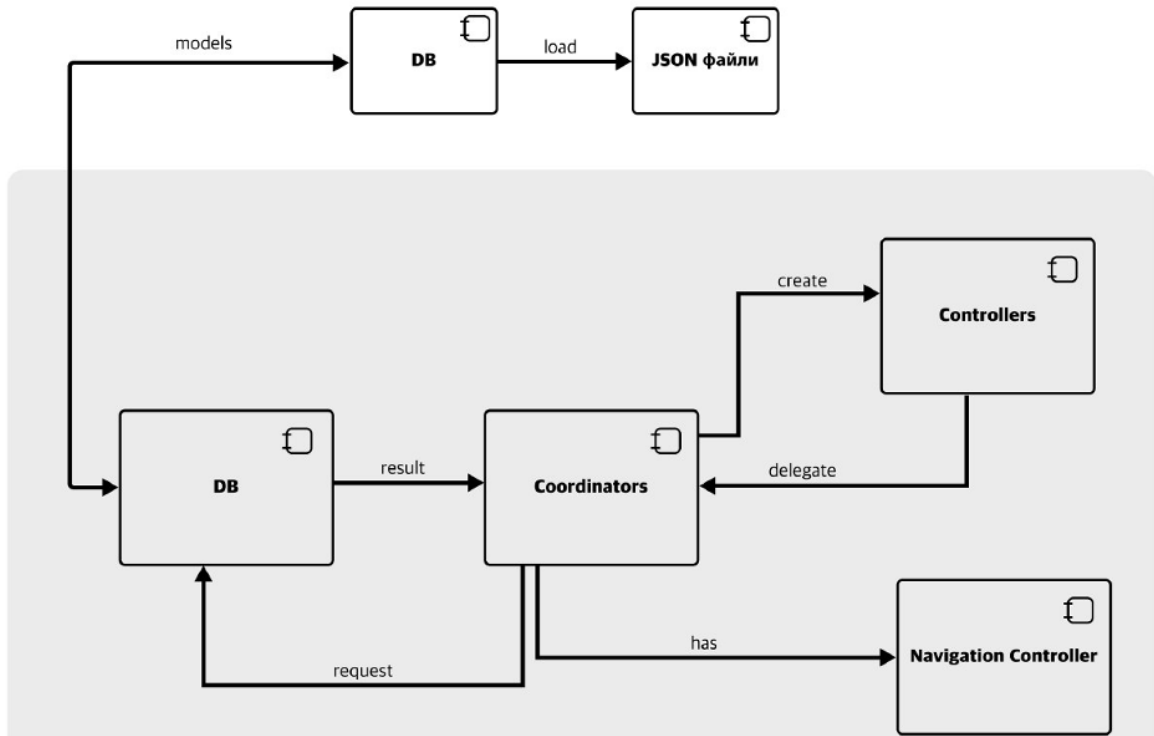


Рисунок 3.6 – Структурна схема використання бази даних

3.5 Способи реалізації розпізнавання упаковок препаратів

Оскільки суть роботи - показати інформацію про препарат користувачеві, програма повинна вміти розпізнавати текст. Після того, як на сканованій поверхні буде розпізнано слова (а не тільки літери), робить запит на API порталу доступні ліки, у відповідь очікується вся інформація про препарат.

На цьому етапі можна зіткнутися з такою проблемою, що робиться занадто багато запитів і час обробки сильно збільшується з кожною секундою використання програми.

Щоб цього уникнути - необхідно додати вибірку до слів, які надсилаються в запиті. Оскільки предмет сканування специфічний та обмежений – було прийнято рішення додати можливість розпізнавання упаковок медичних препаратів. Таким чином у нас спочатку буде здійснено пошук моделі упаковки, потім розпізнавання тексту і лише потім звернення до API.

На рисунку 3.7 зображено повну схему роботи логіки від сканування до доставки шуканих слів до API.

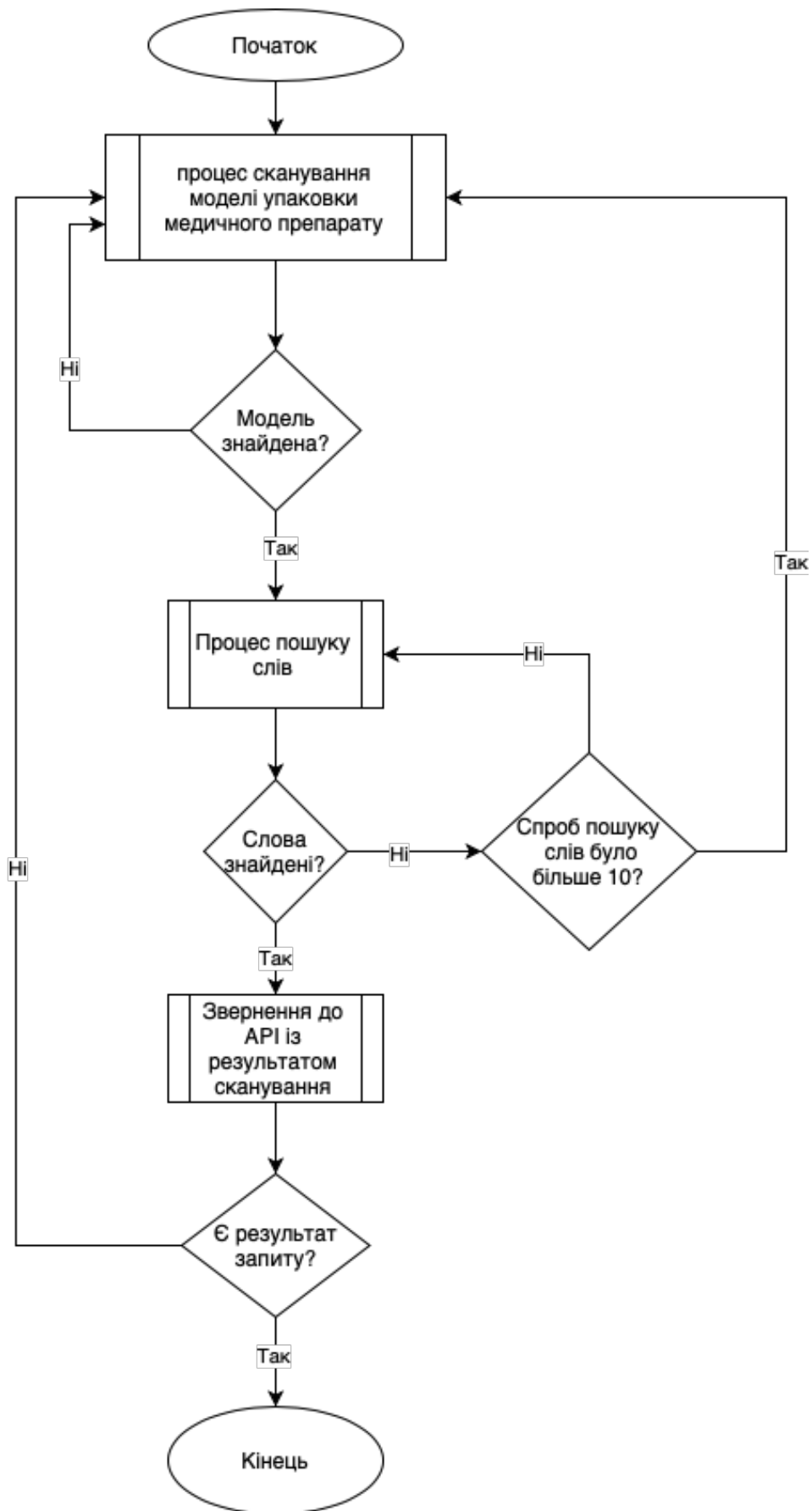


Рисунок 3.7 – Схема роботи доставки результатів сканування до API

Для вирішення завдання розпізнавання упаковок було прийнято рішення використовувати алгоритми машинного навчання. У наступних розділах буде переведено порівняння існуючих алгоритмів та бібліотек, які їх реалізують.

3.6 Порівняння та вибір алгоритмів машинного навчання

У цьому розділі варто розглянути існуючі алгоритми машинного навчання та вибрати найдієвіший для нашого завдання.

3.6.1 Обмежена машина Больцмана

Обмежена машина Больцмана (англ. restricted Boltzmann machine) [15], скорочено RBM - це стохастичні процеси навчання, що мають рекурентну структуру та є основою ранніх методів оптимізації, що використовуються в ANN.

Машина Больцмана була винайдена Джеффри Хінтоном та Террі Сейновськи у 1985 році. Більше ясності можна спостерігати за словами Хінтона про Машину Больцмана.

«Дивною особливістю цієї мережі є те, що вона використовує лише локально доступну інформацію» - Ackley, Hinton 1985.

Декілька важливих моментів про машину Больцмана:

- вони використовують рекурентну структуру;
- вони складаються із стохастичних нейронів, які мають один із двох можливих станів: 1 або 0;
- деякі з нейронів у цьому є адаптивними (вільний стан), а деякі затиснуті (заморожений стан).

Основна мета Boltzmann Machine – оптимізувати вирішення проблеми. Завданням Boltzmann Machine є оптимізація ваги та кількості нейронів, пов'язаних із цією конкретною проблемою.

3.6.2 Автокодувальник

Автокодувальник (англ. autoencoder, також — автоасоціатор) — спеціальна архітектура штучних нейронних мереж, що дозволяє застосовувати навчання без вчителя під час використання методу зворотнього поширення помилки. Найпростіша архітектура автокодувальника - мережа прямого поширення, без зворотніх зв'язків, найбільш схожа з перцептроном і містить вхідний шар, проміжний шар та вихідний шар. На відміну від перцептрону, вихідний шар автокодувальника повинен містити стільки ж нейронів, як і вхідний шар [16].

Основний принцип роботи та навчання мережі автокодувальника — отримати на вихідному шарі відгук, найближчий до вхідного. Щоб рішення не виявилось тривіальним, на проміжний шар автокодувальника накладають обмеження: проміжний шар повинен бути меншою розмірності, ніж вхідний і вихідний шари, або штучно обмежується кількість одночасно активних нейронів проміжного шару - розріджена активація. Ці обмеження змушують нейромережу шукати узагальнення і кореляцію в даних, що надходять на вхід, виконувати їх стиснення. Таким чином, нейромережа автоматично навчається виділяти із вхідних даних загальні ознаки, які кодуються у значеннях ваг штучної нейронної мережі. Так, при навчанні мережі на наборі різних вхідних зображень, нейромережа може самостійно навчитися розпізнавати лінії та смуги під різними кутами [16].

Найчастіше автокодувальники застосовують каскадно для навчання глибоких (багатошарових) мереж. Автокодувальники застосовують для попереднього навчання глибокої мережі без вчителя. Для цього шари навчаються один за одним, починаючи з перших. До кожного нового ненавченого шару на час навчання підключається додатковий вихідний шар, що доповнює мережу до архітектури автокодувальника, після чого на вхід мережі подається набір даних для навчання. Вага ненавченого шару та додаткового шару автокодувальника обробляються за допомогою методу

зворотного розповсюдження помилки. Потім шар автокодувальника відключається і створюється новий, що відповідає наступному ненавченому шару мережі. На вхід мережі знову подається той же набір даних, навчені перші шари мережі залишаються без змін і працюють як вхідні для чергового автокодировщика шару, що навчається. Так навчання продовжується для всіх шарів мережі, крім останніх. Останні шари мережі зазвичай навчаються без використання автокодувальника за допомогою того ж методу зворотного поширення помилки та на маркованих даних (навчання з учителем) [16].

Після того, як цей метод був запропонований у 2006 р. Джеффрі Хінтоном та Русланом Салахутдіновим, досить швидко виявилося, що нових методів ініціалізації випадковими вагами виявляється достатньо для подальшого навчання глибоких мереж. Запропонована у 2014 р. пакетна нормалізація дозволила навчати ще глибші мережі, запропонований наприкінці 2015 р. метод залишкового навчання дозволив навчати мережі довільної глибини [16].

Основними практичними додатками автокодувальників залишаються зменшення шуму даних, а також зменшення розмірності багатовимірних даних для візуалізації. З певними застереженнями, що стосуються розмірності та розрідженості даних, автокодувальники можуть дозволяти отримувати проєкції багатовимірних даних, які виявляються кращими за тих, що дає метод основних компонентів або будь-який інший класичний метод [16].

3.6.3 Згорткова нейронна мережа

Згорткова нейронна мережа (англ. convolutional neural network, CNN) - спеціальна архітектура штучних нейронних мереж, запропонована Яном Лекуном в 1988 році і націлена на ефективне розпізнавання образів, входить до складу технологій глибокого навчання (англ. deep learning). Використовує деякі особливості зорової кори, у якій були відкриті звані прості клітини, реагують на прямі лінії під різними кутами, і складні клітини, реакція яких

пов'язані з активацією певного набору простих клітин. Таким чином, ідея згорткових нейронних мереж полягає в чергуванні згорткових шарів (англ. convolution layers) та субдискретизуючих шарів (англ. subsampling layers або англ. pooling layers, шарів підвиборки) [17].

Структура мережі - односпрямована (без зворотних зв'язків). Для навчання використовуються стандартні методи, найчастіше метод зворотного розповсюдження помилки. Функція активації нейронів (передавальна функція) - будь-яка, на вибір дослідника [17].

Назву архітектура мережі отримала через наявність операції згортки, суть якої в тому, що кожен фрагмент зображення множиться на матрицю (ядро) згортки поелементно, а результат підсумовується та записується в аналогічну позицію вихідного зображення [17].

Архітектура та принцип роботи - у звичайному перцептроні, який є повнозв'язковою нейронною мережею, кожен нейрон пов'язаний з усіма нейронами попереднього шару, причому кожен зв'язок має свій персональний ваговий коефіцієнт [17].

У згортковій нейронній мережі в операції згортки використовується лише обмежена матриця ваг невеликого розміру, яку «рухають» по всьому шару, що обробляється (на самому початку — безпосередньо за вхідним зображенням), формуючи після кожного зсуву сигнал активації для нейрона наступного шару з аналогічною позицією. Тобто для різних нейронів вихідного шару використовуються та сама матриця ваг, яку також називають ядром згортки. Її інтерпретують як графічне кодування будь-якої ознаки, наприклад, наявності похилої лінії під певним кутом. Тоді наступний шар, що вийшов в результаті операції згортки такою матрицею ваг, показує наявність даної ознаки в шарі, що обробляється, і її координати, формуючи так звану карту ознак (англ. feature map) [17].

Природно, у згортковій нейронній мережі набір ваг не один, а ціла гама, що кодує елементи зображення (наприклад, лінії та дуги під різними кутами). У цьому такі ядра згортки не закладаються дослідником заздалегідь, а

формується самостійно шляхом навчання мережі класичним методом зворотного поширення помилки [17].

Прохід кожним набором ваг формує свій власний екземпляр карти ознак, роблячи нейронну мережу багатоканальною (багато незалежних карт ознак на одному шарі). Також слід зазначити, що при переборі шару матрицею ваг її пересувають зазвичай не на повний крок (розмір цієї матриці), а на невелику відстань. Так, наприклад, при розмірності матриці ваги 5×5 її зрушують на один або два нейрони (пікселі) замість п'яти, щоб не «переступити» шуканої ознаки [17].

Операція субдискретизації (англ. subsampling, англ. pooling, що також перекладається як «операція підвиборки» або операція об'єднання), виконує зменшення розмірності сформованих карт ознак. У даній архітектурі мережі вважається, що інформація про факт наявності шуканої ознаки важливіша за точне знання координат, тому з кількох сусідніх нейронів карти ознак вибирається максимальний і приймається за один нейрон ущільненої карти ознак меншої розмірності. За рахунок цієї операції, крім прискорення подальших обчислень, мережа стає інваріантнішою до масштабу вхідного зображення [17].

Розглянемо типову структуру згорткової нейронної мережі докладніше. Мережа складається з великої кількості шарів. Після початкового шару (вхідного зображення) сигнал проходить серію згорткових шарів, в яких чергується власне згортка та субдискретизація (пулінг) [17].

Чергування шарів дозволяє становити «карти ознак» з карт ознак, кожному наступному шарі карта зменшується у вигляді, але збільшується кількість каналів. Насправді це означає здатність розпізнавання складних ієрархій ознак. Зазвичай після проходження кількох шарів карта ознак вироджується у вектор або навіть скаляр, але таких карт ознак стають сотні. На виході згорткових шарів мережі додатково встановлюють кілька шарів нейронної повнозв'язкової мережі (перцептрон), на вхід якому подаються кінцеві карти ознак [17].

Переваги алгоритму:

- один з кращих алгоритмів розпізнавання та класифікації зображень [33];
- у порівнянні з повнозв'язною нейронною мережею (типу перцептрону) — набагато менша кількість ваг, що настроюються, так як одне ядро ваг використовується цілком для всього зображення, замість того, щоб робити для кожного пікселя вхідного зображення свої персональні вагові коефіцієнти. Це підштовхує нейромережу при навчанні до узагальнення інформації, що демонструється, а не попінкельного запам'ятовування кожної показаної картинки в міриадах вагових коефіцієнтів, як це робить перцептрон [33];
- зручне розпаралелювання обчислень, а отже, можливість реалізації алгоритмів роботи та навчання мережі на графічних процесорах [33];
- відносна стійкість до повороту та зсуву зображення, що розпізнається [33];
- навчання за допомогою класичного методу зворотного розповсюдження помилки [33].

До недоліків можна віднести те, що занадто багато параметрів мережі, що варіюються; незрозуміло, для якого завдання та обчислювальної потужності які потрібні налаштування [17].

Так, до параметрів, що варіюються, можна віднести: кількість шарів, розмірність ядра згортки для кожного з шарів, кількість ядер для кожного з шарів, крок зсуву ядра при обробці шару, необхідність шарів субдискретизації, ступінь зменшення ними розмірності, функція зменшення розмірності (вибір максимуму, середнього тощо), передатна функція нейронів, наявність і параметри вихідної повнозв'язкової нейромережі на виході згорткової [17].

Всі ці параметри суттєво впливають на результат, але вибираються дослідниками емпірично. Існує кілька вивірених і добре працюючих

конфігурацій мереж, але не вистачає рекомендацій, за якими потрібно будувати мережу для нового завдання [17].

3.6.4 Fully-connected neural networks

Приклад повної нейронної мережі можна побачити на рисунку 3.8.

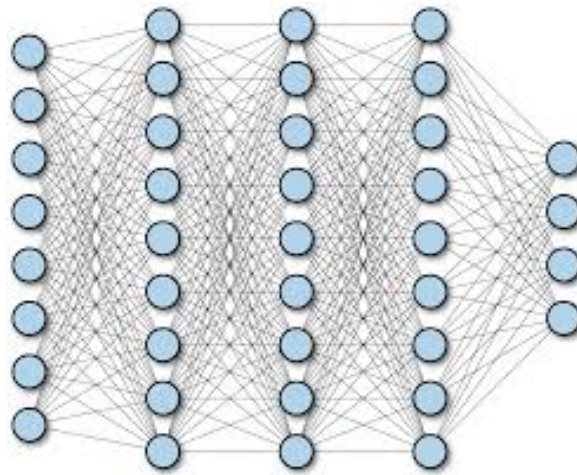


Рисунок 3.8 – Приклад повної нейронної мережі

Суть у тому, що кожен нейрон одного шару пов'язаний з кожним нейроном наступного та попереднього (якщо вони є) [18].

Перший шар – вхідний. Наприклад, якщо ми хочемо подати зображення $256 \times 256 \times 3$ на вхід такої мережі, то рівно $256 \times 256 \times 3$ нейронів у вхідному шарі нам і знадобиться (кожен нейрон прийматиме 1 компонент (R, G або B) пікселя). Якщо хочемо подати зріст людини, її вагу та ще 23 ознаки, то знадобиться 25 нейронів у вхідному шарі. Кількість нейронів на виході - кількість ознак, які ми хочемо передбачити. Це може бути як одна ознака, так і всі сто. У загальному випадку по вихідному шару мережі можна майже напевно сказати - яке завдання вона вирішує [18].

Задачі, які може вирішувати FCNN:

- завдання регресії. Наприклад, передбачення вартості магазину за якимись вхідними критеріями типу країни, міста, вулиці, прохідності тощо [18];

- завдання класифікації. Наприклад, класика - MNIST classification [18];

Недоліки FCNN:

- у нейронах одного шару немає «загальної» інформації (всі ваги у мережі унікальні) [18];
- величезна кількість навчальних параметрів (ваг), якщо ми хочемо навчати мережу по фотографіях [18].

3.7 Огляд програмних засобів глибокого навчання

Існує безліч програмних засобів на вирішення завдань глибокого навчання. У таблиці 3.6 можна знайти загальне порівняння функціональних можливостей найвідоміших. Перші шість програмних бібліотек реалізують найширший спектр методів глибокого навчання. Розробники надають можливості для створення пов'язаних нейромереж (fully connected neural network, FC NN), згорткових нейронних мереж (convolutional neural network, CNN), автокодувальників (autoencoder, AE) та обмежених машин Больцмана (restricted Boltzmann machine, RBM) [18].

Необхідно звернути увагу на бібліотеки, що залишилися. Незважаючи на те, що вони мають меншу функціональність, у деяких випадках їх простота допомагає досягти більшої продуктивності [18].

Таблиця 3.6 – Порівняння Realm та Core Data.

Назва	Мова	ОС	FC NN	CNN	AE	RBM
DeepLearnTool box [19]	Matlab	Windows, Linux	+	+	+	+
Pylearn2 [19]	Python	Linux, Vagrant	+	+	+	+

Продовження таблиці 3.6

Назва	Мова	ОС	FC NN	CNN	AE	RBM
Deepnet [19]	Python	Linux	+	+	+	+
Deepmat [19]	Matlab	?	+	+	+	+
Torch [19]	Lua, C	Linux, Mac OS X, iOS, Android	+	+	+	+
Theano [19]	Python	Windows, Linux, Mac	+	+	+	+
Darch	R	Windows, Linux	+	-	+	+
Caffe	C++	Linux, OS X	+	+	-	-
mnForge	C++	Linux	+	+	-	-
CXXNET	C++	Linux	+	+	-	-
Cuda-convnet	C++	Windows, Linux	+	+	-	-
Cuda CNN	Matlab	Windows, Linux	+	+	-	-
CoreML, Vision	C, Objective-C, Swift	Mac OS X, iOS	+	+	+	+

Оскільки ми розробляємо додаток під мобільний пристрій на операційній системі iOS, слід розглядати ті програмні рішення, які підтримують iOS. У нашому випадку це CoreML та Torch.

3.7.1 Бібліотека Torch

Torch – бібліотека для наукових обчислень із широкою підтримкою алгоритмів машинного навчання. Розробляється Idiap Research Institute, New York University та NEC Laboratories America, починаючи з 2000 р., поширюється під ліцензією BSD [19].

Бібліотека реалізована мовою Lua з використанням C та CUDA. Швидка скриптова мова Lua разом із технологіями SSE, OpenMP, CUDA дозволяють Torch показувати непогану швидкість порівняно з іншими бібліотеками. На даний момент підтримуються операційні системи Linux, FreeBSD, Mac OS X. Основні модулі працюють і на Windows. У залежностях Torch знаходяться пакети imagemagick, gnuplot, nodejs, npm та інші [19].

Бібліотека складається з набору модулів, кожен із яких відповідає за різні стадії роботи з нейромережами. Так, наприклад, модуль nn забезпечує конфігурування нейромережі (визначення шарів, та їх параметрів), модуль optim містить реалізації різних методів оптимізації, що застосовуються для навчання, а gnuplot надає можливість візуалізації даних (побудова графіків, показ зображень і т.д.). Встановлення додаткових модулів дає змогу розширити функціонал бібліотеки.

Torch дозволяє створювати складні нейромережі за допомогою механізму контейнерів. Контейнер - це клас, що поєднує оголошені компоненти нейромережі в одну загальну конфігурацію, яка може бути передана в процедуру навчання. Компонентом нейромережі можуть бути не тільки повнозв'язкові або згорткові шари, а й функції активації чи помилки, а також готові контейнери. Torch дозволяє створювати такі шари:

- повнозв'язковий шар (Linear);
- функції активації: гіперболічний тангенс (Tanh), вибір мінімального (Min) чи максимального (Max), softmax-функція (SoftMax) та інші;

- згорткові шари: згортка (Convolution), проріджування (SubSampling), просторове поєднання (MaxPooling, AveragePooling, LPPooling), різницева нормалізація (SubtractiveNormalization).

Під час навчання можуть використовуватися такі методи оптимізації:

- стохастичний градієнтний спуск (SGD);
- середній стохастичний градієнтний спуск (Averaged SGD) [19];
- алгоритм Бройдена – Флетчера – Гольдфарба – Шанно (L-BFGS) [19];
- метод сполучених градієнтів (Conjugate Gradient, CG). [34]

3.7.2 CoreML

Core ML – це бібліотека машинного навчання, випущена у світ Apple на WWDC 2017.

Вона дозволяє iOS розробникам додавати у свої програми персоналізований досвід у режимі реального часу з використанням передових локальних моделей машинного навчання за допомогою Neural Engine.

12 вересня 2017 року Apple представила світу чіп A11 Bionic (рис 3.9) з Neural Engine. Це нейромережеве апаратне забезпечення може виконувати до 600 базових операцій на секунду (BOPS) і використовується для FaceID, Animoji та інших завдань машинного навчання. Розробники можуть використовувати Neural Engine за допомогою Core ML API.

Core ML оптимізує продуктивність на пристрої за рахунок використання ресурсів центрального процесора, графічного процесора та Neural Engine, мінімізуючи обсяг пам'яті та енергоспоживання.

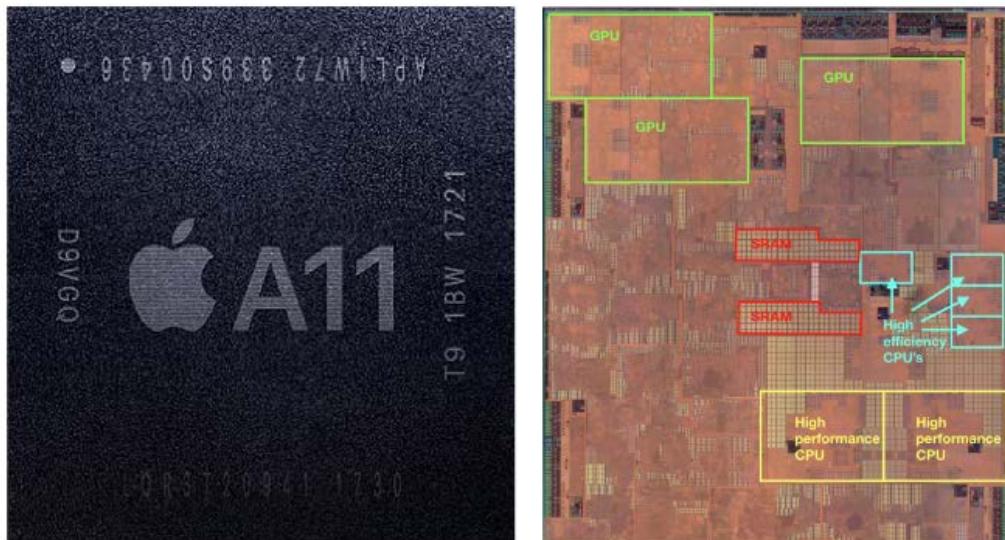


Рисунок 3.9 – чіп A11 Bionic

Начинка чіпа A11 Bionic:

- кількість транзисторів: 4.3 мільярди;
- кількість ядер: 6 ядер ARM (64 біти) – 2 високочастотні (2.4 ГГц) 4 з низьким енергоспоживанням;
- кількість графічних процесорів: 3;
- neural engine – 600 базових операцій за секунду.

Варто відзначити, що запуск моделі локально на пристрої користувача усуває необхідність у мережному підключенні, що допомагає зберігати конфіденційність даних користувача та покращує відгук вашої програми [35].

Core ML є основою для фреймворків та функціоналу даної предметної галузі. Core ML підтримує Vision для аналізу зображень, Natural Language для обробки тексту, Speech для перетворення звуку на текст і Sound Analysis для ідентифікації звуків в аудіо (рис. 3.10).

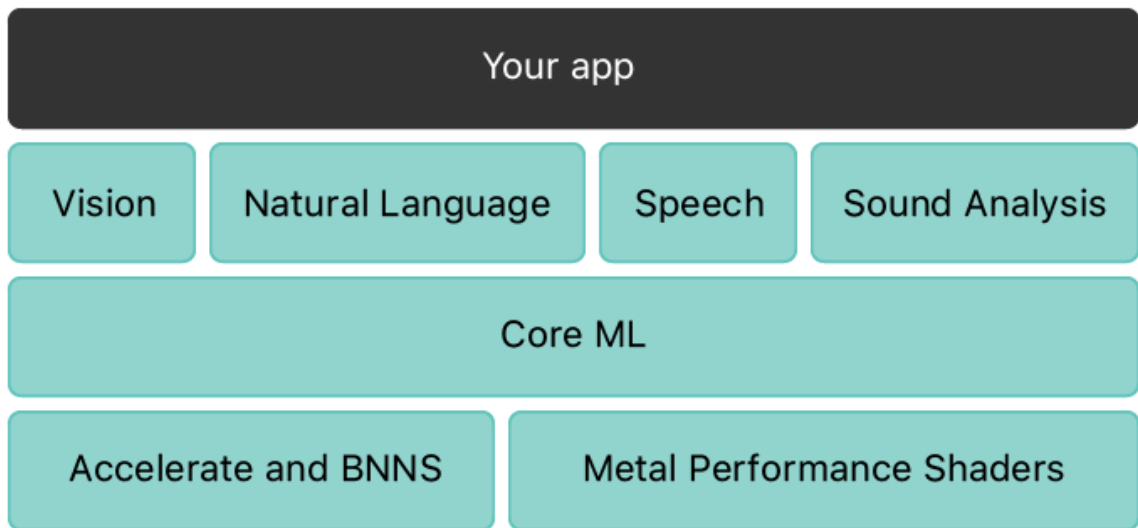


Рисунок 3.10 – Структура та функції CoreML

Рекомендації від Apple зі збору даних:

- використовуйте щонайменше 10 зображень на категорію — чим більше, тим краще;
- уникайте незбалансованих наборів даних, підготувавши приблизно рівну кількість зображень на кожну категорію;
- зробіть вашу модель більш надійною, увімкнувши параметри Create ML UI's Augmentation: Crop, Rotate, Blur, Expose, Noise та Flip;
- знімайте багато зображень під різними кутами, з різним фоном та в різних умовах освітлення. Імітуйте реальні умови зйомки, включаючи шум та розмитість зображення;
- сфотографуйте зразки об'єктів у руках, щоб імітувати реальних користувачів, які намагаються класифікувати об'єкти у руках;
- виключайте з поля зору інші об'єкти, особливо ті, які потрібно класифікувати по-іншому [35].

3.8 Порівняння швидкості розпізнавання бібліотек Torch та CoreML

Щоб порівняти швидкість розпізнавання у різних бібліотеках необхідно вимірювати швидкість практично, за однакових умов.

Я перевірятиму швидкість розпізнавання однієї картинки, на одному пристрої в двох різних бібліотеках.

Спочатку перевіримо бібліотеку CoreML. На рисунку 3.11 зображено код, який фіксує час до початку розпізнавання та після. Для правдивості повторимо процес декілька раз.

```
let startTimeCM = CACurrentMediaTime()
let prediction = try self.mlmodel.prediction(image: img)
let iout = prediction.iout
let endTimeCM = CACurrentMediaTime()
let intervalCM = 1000.0*(endTimeCM - startTimeCM)
debugText = String(format: "[CoreML] Prediction time=%.1f ms", intervalCM)
print(debugText)
```

Рисунок 3.11 – Фіксація часу процесу розпізнавання CoreML

Результат можна побачити на рисунку 3.12.

```
[CoreML] Prediction time=54.45189044262323
[CoreML] Prediction time=73.59831611042925
[CoreML] Prediction time=56.754243819766764
[CoreML] Prediction time=87.44707444216232
[CoreML] Prediction time=51.839737264402665
[CoreML] Prediction time=89.35784846176374
[CoreML] Prediction time=60.38116464528781
[CoreML] Prediction time=65.7635545135874
[CoreML] Prediction time=66.26105629666768
[CoreML] Prediction time=70.42502622856624
[CoreML] Prediction time=60.19432582053865
[CoreML] Prediction time=62.36272081628698
[CoreML] Prediction time=50.37441015913821
[CoreML] Prediction time=80.8854346557684
[CoreML] Prediction time=74.28055399966266
[CoreML] Prediction time=69.23957195822925
[CoreML] Prediction time=80.21843686758785

Average time = 67.8726686177923
```

Рисунок 3.12 – Результат фіксації часу процесу розпізнавання CoreML

На рисунку 3.13 код для вимірювання часу сканування бібліотеки Torch.

```
def load_split_train_test(datadir, valid_size = .2):
    train_transforms = transforms.Compose([transforms.Resize(224),
                                          transforms.ToTensor(),
                                          ])

    test_transforms = transforms.Compose([transforms.Resize(224),
                                          transforms.ToTensor(),
                                          ])

    train_data = datasets.ImageFolder(datadir,
                                     transform=train_transforms)
    test_data = datasets.ImageFolder(datadir,
                                    transform=test_transforms)

    num_train = len(train_data)
    indices = list(range(num_train))
    split = int(np.floor(valid_size * num_train))
    np.random.shuffle(indices)
    from torch.utils.data.sampler import SubsetRandomSampler
    train_idx, test_idx = indices[split:], indices[:split]
    train_sampler = SubsetRandomSampler(train_idx)
    test_sampler = SubsetRandomSampler(test_idx)
    trainloader = torch.utils.data.DataLoader(train_data,
                                             sampler=train_sampler, batch_size=64)
    testloader = torch.utils.data.DataLoader(test_data,
                                             sampler=test_sampler, batch_size=64)
    return trainloader, testloader

trainloader, testloader = load_split_train_test(data_dir, .2)
print(trainloader.dataset.classes)
```

Рисунок 3.13 – Фіксація часу процесу розпізнавання Torch

```
[Torch] Prediction time=78.48139342444972
[Torch] Prediction time=111.42356016038579
[Torch] Prediction time=117.1244091613462
[Torch] Prediction time=95.42581356731066
[Torch] Prediction time=115.74438934218655
[Torch] Prediction time=111.98219613472676
[Torch] Prediction time=76.7437836065644
[Torch] Prediction time=93.43704642250697
[Torch] Prediction time=96.761224772889
[Torch] Prediction time=100.71241148492308
[Torch] Prediction time=73.6650874146288
[Torch] Prediction time=83.78280858911208
[Torch] Prediction time=104.21526447871173
[Torch] Prediction time=116.9344077749797
[Torch] Prediction time=101.6869899881899
[Torch] Prediction time=110.89444689137477
[Torch] Prediction time=53.525940295610305

Average time = 96.62006902999393
```

Рисунок 3.14 – Результат фіксації часу процесу розпізнавання Torch

У процесі вимірів часу розпізнавання було перевірено аналіз якості та виведено кількість помилок при розпізнаванні.

Для цього було підключено логування та відстеження помилок. Щоразу при успішному або невдалому скануванні файл логів заноситься запис. Результати роботи CoreML можна побачити на рисунку 3.15, результати роботи Torch зображені на рисунку 3.16

```
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Recognition failed, There was a problem starting recording
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Recognition failed, There was a problem starting recording
Recognition failed, There was a problem starting recording
Model successfully recognized
Model successfully recognized
Model successfully recognized
Recognition failed, There was a problem starting recording
Recognition failed, There was a problem starting recording

Success: 17
Error: 5
```

Рисунок 3.15 – Кількість успішних розпізнавань бібліотекою CoreML

☐

```

Model successfully recognized
Recognition failed, Code=203
Model successfully recognized
Recognition failed, Code=203
Model successfully recognized
Model successfully recognized
Recognition failed, Code=203
Model successfully recognized
Recognition failed, Code=203
Model successfully recognized
Recognition failed, Code=203
Recognition failed, Code=203
Model successfully recognized
Model successfully recognized
Recognition failed, Code=203
Model successfully recognized
Recognition failed, Code=203
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized
Model successfully recognized

```

Success: 14
Error: 8

Рисунок 3.16 – Кількість успішних розпізнавань бібліотекою Torch

Проаналізувавши кількість помилок при розпізнаванні та швидкість процесу сканування виведемо результати до таблиці 3.7.

Можна побачити що бібліотека CoreML відпрацьовує краще, ніж Torch, а саме CoreML має на 37.5% помилок менше при розпізнаванні, ніж Torch.

Таблиця 3.7 – Порівняння часу розпізнавання та якість бібліотек CoreML та Torch.

	CoreML	Torch
Середній час розпізнавання	67.87ms	96.62ms
Кількість помилок при розпізнаванні	Помилки: 5 Успішно: 17	Помилки: 8 Успішно: 14

3.9 Модернізація алгоритму розпізнавання

Результати використання бібліотек показали, що CoreML лідирує за такими параметрами, як швидкість та якість розпізнавання, тому будемо використовувати цю бібліотеку. Як відомо, вона підтримує всі класичні алгоритми машинного навчання.

Розробник може сам вибирати, який алгоритм використовувати, залежно від потреб, але за стандартом використовується алгоритм CNN.

Якщо коротко описати, процес розпізнавання проходить у два етапи. Сканування об'єкта, складання моделі. Потім дана модель потрапляє на другий етап – розпізнавання об'єкта шляхом порівнювання та аналізу.

Такий самий процес розпізнавання працює і для алгоритму FC NN.

Зробимо вимірювання повного циклу розпізнавання кожного алгоритму. На рисунку 3.17 можна побачити що з використання алгоритму CNN час розпізнавання моделі нижче, ніж у алгоритмі FC NN.

```

Model scan started
.
..
...
....
.....
.....
.....
.....
.....
.....
.....
Model scan finished with time: 35.2312ms

*****

Recognition started
.
..
...
....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
Recognition finished with time: 47.847512ms

```

Рисунок 3.17 – Вимірювання часу циклу розпізнавання з алгоритмом CNN

На другому алгоритмі (рис 3.18) швидкість сканування моделі краще, ніж у алгоритмі CNN.

```

Model scan started
.
..
...
....
.....
.....
.....
.....
.....
.....
.....
Model scan finished with time: 23.1092ms

*****

Recognition started
.
..
...
....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
Recognition finished with time: 68.847512ms

```

Рисунок 3.18 – Вимірювання часу циклу розпізнавання з алгоритмом FC NN

Таблиця 3.8 – Порівняння швидкості розпізнавання алгоритмів CNN та FC NN

	CNN	FC NN
Час сканування моделі	35.2312 мс.	23.1092 мс.
Час розпізнавання моделі	47.85 мс.	68.847512 мс.

З результатів, які вказані в таблиці 3.8 виходить, що для етапу сканування моделі краще відпрацьовує алгоритм FC NN, а для розпізнавання - CNN.

Об'єднавши ці два алгоритми, нам вдасться зменшити час процесу сканування на 36.56%.

Це вдалося реалізувати з допомогою комбінування двох алгоритмів. Для цього нам доведеться модифікувати один з них, а саме процес сканування проведемо за допомогою алгоритму FC NN, після чого роботу алгоритму перериваємо і отриману модель передаємо алгоритму CNN на фінальний етап - процес порівнювання вхідної моделі, яку ми отримали з попереднього кроку з вихідною моделлю.

```

Model scan started
.
..
...
....
.....
.....
.....
.....
.....
.....
.....
Model scan finished with time: 25.5312ms

*****

Recognition started
.
..
...
....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
Recognition finished with time: 39.5812ms

```

Рисунок 3.19 – Результат комбінування алгоритмів CNN та FC NN

У таблиці 3.9 відображено результат модифікації алгоритму, час вказаний у мілісекундах.

Таблиця 3.9 – Результати модифікації алгоритмів розпізнавання

Алгоритм	Час всього циклу розпізнавання
CNN	83.0812 ms.
FC NN	91.9592 ms.
CNN + FC NN	65.1124 ms.

3.10 Висновки по розділу 3

Порівнявши бібліотеки Torch і CoreML можна зробити висновок, що для наших потреб більше підходить CoreML, оскільки ця бібліотека нова вийшла в 2017 році і також є нативною бібліотекою, яка використовується на пристроях компанії Apple. Завдяки використанню ресурсів центрального процесора та популярних алгоритмів машинного навчання - гарантується швидкодія розпізнавання об'єктів, текстів, мовлення та іншого.

Після проведення тестів швидкості розпізнавання виявили, що бібліотека CoreML виконує свою роботу на 42% швидше, ніж Torch.

Також нам вдалося модифікувати класичні алгоритми розпізнавання, які використовуються в бібліотеці Core ML шляхом комбінування двох алгоритмів CNN та FC NN. В результаті ми отримали ідеальне співвідношення швидкості розпізнавання та його якості.

Нам вдалося прискорити процес розпізнавання на 36.56%. Всі дослідження щодо покращення швидкості розпізнавання проводилися практичним шляхом.

CoreML працює на вдосконалених алгоритмах Accelerate, FC NN та CNN. Удосконалені вони тим, що використовуються ресурси не CPU + GPU, а лише CPU. Це можливо завдяки чіпам від Apple, які розробляються за технологією Bionic і здатні обробляти 600+ операцій за секунду. Завдяки використанню CPU не завантажується пам'ять і не йде надмірна витрата заряду акумулятора і через сукупність цих факторів виходить, що під час

розпізнавання об'єктів телефон не розряджається швидше, не починає підвисати.

Також є можливість завантаження автономних моделей і класифікацій для розпізнавання специфічних об'єктів.

Після аналізу предметної області було проаналізовано і розроблено архітектуру програми. Програмна реалізація виконана на об'єктно-орієнтованій мові Swift. Дана мова використовується для розробки додатків для Apple платформ. Як середовище розробки був використаний Xcode, як офіційне середовище розробки.

В якості загальної структури додатку було обрано патерни Coordinator та MVP.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Інформація про призначення та функціонал програми

Розроблений програмний продукт призначений для швидкого пошуку інформації про медикаменти шляхом сканування упаковки від препарату.

Також даний продукт можна використовувати у аптеках для фармацевтів-стажерів для швидкого визначення складу препаратів та іншої інформації про медикаменти.

4.2 Характеристики програми

Додаток розроблено для пристроїв операційної системи IOS, версія якої не повинна бути нижче 13.0.

4.3 Середовище розробки та тестування

Для запуску програмного забезпечення на симуляторі – необхідно мати Xcode з версією не нижче 12.5.1.

Unit та UI тести були написані з використанням пристрою iPhone 13, тому для запуску XCTests потрібно обирати iPhone 13 у списку пристроїв та симуляторів (рис. 4.1)



Рисунок 4.1 Список активних пристроїв та симуляторів Xcode

4.4 Функції програми

Додаток дозволяє:

- сканувати у режимі реального часу;
- сканувати через завантаження зображення з галереї;

– сканувати за допомогою фотографії через сам додаток, без використання інших програм для фотографування.

4.5 Менеджери залежностей

Для полегшення розробки є сторонні бібліотеки. Для встановлення даних бібліотек у проєкт використовують різні менеджери залежностей, про які описано нижче. При розробці свого додатку я використовував Cocoapods.

4.5.1 Cocoapods

CocoaPods [28]- це дуже потужний інструмент, що має статус *must have* для будь-якого iOS Developer'a. За допомогою нього ми можемо легко, швидко та просто підключати різноманітні утиліти та бібліотеки, які значно полегшують нам життя при розробці програм.

Для використання необхідно в корені проєкту прописати в терміналі `pod init`, після чого з'явиться файл `Podfile` (рис. 4.2). Саме до цього файлу необхідно додавати назви бібліотек. Після того, як `Podfile` заповнений – у терміналі потрібно виконати `pod install`. Після цього Cocoapods завантажить потрібні бібліотеки та створить файл `workspace`, в якому будуть об'єднані файли проєкту та бібліотек.

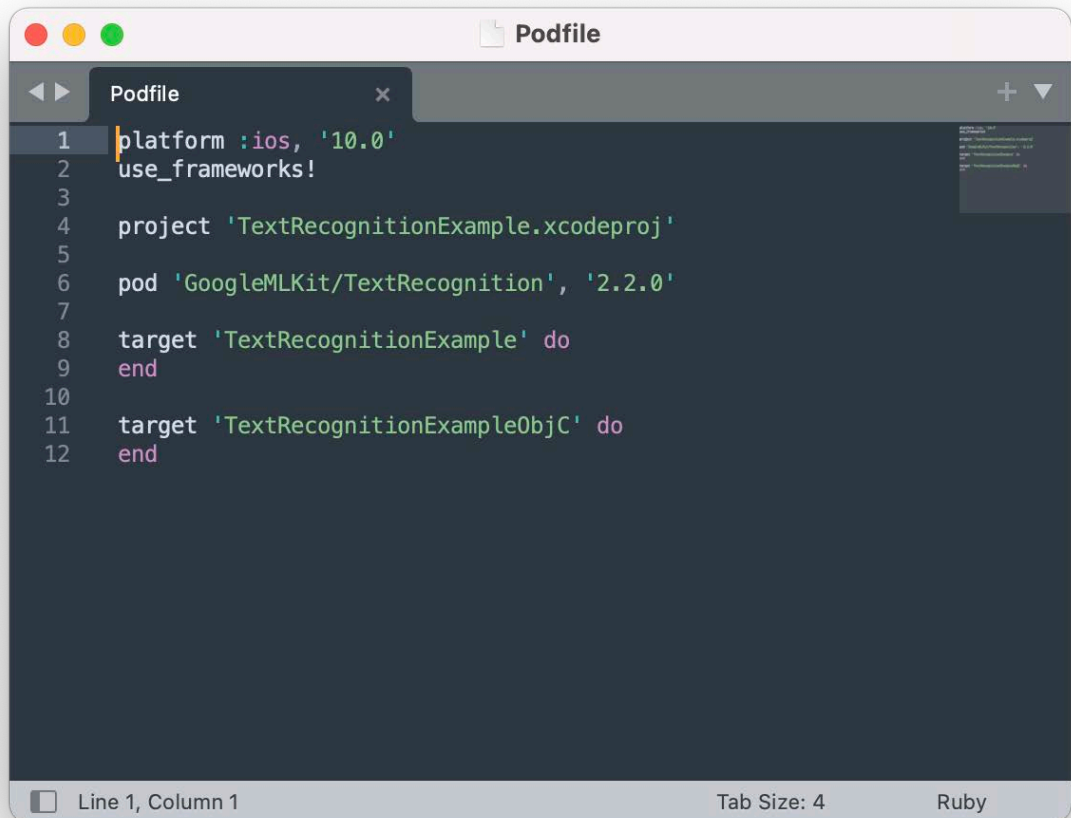


Рисунок 4.2 Podfile

4.5.2 Carthage та порівняння з CocoaPods

Особливості Carthage:

- Carthage схожий на CocoaPods, керує сторонніми фреймворками та залежностями для користувачів, але не змінює файли проєкту автоматично та не генерує конфігурації;
- Carthage – це децентралізований інструмент управління залежностями. Під час встановлення залежностей вам не потрібно заморочуватися, щоб отримати всі залежності CocoaPods, що економить час;
- це не нав'язливо для проєкту. Carthage також має відносно простий дизайн. Він використовує функції Xcode [22]. Коли розробники створюють залежність, це набагато простіше, ніж CocoaPods;

- залежності, якими управляє Carthage, необхідно скомпілювати лише один раз. Коли проєкт скомпільовано чисто, тобто після очищення робочого простору та видалення кешу файлів, залежності не будуть перекомпільовані, що заощадить час;
- автоматичне програмування сторонніх фреймворків як Dynamic framework (динамічна бібліотека);
- повна інтеграція з CocoaPods, проєкт може мати як CocoaPods, так і Carthage.

Недоліки:

- підтримує лише iOS8 +;
- він підтримує лише фреймворки, тому його не можна використовувати для розробки версій системи до iOS 8;
- підтримувані сторонні фреймворки та залежності, встановлені Carthage, не такі багаті, як CocoaPods;
- неможливо знайти вихідний код у Xcode;
- розмір інсталяційного пакета більший, ніж розмір пакета, встановленого з CocoaPods.

4.5.3 Swift Package Manager

Разом із релізом у open source мови Swift 3 грудня 2015 року Apple представила децентралізований менеджер залежностей Swift Package Manager.

До публічної версії доклали руку відомі Max Howell, автор Homebrew, і Matt Thompson, який написав AFNetworking. SwiftPM покликаний автоматизувати процес встановлення залежностей, а також подальше тестування та складання проєкту мовою Swift на всіх доступних операційних системах, проте поки що його підтримують лише macOS та Linux.

Мінімальні вимоги – Swift 3.0. Щоб відкрити файл проєкту, потрібно Xcode 8.0 або вище. SwiftPM дозволяє працювати з проєктами без `xcodeproj`-файлу, тому Xcode на OS X не є обов'язковим, а на Linux його і так немає [30].

Варто розвіяти сумнів – проєкт ще в активній розробці. Використання UIKit, AppKit та інших фреймворків iOS та OS X SDK як залежностей недоступне, оскільки SwiftPM підключає залежності у вигляді вихідного коду, який потім збирає. Таким чином, використання SwiftPM на iOS, watchOS та tvOS можливе, але тільки з використанням Foundation та залежностей сторонніх бібліотек з відкритого доступу. Один єдиний `import UIKit` робить вашу бібліотеку непридатною для розповсюдження через SwiftPM.

Тепер давайте розберемося з основними компонентами SwiftPM:

- модуль (module) – набір *.swift-файлів, що виконує певне завдання. Один модуль може використовувати функціонал іншого модуля, який підключає як залежність. Проєкт може бути зібраний на основі єдиного модуля. Поділ вихідного коду на модулі дозволяє виділити в окремий модуль функцію, яку можна буде використовувати повторно під час збирання іншого проєкту. Наприклад, модуль запитів мережі або модуль роботи з базою даних. Модуль використовує поріг інкапсуляції рівня `internal` і є бібліотекою (library), яка може бути підключена до проєкту. Модуль може бути підключений як з того самого пакета (представлений у вигляді іншого таргету), так і з іншого пакета (представлений у вигляді іншого продукту);

- продукт (product) – результат складання таргету (target) проєкту. Це може бути бібліотека (library) або файл, що виконується (executable). Продукт включає вихідний код, який відноситься безпосередньо до цього продукту, а також вихідний код модулів, від яких він залежить;

- пакет (package) – набір *.swift-файлів та `manifest`-файлу `Package.swift`, який визначає ім'я пакета та набір вихідних файлів. Пакет містить один або більше модулів [30].

4.6 Установка та запуск робочого місця

Для правильного функціонування проєкту, програмісту треба, на комп'ютері з операційною системою не нижче macOS Big Sur (11.5.2), та встановленою на ньому середою розробки Xcode 12.5.1, запустити файл «.xcworkspace». Оскільки даний файл являє собою робочу область в якій співіснує сам проєкт та бібліотеки, які були інтегровані за допомогою менеджера залежностей Cocoapods, програмісту слід переконатись, що встановлена актуальна версія менеджера залежностей і він готовий до роботи.

Для тестування додатку треба мати пристрій з встановленою на ньому операційною системою IOS, версія якої не повинна бути менше 13.0. Також є можливість тестувати розроблену систему за допомогою вбудованого в середу розробки Xcode симуляторі.

4.7 Особливості побудови дизайну

Для побудови та верстки дизайну було обрано бібліотеку SnapKit [21]. Зручність бібліотеки полягає в тому, що верстка відбувається через код, а не через Storyboard. Дане рішення було прийнято у зв'язку з тим, що коли змінюєш дизайн через Storyboard – не дуже зручно відстежувати зміни, оскільки вони перетворюються на xml файл з не зовсім зрозумілим синтаксисом.

Ця бібліотека легка у використанні та зручна для масштабування у майбутньому, коли над одним проєктом працюватиме багато розробників. Завдяки цьому буде мало git-conflicts.

Різницю між Storyboard .xml (рис 4.3) та SnapKit(рис. 4.4) можна побачити на рисунках нижче

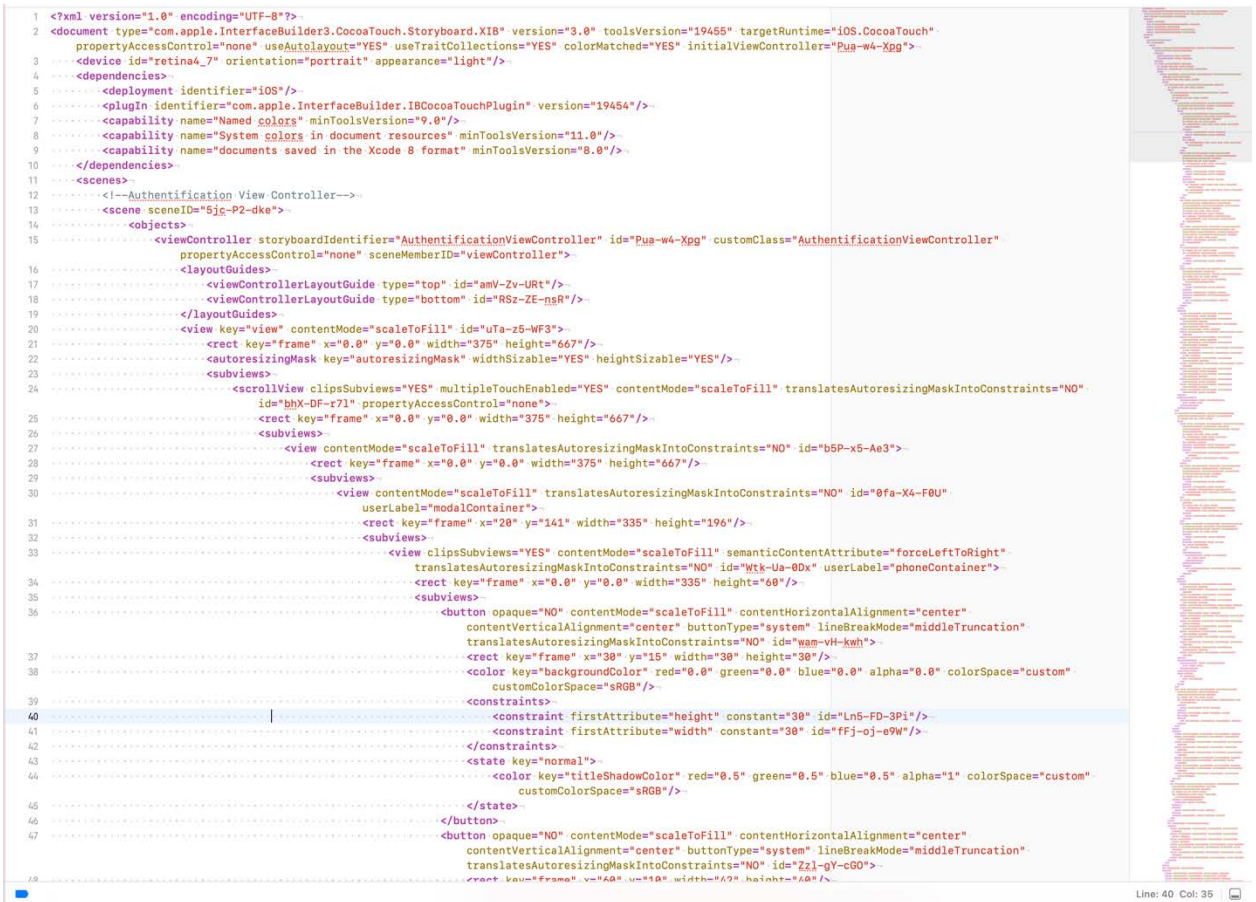


Рисунок 4.3 – Приклад побудови дизайну через Storyboard

```

56
57 private func setupConstraints() {
58     layoutGuide.snp.makeConstraints { make in
59         make.edges.equalToSuperview()
60     }
61
62     titleLabel.snp.makeConstraints {
63         $0.leading.trailing.equalToSuperview().inset(20)
64         $0.top.equalToSuperview().inset(20)
65     }
66
67     stackView.snp.makeConstraints {
68         $0.leading.trailing.equalTo(titleLabel)
69         $0.top.equalTo(titleLabel.snp.bottom).offset(20)
70     }
71
72     dismissButton.snp.makeConstraints {
73         $0.top.equalTo(stackView.snp.bottom).offset(20)
74         $0.leading.trailing.equalTo(titleLabel)
75         $0.bottom.equalTo(safeAreaLayoutGuide).inset(13)
76     }
77 }

```

Рисунок 4.4 – Приклад побудови дизайну через SnapKit

Як можна побачити – синтаксис SnapKit не навантажений і цілком зрозумілий.

4.8 Блок схеми користуванням роботи додатку

Додаток має три логічні розділи.

- сканування по фотографії з галереї (рис 4.6) та головна сторінка водночас, яка відкривається при запуску додатку (рис 4.5);
- сканування шляхом створення фотографії з програми (рис 4.7);
- сканування у реальному часі(рис 4.8).

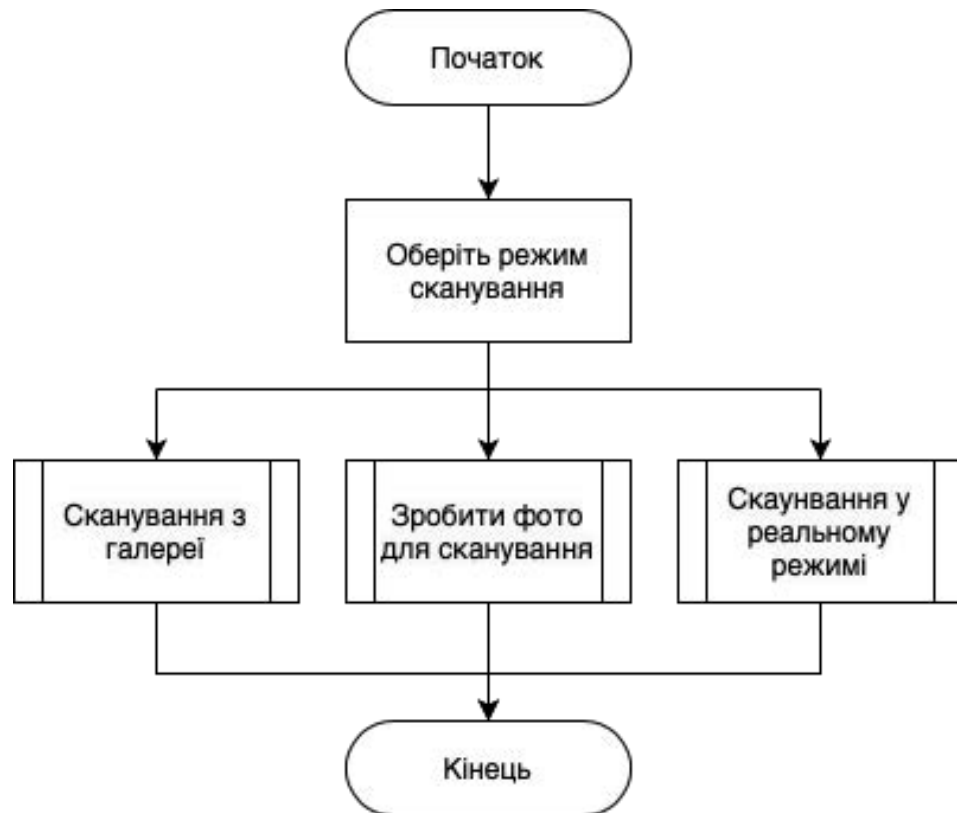


Рисунок 4.5 – Блок схема роботи головної сторінки

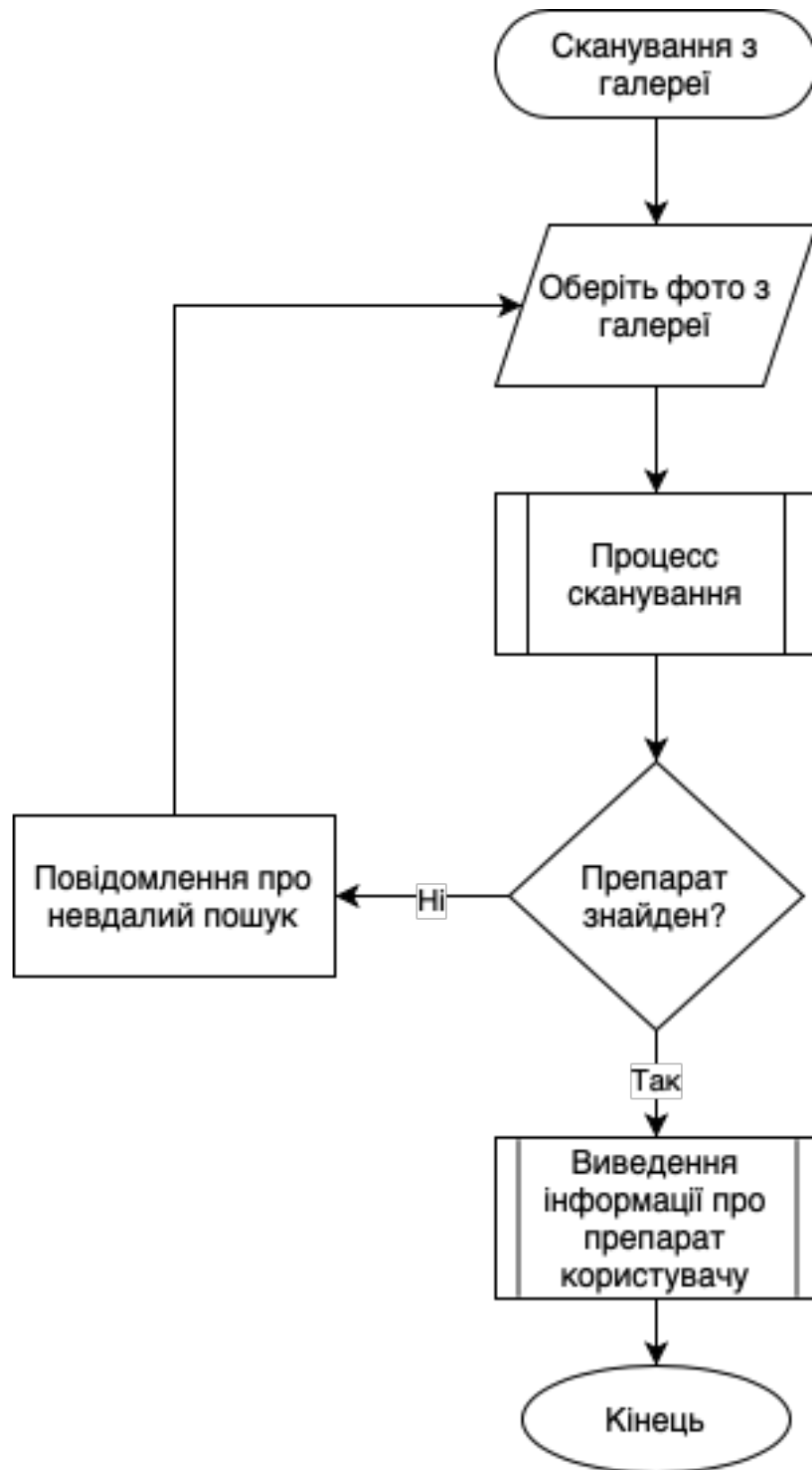


Рисунок 4.6 – Алгоритм роботи сканування фото з галереї

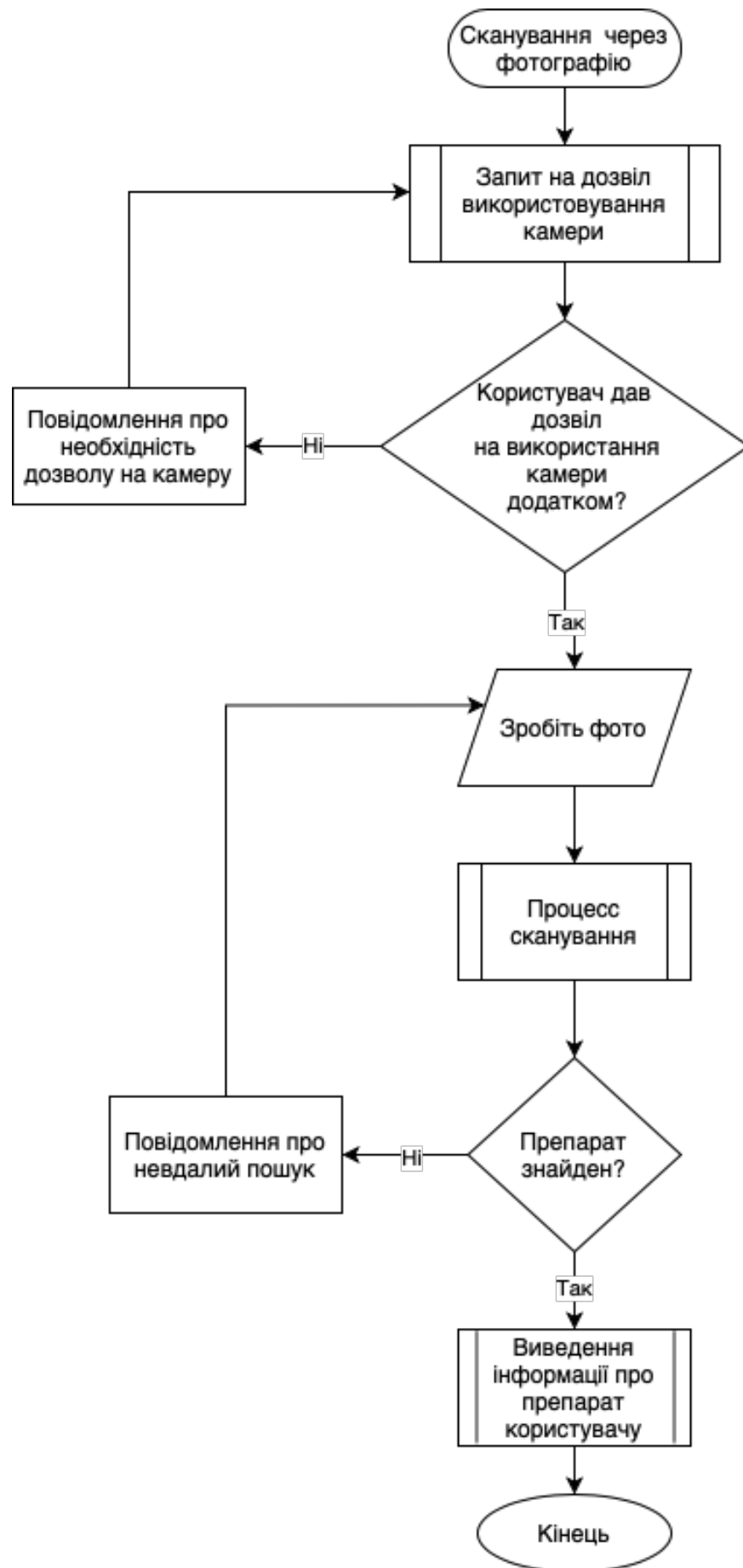


Рисунок 4.7 – Алгоритм роботи сканування за фотографією з камери додатку

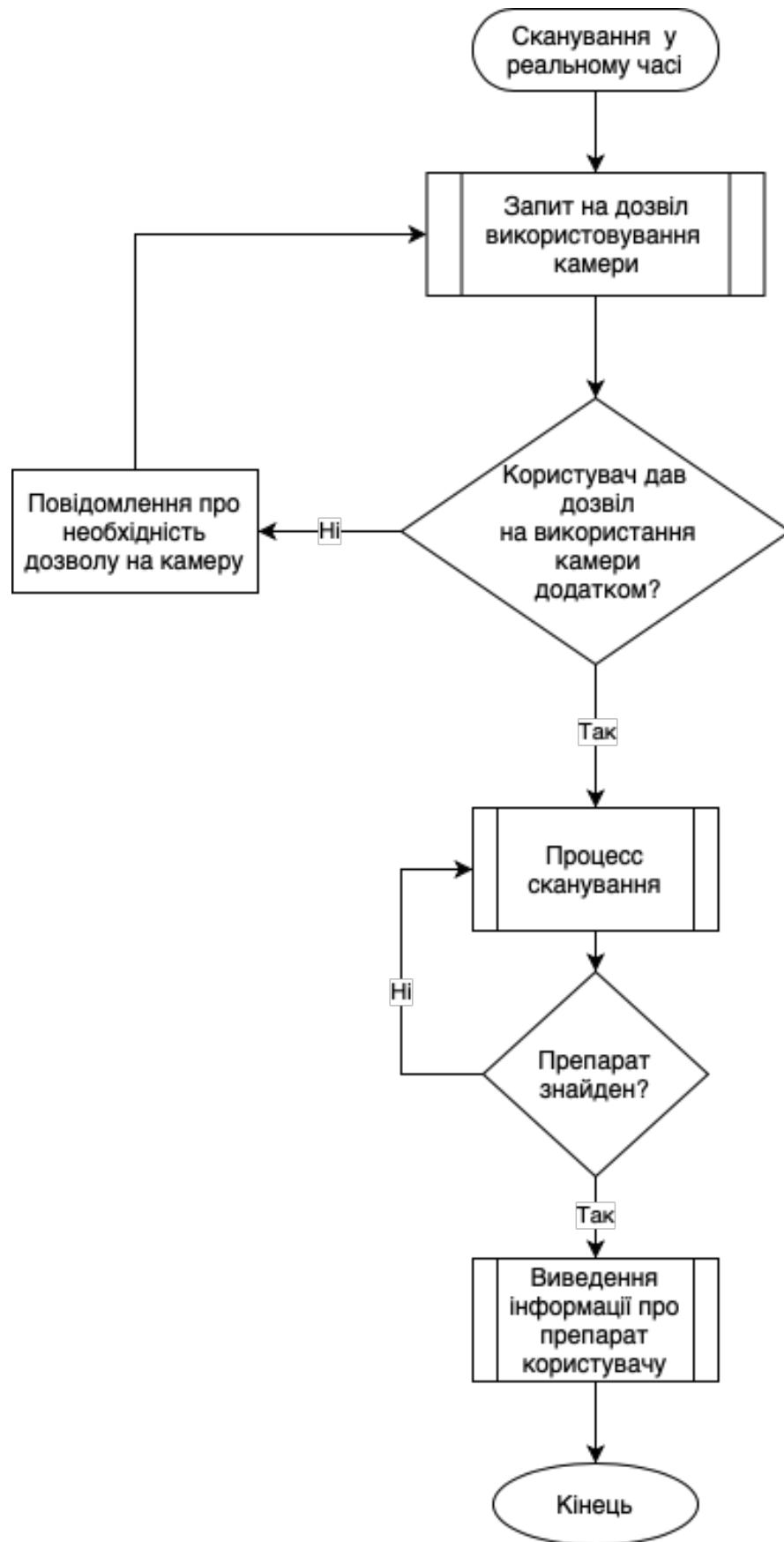


Рисунок 4.8 – Алгоритм сканування у реальному часі

4.8.1 Процес сканування

Процес сканування відбувається у кілька етапів. Спочатку сканується поверхня на об'єкт наявності моделі упаковки препарату, потім відбувається скінювання тексту та пошук слів, після чого йде звернення до АПІ зі знайденими словами.

Необхідність у скануванні моделі упаковки полягає в тому, що без цього кроку абсолютно всі знайдені слова надсилатимуть у запиті, що негативно позначиться на швидкості видачі необхідного результату та з кожною секундою уповільнюватиметься пошук.

Алгоритм процесу скінарування можна спостерігати на рисунку 4.9.

Як можна побачити – є обмеження у 10 спроб на пошук слів. Це було зроблено для того, щоб програма не зависала на етапі сканування тексту і у разі проблем з якістю сканованої поверхні - алгоритм запускався заново

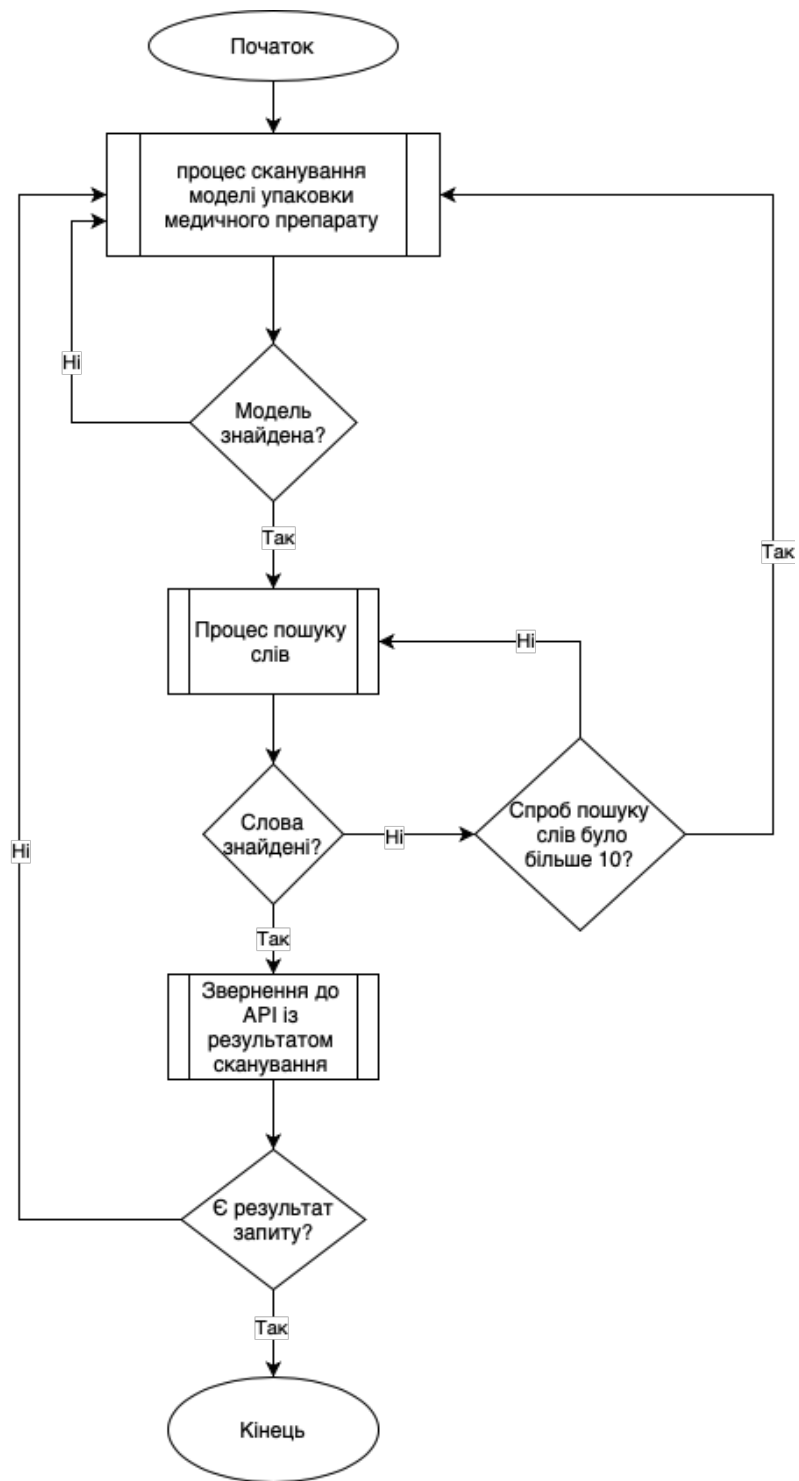


Рисунок 4.9 – Алгоритм процесу сканування

4.9 Діаграми класів

На рисунку 4.10 зображена діаграма класів та всі атрибути.

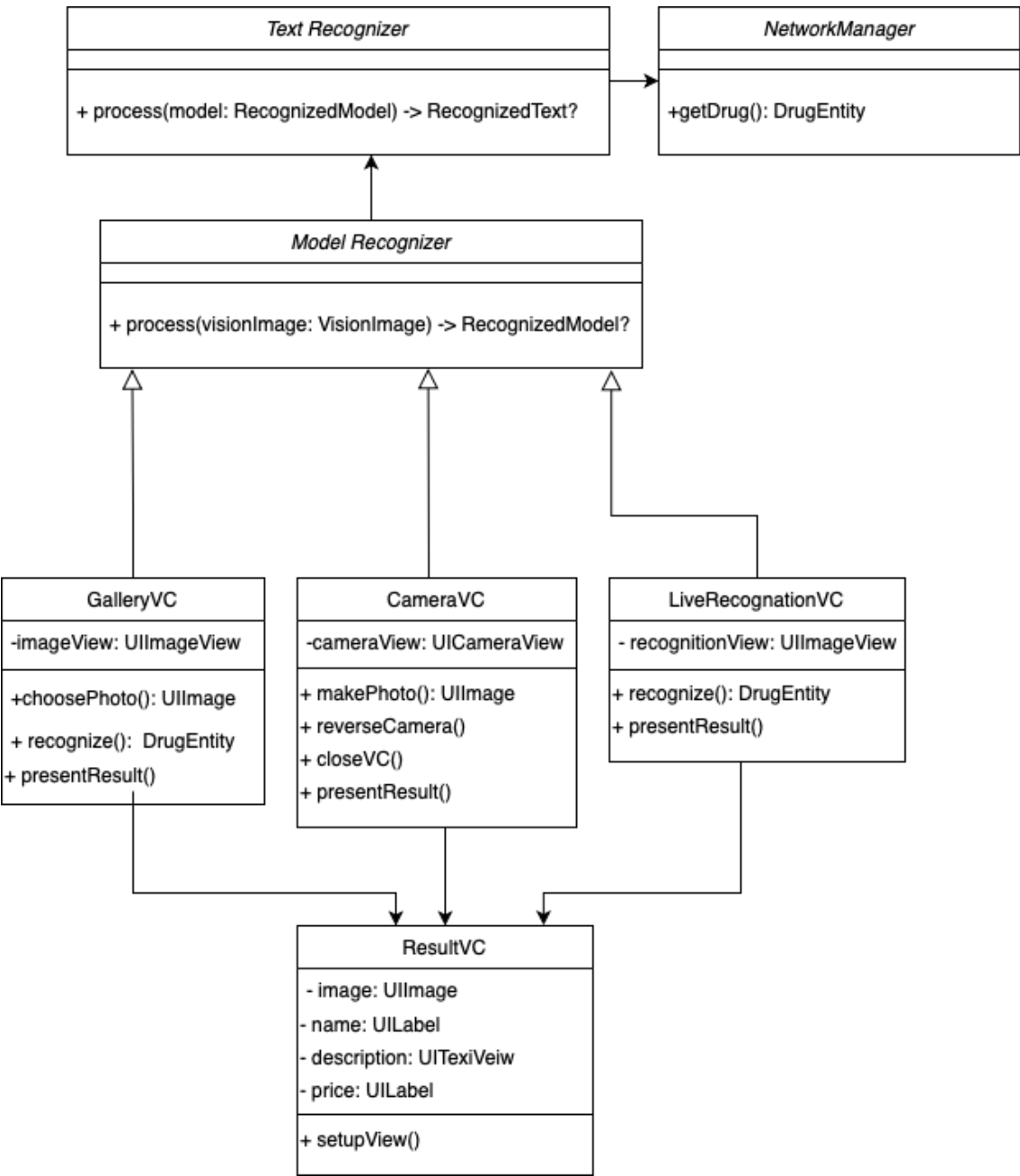


Рисунок 4.10 – UML діаграма класів

Таблиця 4.1 – Клас GalleryVC

Властивості		
Назва	Тип	Опис
imageView	UIImageView *	Поле, де буде зображена картинка, яку обрав користувач

Продовження таблиці 4.1

Методи	
Назва	Опис
+ choosePhoto() UIImage	Метод, який відкриває стандартне вікно галереї.
+ recognize(): DrugEntity	Метод, який викликається після того, як користувач обрав картинку з галереї. Повертає метод сутність DrugEntity, у якій знаходиться вся інформація про препарат. Сутність є опціональною, та не обов'язково повинно мати в собі даних, якщо результату пошуку немає.
+ presentResult()	Відкриває екран з результатом пошук, якщо результат пошуку був успішний.

Таблиця 4.2 – Клас CameraVC

Властивості		
Назва	Тип	Опис
cameraView	UICameraView *	Поле, де буде зображена картинка з камери
Методи		
Назва	Опис	
+ makePhoto() UIImage	Метод, який робить знімок.	
+ reverseCamera()	Переключається з фронтальної на задню камеру.	
+ presentResult()	Відкриває екран з результатом пошук, якщо результат пошуку був успішний.	
+ closeVC()	Закриває поточний екран.	

Таблиця 4.3 – Клас LiveRecognitionVC

Властивості		
Назва	Тип	Опис
recognitionView	UIImageView *	Поле, де буде зображена картинка з камери
Методи		
Назва	Опис	
+ recognize(): DrugEntity	Метод, який звертається до Model Recognition та повертає сутність з бази даних за результатами пошуку.	
+ presentResult()	Відкриває екран з результатом пошук, якщо результат пошуку був успішний.	

Таблиця 4.4 – Клас ResultVC

Властивості		
Назва	Тип	Опис
image	UIImage *	Поле, де буде зображена картинка препарату.
name	UILabel *	Назва препарату.
description	UITextView *	Поле для опису препарату.
price	UILabel *	Ціна.
Методи		
Назва	Опис	
+ setupView()	Метод, який встановлює елементи дизайну на екрані.	

Таблиця 4.5 – Клас ModelRecognizer

Методи	
Назва	Опис
+ process(visionImage: VisionImage): RecognizedModel	Метод який запускає процес сканування та пошуку моделі упаковки препарату. Метод приймає картинку, повертає знайдену модель препарату для подальшого пошуку

Таблиця 4.6 – Клас TextRecognizer

Методи	
Назва	Опис
+ process(model: RecognizedModel): RecognizedText	Метод який запускає процес пошуку тексту на відсканованій моделі. Метод приймає модель, повертає знайдені слова

Таблиця 4.7 – Клас NetworkManager

Методи	
Назва	Опис
+ getDrug(): DrugEntity	Метод який звертається із знайденими на моделі словами до API та повертає результат у вигляді сутності з бази даних.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

У цьому розділі будуть показані основні екрани застосунку і опис їх функцій.

5.1 Головний екран

На рисунку нижче (рис 5.1) показано знімок екрана, який завантажується першим при відкритті програми.

Для зручності, щоб не завантажувати дизайн зайвими кнопками для переходу в розділ - головний екран поєднаний з екраном сканування фотографій з бібліотеки.

Також на цьому екрані можна помітити три основні кнопки нагорі:

- перша веде на відкриття галереї;
- друга відкриває камеру телефону для того, щоб можна було зробити фотографію;
- третя кнопка запускає режим сканування у реальному часі.



Рисунок 5.1 – Головний екран з елементами навігації до інших розділів

5.2 Режим фотографії

На екран фотографії (рис 5.2) користувач потрапляє коли натискає другу кнопку у верхньому полі навігації.

На цьому екрані користувач може зробити якісне фото без втрати якості, увімкнути спалах, якщо погане освітлення та налаштувати фокус для покращення якості.



Рисунок 5.2 – Екран фотографії

5.3 Режим сканування у реальному часі

Натиснувши на третю кнопку в меню навігації, відкривається новий екран, на якому запускається режим сканування в реальному часі (рис 5.3). Сканування відбуватиметься доти, доки не знайдеться препарат у базах даних.

На цьому екрані немає можливості включити спалах, однак налаштований автофокус і камера постійно намагатиметься сфокусуватися на об'єктах перед собою.

Тому для максимально швидкого результату цей режим необхідно використовувати при хорошому освітленні та тримати камеру перед об'єктом сканування.



Рисунок 5.3 – Режим сканування у реальному часі

Тут можна побачити як відпрацьовує алгоритм сканування та виділяє на упаковці слова червоним.

5.4 Результат сканування

Після того, як препарат буде знайдений - користувачеві відобразиться результат (рис 5.4) у вигляді спливаючого вікна, яке можна легко закрити.

Коли вікно з результатом відкрито - сканування припиняється. Як тільки вікно буде закрито свайпом донизу - сканування продовжиться.



Рисунок 5.4 – Результат сканування

ВИСНОВКИ

В результаті виконання магістерської роботи було створено та програмно реалізовано систему сканування та розпізнавання медичних препаратів з подальшим виводом інформації про них.

Для досягнення мети роботи було поставлено наступні задачі:

- проаналізовано існуючі аналоги;
- виявлено переваги та проблеми існуючих систем;
- сформовано вимоги для системи, що розроблялась;
- обран алгоритм розпізнавання, який працює на основі машинного навчання;
- розроблено додаток для операційної системи IOS 13.0;
- оформлено керівництво користувача і програміста;
- проведено тестування системи.

Наукова новизна полягає в тому, що для досягнення мети були використані найскладніші алгоритми машинного навчання, які дозволяють за лічені секунди обробляти дуже велику кількість інформації. Було модифіковано алгоритм розпізнавання шляхом комбінування двох алгоритмів. Завдяки чому вдалося скоротити час розпізнавання на 36.56%. Усі етапи покращення були зафіксовані у вигляді скріншотів та відображені у порівняльних таблицях

Також було закладено розширювану архітектуру програми, яка дозволить розвивати систему в майбутньому і без безлічі переробок додавати новий функціонал у додаток

Практичне значення отриманих результатів полягає в тому, що на основі розробленої моделі можна розвивати програму абсолютно в будь-який бік, а не тільки на об'єкти медицини. Для цього потрібно лише змінити пошукову модель даних з упаковок медикаментів на будь-яку іншу та обрати потрібний API сервіс для пошуку матеріалу.

Всі завдання, які були поставлені для виконання магістерської роботи, були повністю виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1.Что такое «фуфломицины» [Электрон. ресурс]. – Режим доступа: <https://www.ridus.ru/news/336314>.
- 2.Все про Плацебо [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Плацебо>.
- 3.Ліки контроль [Електрон. ресурс]. – Режим доступу: <https://likicontrol.com.ua/завантажити-ліки-контроль/>.
- 4.Честный знак [Электрон. ресурс]. – Режим доступа: <https://честныйзнак.рф>.
- 5.Что такое UI/UX [Электрон. ресурс]. – Режим доступа: https://skillbox.ru/media/design/ux_ui_dizayn_chno_eto_takoe/.
- 6.Swift versus Objective-C [Electronic resource]. – Access mode: <https://www.ideamotive.co/blog/swift-vs-objective-c-which-should-you-pick-for-your-next-ios-mobile-app>.
- 7.React Native [Electronic resource]. – Access mode: <https://reactnative.dev>.
- 8.JetBrains [Electronic resource]. – Access mode: <https://www.jetbrains.com>.
- 9.Java [Electronic resource]. – Access mode: <https://www.java.com/ru/download/manual.jsp>.
10. Ruby [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Ruby>.
11. Python [Electronic resource]. – Access mode: <https://www.python.org>.
12. Apple Script [Electronic resource]. – Access mode: <https://ru.wikipedia.org/wiki/AppleScript>.
13. Coordinator Swift [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/company/umbrellaitcom/blog/426507/>.
14. Realm [Electronic resource]. – Access mode: <https://realm.io>.
15. Boltzmann machine [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Boltzmann_machine.

16. Autocoder [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Автокодировщик>.
17. CNN или Сверточная нейронная сеть [Электрон. ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть.
18. Fully connected neural network [Electronic resource]. – Access mode: <https://radiopaedia.org/articles/fully-connected-neural-network>.
19. Сравнение библиотек глубокого обучения на примере задачи классификации рукописных цифр [Электрон. ресурс]. – Режим доступа: <https://www.pvsm.ru/algorithmy/88240>.
20. Unit Testing [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Unit_testing.
21. SnapKit [Electronic resource]. – Access mode: <https://snapkit.io>.
22. Начало работы с Xcode [Электрон. ресурс]. – Режим доступа: <https://riptutorial.com/ru/xcode>.
23. Что такое Xcode [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Xcode>.
24. Official page Apple Developer. [Electronic resource]. – Access mode: <https://developer.apple.com/xcode/>.
25. Developer Documentation [Electronic resource]. – Access mode: <https://developer.apple.com/documentation/>.
26. Что такое MVC Xcode [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Model-View-Controller>.
27. Основные принципы MVC Xcode [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/post/181772/>.
28. CocoaPods [Electronic resource]. – Access mode: <https://cocoapods.org>.
29. Что такое дополненная реальность или AR [Электрон. ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Дополненная_реальность.
30. Swift documentation [Electronic resource]. – Access mode: <https://www.apple.com/ru/swift/>.

31. Все про мову програмування Swift [Електрон. ресурс]. – Режим доступу: [https://uk.wikipedia.org/wiki/Swift_\(мова_програмування\)](https://uk.wikipedia.org/wiki/Swift_(мова_програмування)).
32. MVP vs MVVM in iOS Development. [Electronic resource]. – Access mode: <https://medium.com/@ahmedragabissa/mvp-vs-mvvm-in-ios-development-450d354300df>.
33. Перцептрон [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Перцептрон>.
34. Conjugate gradient method. [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Conjugate_gradient_method.
35. CoreML основы [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/company/otus/blog/513186/>.

Додаток А
Текст програми

A1. ViewController.swift

```
import UIKit

extension UITableView {

    // Universal method for registration UITableViewCell

    final func register<T: UITableViewCell>(cellType: T.Type)

        where T: ReuseIdentifying {

            self.register(UINib.init(nibName: T.reuseIdentifier, bundle: Bundle.main),

                forCellReuseIdentifier: T.reuseIdentifier)

        }

    // Retrieving cell reuseIdentifier

    final func dequeueReusableCell<T: UITableViewCell>(for indexPath: IndexPath, cellType:
T.Type = T.self) -> T

        where T: ReuseIdentifying {

            guard let cell = self.dequeueReusableCell(withIdentifier: cellType.reuseIdentifier, for:
indexPath) as? T else {

                fatalError(

                    "Failed to dequeue a cell with identifier \(cellType.reuseIdentifier) matching type
\(cellType.self). "

                )

            }

            return cell

        }

    }

}
```

A2. AppDelegate.swift

```
import UIKit

@UIApplicationMain
```

```

class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        // Override point for customization after application launch.

        setDefaults()

        let tabBarController = UINavigationController.instantiateFrom(appStoryboard: .main)

        self.window = UIWindow.init(frame: UIScreen.main.bounds)

        self.window?.rootViewController = tabBarController

        self.window?.makeKeyAndVisible()

        return true

    }

extension String {

    static let emptyLine = ""

    static let space = " "

    static let comma = ","

    static let percent = "%"

    static let colon = ":"

    static let celsius = "°C"

    static let meter = "m/s"

}

extension UIAlertController {

    static func universalAlert(title: String, message: String, actions: [(title: String, action: () -> Void)]?) ->
UIAlertController {

        let alert = UIAlertController(title: title, message: message, preferredStyle: .alert)

        if let actions = actions {

```

```

    for action in actions {

        let alertAction = UIAlertAction(title: action.title, style: .default) { _ in

            action.action()

        }

        alert.addAction(alertAction)

    }

    } else {

        let okAction = UIAlertAction(title: "OK", style: .cancel, handler: nil)

        alert.addAction(okAction)

    }

    return alert

}

public enum AlertAction {

    case cancel

    case confirm

}

// NibView Configurator

class NibView: UIView, Nibable {

    override init(frame: CGRect) {

        super.init(frame: frame)

        setupNib()

    }

    required init?(coder aDecoder: NSCoder) {

        super.init(coder: aDecoder)

        setupNib()

    }

```

```

    }

    private func setupNib() {

        guard let view = loadNib() else { return }

        addSubview(view)

        view.frame = bounds

        view.autoresizingMask = [.flexibleWidth, .flexibleHeight]

        setupUI()

    }

    private func loadNib() -> UIView? {

        guard let nib = Self.nib else { return nil }

        return nib.instantiate(withOwner: self, options: nil).first as? UIView

    }

    func setupUI() {}

}

protocol Nibable: ReuseIdentifying {

    static var nib: UINib? { get }

}

extension Nibable {

    static var nib: UINib? {

        let identifier = self.reuseIdentifier

        guard Bundle.main.path(forResource: identifier, ofType: "nib") != nil else {

            return nil

        }

        return UINib(nibName: identifier, bundle: nil)

    }

}

```

```

}

protocol ReuseIdentifying {

    static var reuseIdentifier: String { get }

}

extension ReuseIdentifying {

    static var reuseIdentifier: String {

        return String(describing: Self.self)

    }

}

```

A3. Lifecycle.swift

```

import Foundation

protocol LifecyclePresenter {

    func viewWillAppear()

    func viewDidLoad()

    func viewWillDisappear()

    func viewDidAppear()

    func viewDidDisappear()

}

extension LifecyclePresenter {

    func viewWillAppear() { }

    func viewDidLoad() { }

    func viewWillDisappear() { }

    func viewDidAppear() { }

    func viewDidDisappear() { }

```

```
}
```

A4. Utilites.swift

```
public protocol CommonRouter: AnyObject {

    var viewController: UIViewController? { get set }

    func present(viewController: UIViewController, animated: Bool)

    func push(_ viewController: UIViewController, animated: Bool)

    func dismiss(animated: Bool)

    func popToRoot(animated: Bool)

    func popViewController(animated: Bool)

    func presentAlert(with alert: UIAlertController)

}

extension CommonRouter {

    func push(_ viewController: UIViewController) {

        push(viewController, animated: true)

    }

}

open class CommonRouterImplementation: CommonRouter {

    public weak var viewController: UIViewController?

    public init(viewController: UIViewController) {

        self.viewController = viewController

    }

    var navController: UINavigationController? {

        return viewController?.navigationController

    }

}
```

```

public func push(_ viewController: UIViewController, animated: Bool = true) {

    DispatchQueue.main.async {

        self.navigationController?.pushViewController(viewController, animated: animated)

    }

}

public func present(viewController: UIViewController, animated: Bool = true) {

    DispatchQueue.main.async {

        self.viewController?.present(viewController, animated: animated)

    }

}

public func dismiss(animated: Bool) {

    viewController?.dismiss(animated: animated)

}

public func popToRoot(animated: Bool) {

    navigationController?.popToRootViewController(animated: animated)

}

public func popViewController(animated: Bool) {

    navigationController?.popViewController(animated: animated)

}

public func showAlert(with alert: UIAlertController) {

    DispatchQueue.main.async {

        self.viewController?.present(alert, animated: true)

    }

}

}

```

```

}

/// Converts an image buffer to a `UIImage`.

///

/// @param imageBuffer The image buffer which should be converted.

/// @param orientation The orientation already applied to the image.

/// @return A new `UIImage` instance.

public static func createUIImage(

    from imageBuffer: CVImageBuffer,

    orientation: UIImage.Orientation

) -> UIImage? {

    let ciImage = CIImage(cvPixelBuffer: imageBuffer)

    let context = CIContext(options: nil)

    guard let cgImage = context.createCGImage(ciImage, from: ciImage.extent) else { return nil }

    return UIImage(cgImage: cgImage, scale: Constants.originalScale, orientation: orientation)

}

/// Converts a `UIImage` to an image buffer.

///

/// @param image The `UIImage` which should be converted.

/// @return The image buffer. Callers own the returned buffer and are responsible for releasing it

/// when it is no longer needed. Additionally, the image orientation will not be accounted for

/// in the returned buffer, so callers must keep track of the orientation separately.

```

A5. SharedService.swift

```

import Foundation

import CoreData

import UIKit

```

```

class Service {

    static let shared = Service()

    private let _Manager = _Manager()

    private let coreDataService = CoreDataServie()

    var coordinates: (lat: Double, lng: Double) {

        didSet {

            UserDefaults.standard.set(coordinates.lat, forKey: "latitude")

            UserDefaults.standard.set(coordinates.lng, forKey: "longitude")

            UserDefaults.standard.synchronize()

            NotificationCenter.default.post(name: Notification.Name("Update_"), object: nil)

        }

    }

    private init() {

        coordinates = (lat: UserDefaults.standard.double(forKey: "latitude"),

            lng: UserDefaults.standard.double(forKey: "longitude"))

    }

    func update_(type: _Type = .hourly, completion: @escaping ([_Entity]?) -> Void) {

        Manager.get_(lat: coordinates.lat, lng: coordinates.lng) { [weak self] result in

            switch result {

            case .success(let value):

                UserDefaults.standard.set(value.properties?.meta?.updatedAt?.formatted, forKey: "updatedAt")

                UserDefaults.standard.synchronize()

                self?.save_ToDB(with: value, type: type, completion: completion)

            case .failure(let error):

                switch type {

```

```

        case .hourly:

            self?.getHourly_(completion: completion)

        case .daily:

            self?.getdaily_(completion: completion)

        }

        print(error.localizedDescription)

    }

}

}

func save_ToDB(with model: _Model, type: _Type, completion: @escaping ([_Entity]?) -> Void) {

    removeAllData()

    guard let context = CoreDataService.configureContext() else { return }

    guard let entity = NSEntityDescription.entity(forEntityName: "_Entity", in: context) else { return }

    model.properties?.timeseries?.forEach { info in

        let _Object = _Entity(entity: entity, insertInto: context)

        _Object.date = info.time

        if let humidity = info.data?.instant?.details?.relativeHumidity {

            _Object.humidity = humidity

        }

        _Object.next_12_hour_icon = info.data?.next12_Hours?.summary?.symbolCode

        _Object.next_1_hour_icon = info.data?.next1_Hours?.summary?.symbolCode

        _Object.next_6_hour_icon = info.data?.next6_Hours?.summary?.symbolCode

        if let temperature = info.data?.instant?.details?.airTemperature {

            _Object.temperature = temperature

        }

    }

}

```

```

if let temperatureMax = info.data?.next6_Hours?.details?.airTemperatureMax {
    _Object.temperature_max = temperatureMax
}

if let temperatureMin = info.data?.next6_Hours?.details?.airTemperatureMin {
    _Object.temperature_max = temperatureMin
}

if let windSpeed = info.data?.instant?.details?.windSpeed {
    _Object.wind_speed = windSpeed
}
}

if coreDataService.saveContext(with: context) {
    switch type {
    case .hourly:
        getHourly_(completion: completion)
    case .daily:
        getdaily_(completion: completion)
    }
} else {
    print("SAVE ERROR")
}
}

func preloadData(type: _Type = .hourly, completion: @escaping ([_Entity]?) -> Void) {
    switch type {
    case .hourly:
        getHourly_(completion: completion)

```

```

case .daily:

    getdaily_(completion: completion)

}

}

func getHourly_(completion: @escaping ([_Entity]?) -> Void) {

    guard let context = coreDataService.configureContext() else { return }

    let fetchRequest: NSFetchedRequest<_Entity> = _Entity.fetchRequest()

    fetchRequest.returnsObjectsAsFaults = false

    let fromDate = Date() as NSDate

    let toDate = Date.tomorrow as NSDate

    fetchRequest.predicate = NSPredicate(format: "(date >= %@) AND (date <= %@)", fromDate,
toDate)

    fetchRequest.sortDescriptors = [NSSortDescriptor(key: "date", ascending: true)]

    do {

        let objects = try context.fetch(fetchRequest)

        completion(objects)

        print("Successfully fetched +++")

    } catch let error as NSError {

        completion(nil)

        NSLog("FETCH ERROR", error.localizedDescription)

    }

}

func getdaily_(completion: @escaping ([_Entity]?) -> Void) {

    guard let context = coreDataService.configureContext() else { return }

    let fetchRequest: NSFetchedRequest<_Entity> = _Entity.fetchRequest()

    fetchRequest.returnsObjectsAsFaults = false

```

```

let fromDate = Date.tomorrow as NSDate

fetchRequest.predicate = NSPredicate(format: "date >= %@", fromDate)

fetchRequest.sortDescriptors = [NSSortDescriptor(key: "date", ascending: true)]

do {

    let objects = try context.fetch(fetchRequest)

    var filteredObjects: [_Entity] = []

    objects.forEach { item in

        if !filteredObjects.contains(where: { $0.date?.withoutTime == item.date?.withoutTime }) {

            filteredObjects.append(item)

        }

    }

    completion(filteredObjects)

    print("Successfully fetched +++")

} catch let error as NSError {

    completion(nil)

    NSLog("FETCH ERROR", error.localizedDescription)

}

}

func getAll_(completion: @escaping ([_Entity]?) -> Void) {

    guard let context = coreDataService.configureContext() else { return }

    let fetchRequest: NSFetchedRequest<_Entity> = _Entity.fetchRequest()

    do {

        let objects = try context.fetch(fetchRequest)

        completion(objects)

        print("Successfully fetched +++")

    }

}

```

```

    } catch let error as NSError {

        completion(nil)

        NSLog("FETCH ERROR", error.localizedDescription)

    }

}

func getCurrent_() -> _Entity? {

    guard let context = coreDataService.configureContext() else { return nil }

    let fetchRequest: NSFetchedRequest<_Entity> = _Entity.fetchRequest()

    fetchRequest.returnsObjectsAsFaults = false

    let current = Date() as NSDate

    fetchRequest.sortDescriptors = [NSSortDescriptor(key: "date", ascending: true)]

    fetchRequest.predicate = NSPredicate(format: "date >= %@", current)

    fetchRequest.fetchLimit = 1

    do {

        let objects = try context.fetch(fetchRequest)

        return objects.first

    } catch {

        return nil

    }

}

func removeAllData() {

    guard let context = coreDataService.configureContext() else { return }

    getAll_ { objects in

        objects?.forEach { object in

```

```

        context.delete(object)

    }

}

coreDataService.saveContext(with: context)

}

}

```

A6. CoreData.m

```

import CoreData

import UIKit

class CoreDataServie {

    // MARK: - Core Data get context

    func configureContext() -> NSManagedObjectContext? {

        guard let appDelegate = UIApplication.shared.delegate as? AppDelegate else { return nil}

        let context = appDelegate.persistentContainer.viewContext

        return context

    }

    // MARK: - Core Data Saving support

    func saveContext(with context: NSManagedObjectContext) -> Bool {

        do {

            try context.save()

            return true

        } catch {

            return false

        }

    }

}

```

```
}
```

A7. PaginatorUtilites.m

```
import Foundation
```

```
class Paginator {
```

```
    typealias AfterNextClosureType = (_ isFirstPage: Bool) -> Void
```

```
    typealias FetchCompletionClosureType = (_ result: Bool) -> Void
```

```
    typealias FetchClosureType = (_ page: Int, _ completion: @escaping FetchCompletionClosureType) -> Void
```

```
    private var currentPage = 0
```

```
    private var isLoading = false
```

```
    private var isFetching = false
```

```
    private var fetchClosure: FetchClosureType
```

```
    private var afterNextClosure: AfterNextClosureType
```

```
    init(fetchClosure: @escaping FetchClosureType, afterNextClosure: @escaping AfterNextClosureType)
```

```
{
```

```
        self.fetchClosure = fetchClosure
```

```
        self.afterNextClosure = afterNextClosure
```

```
}
```

```
func load() {
```

```
    reset()
```

```
    isLoading = true
```

```
    next()
```

```
}
```

```
func next() {
```

```
    guard currentPage < Int.max, !isFetching, isLoading else {
```

```

        return

    }

    isFetching = true

    fetchClosure(currentPage) { [weak self] result in

        guard let strongSelf = self, strongSelf.currentPage != Int.max else {

            return

        }

        strongSelf.isFetching = false

        strongSelf.afterNextClosure(strongSelf.currentPage == 0)

        strongSelf.currentPage = result ? strongSelf.currentPage + 1 : Int.max

    }

}

func reset() {

    currentPage = 0

    isFetching = false

    isLoading = false

}

}

protocol BrowserOpenable {

    func openLink(with url: URL, on viewController: UIViewController?)

}

extension BrowserOpenable {

    func openLink(with url: URL, on viewController: UIViewController?) {

        switch url.isSupportedBySFSafariViewController {

```

```

case true:

    switch url.type {

        case .externalBrowser: openExternalBrowser(with: url)

        case .default: openSafari(with: url, viewController: viewController)

    }

case false:

    openExternalBrowser(with: url)

    }

}

private func openSafari(with url: URL, viewController: UIViewController?) {

    let vc = SFSafariViewController(url: url)

    viewController?.present(vc, animated: true, completion: nil)

}

private func openExternalBrowser(with url: URL) {

    UIApplication.shared.open(url, options: [:], completionHandler: nil)

}

}

public protocol CommonRouter: AnyObject {

    var viewController: UIViewController? { get set }

    func present(viewController: UIViewController, animated: Bool)

    func push(_ viewController: UIViewController, animated: Bool)

    func presentFromPush(viewController: UIViewController)

    func presentAlert(with message: String)

    func presentAlert(with alert: UIAlertController)

    func showToast(with text: String)

```

```

func openLink(with url: URL)

func presentHUD()

func dismissHUD()

func sendLoadedScreenAnalytics()

func showShareActivity(_ cardEntity: CardEntity?,
                      shareUrl: String,
                      type: ShareType)

func dismiss(animated: Bool)

func popToRoot(animated: Bool)

func popViewController(animated: Bool)
}

extension CommonRouter {

    func push(_ viewController: UIViewController) {

        push(viewController, animated: true)

    }

}

open class CommonRouterImplementation: CommonRouter, BrowserOpenable {

    public weak var viewController: UIViewController?

    public init(viewController: UIViewController) {

        self.viewController = viewController

    }

    var navController: UINavigationController? {

        return viewController?.navigationController

    }

    public func push(_ viewController: UIViewController, animated: Bool = true) {

```

```

DispatchQueue.main.async {

    self.navigationController?.pushViewController(viewController, animated: animated)

}

}

public func presentFromPush(viewController: UIViewController) {

    if var topController = UIApplication.shared.keyWindow?.rootViewController {

        while let presentedViewController = topController.presentedViewController {

            topController = presentedViewController

        }

        viewController.modalPresentationStyle = .fullScreen

        DispatchQueue.main.async {

            topController.present(viewController, animated: true)

        }

    }

}

public func present(viewController: UIViewController, animated: Bool = true) {

    DispatchQueue.main.async {

        self.viewController?.present(viewController, animated: animated)

    }

}

public func dismiss(animated: Bool) {

    viewController?.dismiss(animated: animated)

}

public func popToRoot(animated: Bool) {

    navigationController?.popToRootViewController(animated: animated)

```

```

    }

    public func popViewController(animated: Bool) {

        navigationController?.popViewController(animated: animated)

    }

    public func presentHUD() {

        guard let viewController = viewController else {

            return

        }

        let navigationControllerView = viewController.navigationController?.view

        HudService.shared.presentHUD(on: navigationControllerView ?? viewController.view)

    }

    public func dismissHUD() {

        guard let viewController = viewController else {

            return

        }

        DispatchQueue.main.async {

            let navigationControllerView = viewController.navigationController?.view

            HudService.shared.dismissHUD(view: navigationControllerView ?? viewController.view)

        }

    }

    public func presentAlert(with message: String) {

        let alert = AlertService.alertWithMessage(message)

        DispatchQueue.main.async {

            self.viewController?.present(alert, animated: true)

```

```

    }

}

public func showAlert(with alert: UIAlertController) {

    DispatchQueue.main.async {

        self.viewController?.present(alert, animated: true)

    }

}

public func showToast(with text: String) {

    let toastView = ToastService.toastView(with: text)

    DispatchQueue.main.async {

        self.navController?.view.showToast(

            toastView,

            duration: 3,

            position: CSToastPositionCenter,

            completion: nil

        )

    }

}

public func openLink(with url: URL) {

    openLink(with: url, on: viewController)

}

public func sendLoadedScreenAnalytics() {

    AnalyticsContainer

        .shared

```

```

        .analyticsProvider

        .loadedScreen(title: viewController?.navigationItem.title ?? "Empty")
    }

    public func showShareActivity(_ cardEntity: CardEntity?,
                                shareUrl: String,
                                type: ShareType) {

        ShareManager.shared.shareCard(cardEntity,
                                       with: shareUrl,
                                       type: type,
                                       parentController: viewController)
    }
}

protocol AnalyticsUseCase {
    func setAnalytics(_ url: String)
}

class AnalyticsUseCaseImplementation: AnalyticsUseCase {
    func setAnalytics(_ url: String) {
        Crashlytics.crashlytics().setUserID(UserModel.shared.userId ?? .emptyLine)

        let profile = YMMMMutableUserProfile()
        profile.apply(from: [
            YMMProfileAttribute.customString("Company_ID").withValue(url.replacingOccurrences(of:
"https://", with: ""))
        ])

        YMMYandexMetrica.setUserProfileID(UserModel.shared.email)
    }
}

```

```

        YMMYandexMetrica.report(profile, onFailure: { _ in })
    }
}

protocol CellConfiguratorUseCase {

    func setup(presenter: CardsPresenter,

               openCardAdapter: OpenCardAdapter?,

               reloadAdapter: ReloadAdapter?)

    func congifure<T: UITableViewCell>(cell: T?, cellIndex: Int, card: CardEntity)

}

final class CellConfiguratorUseCaseImplementation: CellConfiguratorUseCase {

    weak var openCardAdapter: OpenCardAdapter?

    var reloadAdapter: ReloadAdapter?

    weak var presenter: CardsPresenter?

    func setup(

        presenter: CardsPresenter,

        openCardAdapter: OpenCardAdapter?,

        reloadAdapter: ReloadAdapter?

    ) {

        self.presenter = presenter

        self.openCardAdapter = openCardAdapter

        self.reloadAdapter = reloadAdapter

    }

    // swiftlint:disable function_body_length

    func congifure<T: UITableViewCell>(cell: T?, cellIndex: Int, card: CardEntity) {

        guard let presenter = presenter else { return }

```

```

if let cell = cell as? ErrorTVCell {

    cell.prepareView(title: card.title, feedType: card.type)

} else if let cell = cell as? FeedTimeLineTVCell {

    cell.reloadDelegate = reloadAdapter

    cell.openDelegate = openCardAdapter

    cell.shareCompletion = presenter.shareButtonTap

    cell.taskEntity = presenter.taskEntity(for: cellIndex)

    cell.prepareView(

        cardModel: card,

        cellIndex: cellIndex,

        selectedTag: presenter.selectedTag,

        likeCallback: presenter.likeButtonTap,

        favoriteCallback: presenter.favoriteButtonTap,

        tagsSelectedCallback: presenter.tagSelected,

        moreWasTapped: presenter.moreTappedRows.contains(cellIndex),

        tapOnLink: presenter.tapOnLink

    )

} else if let cell = cell as? FeedVideoTVCell {

    cell.reloadDelegate = reloadAdapter

    cell.openDelegate = openCardAdapter

    cell.shareCompletion = presenter.shareButtonTap

    cell.taskEntity = presenter.taskEntity(for: cellIndex)

    cell.prepareView(

        cardModel: card,

```

```

        cellIndex: cellIndex,

        selectedTag: presenter.selectedTag,

        likeCallback: presenter.likeButtonTap,

        favoriteCallback: presenter.favoriteButtonTap,

        tagsSelectedCallback: presenter.tagSelected,

        moreWasTapped: presenter.moreTappedRows.contains(cellIndex),

        tapOnLink: presenter.tapOnLink
    )

    } else if let cell = cell as? FeedOpenQuizTVCell {

cell.reloadDelegate = reloadAdapter

cell.openDelegate = openCardAdapter

cell.shareCompletion = presenter.shareButtonTap

cell.taskEntity = presenter.taskEntity(for: cellIndex)

cell.prepareView(cardModel: card,

        cellIndex: cellIndex,

        selectedTag: presenter.selectedTag,

        likeCallback: presenter.likeButtonTap,

        favoriteCallback: presenter.favoriteButtonTap,

        tagsSelectedCallback: presenter.tagSelected,

        addCommentButtonAction: presenter.addCommentButtonAction,

        moreWasTapped: presenter.moreTappedRows.contains(cellIndex),

        tapOnLink: presenter.tapOnLink)

    } else if let cell = cell as? FeedQuizTVCell {

cell.reloadDelegate = reloadAdapter

cell.openDelegate = openCardAdapter

```

```

cell.shareCompletion = presenter.shareButtonTap

cell.taskEntity = presenter.taskEntity(for: cellIndex)

cell.prepareView(cardModel: card,
                 cellIndex: cellIndex,
                 selectedTag: presenter.selectedTag,
                 likeCallback: presenter.likeButtonTap,
                 favoriteCallback: presenter.favoriteButtonTap,
                 addCommentButtonAction: presenter.addCommentButtonAction,
                 addMarkButtonAction: presenter.addMarkButtonAction,
                 tagsSelectedCallback: presenter.tagSelected,
                 moreWasTapped: presenter.moreTappedRows.contains(cellIndex),
                 tapOnLink: presenter.tapOnLink)
} else if let cell = cell as? FeedAudioTVCell {

    cell.reloadDelegate = reloadAdapter

    cell.openDelegate = openCardAdapter

    cell.shareCompletion = presenter.shareButtonTap

    cell.isPost = presenter.isPost

    cell.taskEntity = presenter.taskEntity(for: cellIndex)

    cell.taskCompletionDelegate = presenter as? TaskProtocol

    cell.prepareView(cardModel: card,
                     cellIndex: cellIndex,
                     selectedTag: presenter.selectedTag,
                     likeCallback: presenter.likeButtonTap,
                     favoriteCallback: presenter.favoriteButtonTap,
                     tagsSelectedCallback: presenter.tagSelected,

```

```

        moreWasTapped: presenter.moreTappedRows.contains(cellIndex),

        tapOnLink: presenter.tapOnLink)

    } else if let cell = cell as? TrainingSectionTVCell {

        cell.reloadDelegate = reloadAdapter

        cell.openDelegate = openCardAdapter

        cell.shareCompletion = presenter.shareButtonTap

        cell.taskEntity = presenter.taskEntity(for: cellIndex)

        cell.prepareView(

            trainingModel: card as! TrainingEntity,

            cellIndex: cellIndex,

            addFavoritesCallback: presenter.favoriteButtonTap,

            moreWasTapped: presenter.moreTappedRows.contains(cellIndex),

            tapOnLink: presenter.tapOnLink

        )

    } else if let cell = cell as? UnsupportedTVCell {

        cell.prepareView(cardModel: card)

    }

}

// swiftlint:disable function_body_length

}

class OpenCardAdapter: OpenCardProtocol {

    private weak var router: OpenCardRouter?

    private var surveyDelegate: SurveyDelegate?

```

```

private var commentDelegate: CommentDelegate?

private weak var taskCompletion: TaskProtocol?

private var cellIndex: Int?

private var cardEntities: [CardEntity]?

private let openFromPush: Bool

init(router: OpenCardRouter,

    surveyDelegate: SurveyDelegate?,

    commentDelegate: CommentDelegate?,

    taskCompletion: TaskProtocol? = nil,

    cardEntities: [CardEntity]?,

    openFromPush: Bool

) {

    self.router = router

    self.surveyDelegate = surveyDelegate

    self.commentDelegate = commentDelegate

    self.taskCompletion = taskCompletion

    self.cardEntities = cardEntities

    self.openFromPush = openFromPush

}

func updateCardEntities(with cards: [CardEntity]?) {

    self.cardEntities = cards

}

func open(row: Int, taskId: String?, trainingId: String?) {

    guard let card = cardEntities?[row], let typeString = card.type else { return }

    let type = FeedType(string: typeString)

```

```

        cellIndex = row

        open(with: card, type: type, taskId: taskId, trainingId: trainingId)
    }

func open(with card: CardEntity, type: FeedType, taskId: String? = nil, trainingId: String? = nil) {

    let cardEntityModel = CardEntityModel(

        taskId: taskId,

        trainingId: trainingId,

        cardSourceType: .feed,

        cellIndex: cellIndex,

        taskCompletion: taskCompletion,

        commentClosure: commentDelegate,

        surveyDelegate: surveyDelegate,

        openFromPush: openFromPush
    )

    switch type {

    case .timeline:

        router?.presentTimelineScreen(with: card as? TimeLineCardEntity, cardModelProperties:
cardEntityModel)

    case .duel:

        router?.presentDuelScreen(with: card as? DuelEntity, cardModelProperties: cardEntityModel)

    case .photoMark:

        router?.presentPhotoTagsScreen(with: card as? PhotoMarkCardEntity, cardModelProperties:
cardEntityModel)

    case .test:

        router?.presentTestScreen(with: card as? TestCardEntity, cardModelProperties: cardEntityModel)

    case .longread:

```

```

        router?.presentLongReadScreen(with: card as? LongReadCardEntity, cardModelProperties:
cardEntityModel)

        case .kase:

            router?.presentCaseScreen(with: card as? CaseCardEntity, cardModelProperties: cardEntityModel)

        case .survey:

            router?.presentSurveyScreen(with: card as? SurveyEntity, cardModelProperties: cardEntityModel)

        case .photo:

            router?.presentPhotoScreen( with: card as? PhotoCardEntity, cardModelProperties:
cardEntityModel)

        case .openquestion:

            router?.presentOpenQuestionScreen(with: card as? OpenQuestionEntity, cardModelProperties:
cardEntityModel)

        case .audio:

            router?.presentAudioScreen(with: card as? AudioCardEntity, cardModelProperties:
cardEntityModel)

        case .video:

            router?.presentVideoScreen(with: card as? VideoCardEntity, cardModelProperties:
cardEntityModel)

        case .exam:

            router?.presentExamScreen(with: card as? ExamCardEntity, cardModelProperties:
cardEntityModel)

        default:

            router?.presentTrainingScreen(with: card as? TrainingEntity, cardModelProperties:
cardEntityModel)

    }

}

func openCommentScreen(

    feedTypeText: String,

```

```

        trainingId: String?,

        taskId: String?,

        cardModel: CardEntity?
    ) {

        router?.openCommentScreen(feedTypeText: feedTypeText, trainingId: trainingId, taskId: taskId,
cardModel: cardModel)

    }
}

final class ReloadAdapter: RowReloaderProtocol {

    weak var presenter: CardsPresenter?

    init(presenter: CardsPresenter? = nil) {

        self.presenter = presenter
    }

    func updateRow(row: Int) {

        if let index = presenter?.moreTappedRows.firstIndex(of: row) {

            presenter?.moreTappedRows.remove(at: index)

        } else {

            presenter?.moreTappedRows.append(row)

        }

        presenter?.updateTableView()

    }

}

protocol HashPasswordUseCase {

    func hash(password: String) -> (String, Bool)

}

```

```

class HashPasswordUseCaseImplementation: HashPasswordUseCase {

    func hash(password: String) -> (String, Bool) {

        guard let path = Bundle.main.path(forResource: GlobalConstants.Cipher.plistName,
                                           ofType: GlobalConstants.Cipher.plistFormat),

            let data = try? Data(contentsOf: URL(fileURLWithPath: path)),

            let plist = try? PropertyListSerialization.propertyList(from: data, options: .mutableContainers,
                                                                    format: nil),

            let dataFromPlist = plist as? [String: String],

            let key = dataFromPlist[GlobalConstants.Cipher.key],

            let initVector = dataFromPlist[GlobalConstants.Cipher.initVector] else { return (password, false) }

        do {

            let encrypted = try AES(key: key.bytes,
                                    blockMode: CBC(iv: initVector.bytes),
                                    padding: .pkcs5).encrypt(password.bytes)

            let decrypted = try AES(key: key.bytes,
                                    blockMode: CBC(iv: initVector.bytes),
                                    padding: .pkcs5).decrypt(encrypted)

            if password.bytes == decrypted, let encryptedPassword = encrypted.toBase64() {

                return (encryptedPassword, true)

            } else {

                return (password, false)

            }

        } catch {

            return (password, false)

        }
    }
}

```

```

    }

}

enum ValidationType {
    case emptyLine
    case longViolation(long: Int)
}

protocol ValidatorUseCase {
    func validate(textFields: [UITextField?], on type: ValidationType) -> Bool
}

class ValidatorUseCaseImplementation: ValidatorUseCase {
    func validate(textFields: [UITextField?], on type: ValidationType) -> Bool {
        var checker: Bool = true
        switch type {
        case .emptyLine:
            textFields.forEach { textField in
                textField?.validate([EmptyStringCriteria(), SpacesCriteria()]) { result in
                    switch result {
                    case .valid: break
                    case .notValid:
                        checker = false
                    }
                }
            }
        case .longViolation(let long):
            textFields.forEach { textField in
                textField?.validate([PasswordLongCriteria(long: long)]) { result in
                    switch result {
                    case .valid: break
                    case .notValid:
                        checker = false
                    }
                }
            }
        }
    }
}

```

```

        return checker
    }
}

class RepeatedTimer {

    typealias TimerCallback = (TimeInterval) -> Void

    var timer: Timer?

    var timerCallback: TimerCallback

    var startTime: TimeInterval?

    init(timerCallback: @escaping TimerCallback) {

        self.timerCallback = timerCallback

    }

    func startTimer() {

        startTime = Date.timeIntervalSinceReferenceDate

        timer = Timer.scheduledTimer(timeInterval: 0.1, target: self,

                                     selector:
#selector(RepeatedTimer.timerTick), userInfo: nil, repeats: true)

    }

    func stopTimer() {

        timer?.invalidate()

    }

    @objc func timerTick() {

        timerCallback(Date.timeIntervalSinceReferenceDate - startTime!)

    }

}

```

```

extension UITableView {

    var dataHasChanged: Bool {

        guard let dataSource = dataSource else { return false }

        let sections = dataSource.numberOfSections?(in: self) ?? 0

        if numberOfSections != sections { return true }

        for section in 0..

```

A8. UITableView+TextRecognition.m

```

extension UITableView {

    /**

    Register a Class-Based 'UICollectionViewCell' subclass (conforming to 'Reusable')

    - parameter cellType: the 'UICollectionViewCell' ('Reusable'-conforming) subclass to register

    - seealso: 'register(_:,forCellWithReuseIdentifier:)'

    */

    final func register<T: UICollectionViewCell>(cellType: T.Type)

        where T: ReuseIdentifying {

```

```

        self.register(UINib.init(nibName: T.reuseIdentifier, bundle: Bundle.main),

                        forCellReuseIdentifier: T.reuseIdentifier)

    }

/**

Returns a reusable `UICollectionViewCell` object for the class inferred by the return-type

- parameter indexPath: The index path specifying the location of the cell.

- parameter cellType: The cell class to dequeue

- returns: A `Reusable`, `UICollectionViewCell` instance

- note: The `cellType` parameter can generally be omitted and inferred by the return type,
except when your type is in a variable and cannot be determined at compile time.

- seealso: `dequeueReusableCell(withReuseIdentifier:,for:)`

*/

final func dequeueReusableCell<T: UITableViewCell>(for indexPath: IndexPath, cellType: T.Type =
T.self) -> T

    where T: ReuseIdentifying {

        guard let cell = self.dequeueReusableCell(withIdentifier: cellType.reuseIdentifier, for:
indexPath) as? T else {

            fatalError(

                "Failed to dequeue a cell with identifier \(cellType.reuseIdentifier) matching type
\(cellType.self). "

                + "Check that the reuseIdentifier is set properly in your XIB/Storyboard "

                + "and that you registered the cell beforehand"

            )

        }

        return cell

    }

```

```

}

extension UICollectionView {

  /**

  Register a Class-Based `UICollectionViewCell` subclass (conforming to `Reusable`)

  - parameter cellType: the `UICollectionViewCell` (`Reusable`-conforming) subclass to register

  - seealso: `register(_:forCellWithReuseIdentifier:)`

  */

  final func register<T: UICollectionViewCell>(cellType: T.Type)

    where T: ReuseIdentifying {

      register(UINib.init(nibName: T.reuseIdentifier, bundle: Bundle.main),

        forCellWithReuseIdentifier: T.reuseIdentifier)

    }

  /**

  Returns a reusable `UICollectionViewCell` object for the class inferred by the return-type

  - parameter indexPath: The index path specifying the location of the cell.

  - parameter cellType: The cell class to dequeue

  - returns: A `Reusable`, `UICollectionViewCell` instance

  - note: The `cellType` parameter can generally be omitted and inferred by the return type,

  except when your type is in a variable and cannot be determined at compile time.

  - seealso: `dequeueReusableCell(withReuseIdentifier:,for:)`

  */

  final func dequeueReusableCell<T: UICollectionViewCell>(for indexPath: IndexPath, cellType:
T.Type = T.self) -> T

    where T: ReuseIdentifying {

      guard let cell = dequeueReusableCell(withReuseIdentifier: cellType.reuseIdentifier, for:
indexPath) as? T else {

```

```

        fatalError(
            "Failed to dequeue a cell with identifier \(cellType.reuseIdentifier) matching type
            \(cellType.self). "
            + "Check that the reuseIdentifier is set properly in your XIB/Storyboard "
            + "and that you registered the cell beforehand"
        )
    }

    return cell
}
}

```

NS_ASSUME_NONNULL_BEGIN

@implementation UIImage (TextRecognition)

/**

* Returns a scaled image to the given size.

*

* @param size Maximum size of the returned image.

* @return Image scaled according to the give size or `nil` if image resize fails.

*/

```
final class Cache<Key: Hashable, Value> {
```

```
    private let wrapped = NSCache<WrappedKey, Entry>()
```

```
    func insert(_ value: Value, forKey key: Key) {
```

```
        let entry = Entry(value: value)
```

```
        wrapped.setObject(entry, forKey: WrappedKey(key))
```

```
    }
```

```
    func value(forKey key: Key) -> Value? {
```

```

        let entry = wrapped.object(forKey: WrappedKey(key))

        return entry?.value
    }

    func removeValue(forKey key: Key) {

        wrapped.removeObject(forKey: WrappedKey(key))

    }
}

private extension Cache {

    final class WrappedKey: NSObject {

        let key: Key

        init(_ key: Key) { self.key = key }

        override var hash: Int { return key.hashValue }

        override func isEqual(_ object: Any?) -> Bool {

            guard let value = object as? WrappedKey else {

                return false

            }

            return value.key == key

        }

    }

}

```

```

private extension Cache {

    final class Entry {

        let value: Value

        init(value: Value) {

```

```

        self.value = value

    }

}

}

extension Cache {

    subscript(key: Key) -> Value? {

        get { return value(forKey: key) }

        set {

            guard let value = newValue else {

                // If nil was assigned using our subscript,

                // then we remove any value for that key:

                removeValue(forKey: key)

                return

            }

            insert(value, forKey: key)

        }

    }

}

protocol Criteriable {

    func isConform(to value: String) -> Bool

}

struct EmailCriteria: Criteriable {

    func isConform(to value: String) -> Bool {

        return value.contains("@")

    }

}

```

```
}
```

```
// NOTE: - Criteria which check, that string have any letters or numbers, not spaces only
```

```
struct SpacesCriteria: Criteriable {
```

```
    func isConform(to value: String) -> Bool {
```

```
        return !value.trimmingCharacters(in: .whitespaces).isEmpty
```

```
    }
```

```
}
```

```
struct EmptyStringCriteria: Criteriable {
```

```
    func isConform(to value: String) -> Bool {
```

```
        return !value.isEmpty
```

```
    }
```

```
}
```

```
struct PasswordLongCriteria: Criteriable {
```

```
    let long: Int
```

```
    func isConform(to value: String) -> Bool {
```

```
        return value.count >= long
```

```
    }
```

```
}
```

```
struct StringValidator {
```

```
    var criterias: [Criteriable]
```

```
    init(_ criterias: [Criteriable]) {
```

```
        self.criterias = criterias
```

```
    }
```

```
    func isValid(_ value: String, result: @escaping (ValidatorResult) -> Void) {
```

```

        if criterias.first(where: { $0.isConform(to: value) == false }) != nil {

            result(.notValid)

        }

        result(.valid)

    }

}

protocol AuthWayPresenterProtocol {

    var router: AuthWayRouterProtocol { get }

    func viewDidLoad()

    func viewWillAppear()

    func supportButtonTap()

    func setBottomButtons(buttons: [UIButton])

    func configureButtons(emailButton: AuthWayButton, codeAuthButton: AuthWayButton)

}

final class AuthWayPresenter: AuthWayPresenterProtocol {

    private enum Constants {

        static let mailPhoneAuth = "mailPhoneAuth"

        static let codeAuth = "codeAuth"

        static let bottomButtonFont = "SFUIText-Regular"

        static let fontSize: CGFloat = 14

        static let cornerRadius: CGFloat = 8

        static let borderWidth: CGFloat = 1

    }

    private weak var view: AuthWayView?

```

```

private(set) var router: AuthWayRouterProtocol

private let amplitudeManager: AmplitudeManagerProtocol

init(view: AuthWayView,

      amplitudeManager: AmplitudeManagerProtocol,

      router: AuthWayRouterProtocol

) {

    self.view = view

    self.amplitudeManager = amplitudeManager

    self.router = router

}

func viewDidLoad() {

    view?.setLocalization()

}

func viewWillAppear() {

    view?.setupNavigationBar()

    amplitudeManager.loginMainScreen()

}

func supportButtonTap() {

    router.presentRequestAccessScreen()

}

func setBottomButtons(buttons: [UIButton]) {

    guard let font = UIFont(name: Constants.bottomButtonFont, size: Constants.fontSize) else { return }

    buttons.forEach({ button in

        button.layer.masksToBounds = true

```

```

        button.layer.cornerRadius = Constants.cornerRadius

        button.layer.borderWidth = Constants.borderWidth

        button.layer.borderColor = UIColor.buttonBlueColor.cgColor

        button.titleLabel?.font = font
    })
}

func configureButtons(emailButton: AuthWayButton, codeAuthButton: AuthWayButton) {

    emailButton.imageName = Constants.mailPhoneAuth

    emailButton.text = "login_by_mail_or_phone".localized

    emailButton.isExclusiveTouch = true

    emailButton.isTouchedClosure = { [weak self] in

        DispatchQueue.main.async {

            self?.router.presentOneWayAuthScreen()

        }

        codeAuthButton.imageName = Constants.codeAuth

        codeAuthButton.text = "login_by_company_code".localized

        codeAuthButton.isExclusiveTouch = true

        codeAuthButton.isTouchedClosure = { [weak self] in

            DispatchQueue.main.async {

                self?.router.presentCompanyCodeScreen()

            }

        }

    }
}

```

```

}

protocol EnterCodePresenterProtocol {

    var router: EnterCodeRouterProtocol { get }

    func viewDidLoad()

    func viewWillAppear()

    func setupNavigationBar(navigationController: UINavigationController?,
                            navigationItem: UINavigationController)

    func requestCodeTapped(successCompletion: EmptyClosure?, failureCompletion: EmptyClosure?)

    func continueButtonTapped(
        successCompletion: EmptyClosure?,
        failureCompletion: EmptyClosure?
    )

    func setCodeField(_ textField: OneTimeCodeTextField)

}

final class EnterCodePresenter: EnterCodePresenterProtocol {

    private enum Constants {

        static let timerTime: Int = 60

        static let timeInterval: TimeInterval = 1.0

        static let requestFontSize: CGFloat = 14

    }

    private weak var view: EnterCodeView?

    private(set) var router: EnterCodeRouterProtocol

    public var code: String?

    private let login: String

```

```

private var authType: OneWayAuthPresenter.AuthType

private let amplitudeManager: AmplitudeManagerProtocol

private let navControllerSettingsUseCase: NavigationControllerSettingsUseCase

private let codeTextFieldSource: CodeTextFieldSource

private let authManager: AuthorizationManagerProtocol

init(view: EnterCodeView,

    router: EnterCodeRouterProtocol,

    login: String,

    authType: OneWayAuthPresenter.AuthType,

    amplitudeManager: AmplitudeManagerProtocol,

    navControllerSettingsUseCase: NavigationControllerSettingsUseCase,

    codeTextFieldSource: CodeTextFieldSource,

    authManager: AuthorizationManagerProtocol

) {

    self.view = view

    self.router = router

    self.login = login

    self.authType = authType

    self.amplitudeManager = amplitudeManager

    self.navControllerSettingsUseCase = navControllerSettingsUseCase

    self.codeTextFieldSource = codeTextFieldSource

    self.authManager = authManager

}

func viewDidLoad() {

    view?.setTitle(with: "restore_password".localized)

```

```

view?.setContinueButton()

view?.setLocalization()

authType == .phone

    ? view?.setCodeDescriptionLabel(with: "code_description_phone".localized)

    : view?.setCodeDescriptionLabel(with: "code_description_mail".localized)

view?.setRequestCodeButton(with: getRequestCodeButtonTitle())

waitRequestNewCode()

view?.setRequestLabelsColor(with: .gray)

view?.setConstraints()

}

func viewWillAppear() {

    amplitudeManager.loadedScreen(title: "Enter code screen")

    view?.setKeyboardNotification()

}

func requestCodeTapped(successCompletion: EmptyClosure?, failureCompletion: EmptyClosure?) {

    waitRequestNewCode()

    authManager.postRecovery(email: login) { (error, message) in

        if message == .emptyLine, error == nil {

            successCompletion?()

        } else {

            failureCompletion?()

        }

    }

}

func continueButtonTapped(successCompletion: EmptyClosure?, failureCompletion: EmptyClosure?)
{

```

```

guard let code = code, !code.isEmpty else {

    return router.presentAlert(with: "empty_field".localized)

}

router.presentHUD()

authManager.postCode(

    login: login,

    code: code,

    phone: authType == .phone

) { [weak self] (error, message) in

    guard let self = self else {

        return

    }

    self.router.dismissHUD()

    if error == nil, message == nil {

        self.amplitudeManager.enteredCode(status: "Success")

        self.router.presentNewPasswordScreen(with: self.login, code: code, authType: self.authType)

        successCompletion?()

    } else {

        self.amplitudeManager.enteredCode(status: "Fail")

        self.router.presentAlert(with: "invalid_code_confirmation".localized)

        failureCompletion?()

    }

}

}

```

```

func setCodeField(_ textField: OneTimeCodeTextField) {

    textField.configure()

    textField.didEnterLastDigit = { [weak self] code in

        self?.code = code

        textField.resignFirstResponder()

    }

}

func setupNavigationBar(navigationController: UINavigationController?, navigationItem:
UINavigationControllerItem) {

    navControllerSettingsUseCase.setupNavigationBar(navigationController: navigationController,

                                                    navigationItem: navigationItem)

}

private func waitRequestNewCode() {

    var runCount: Int = Constants.timerTime

    view?.setTimeLabel(with: runCount)

    Timer.scheduledTimer(withTimeInterval: Constants.timeInterval, repeats: true) { [weak self] timer
in

        guard let self = self else { return }

        runCount -= 1

        self.view?.setTimeLabel(with: runCount)

        if runCount == .zero {

            timer.invalidate()

            self.view?.requestLabels(codeIsHidden: false, descriptionIsHidden: true)

            self.view?.setRequestCodeButton(with: self.getRequestCodeButtonTitle())

        }

}

```

```

    }

    self.view?.requestLabels(codeIsHidden: true, descriptionIsHidden: false)

}

private func getRequestCodeButtonTitle() -> NSAttributedString {

    let attributes = [

        NSAttributedString.Key.font: UIFont.boldSystemFont(ofSize: Constants.requestFontSize),

        NSAttributedString.Key.underlineStyle: 1

    ] as [NSAttributedString.Key: Any]

    return NSAttributedString(string: "request_code".localized, attributes: attributes)

}

}

protocol NewPasswordPresenterProtocol {

    var router: NewPasswordRouterProtocol { get }

    func viewDidLoad()

    func viewWillAppear()

    func setupNavigationBar(navigationController: UINavigationController?,

                            navigationItem: UINavigationController)

    func saveButtonTapped(newPassword: UITextField, repeatedPassword: UITextField)

    func recoveryPassword(

        with newPassword: String,

        successCompletion: EmptyClosure?,

        failureCompletion: EmptyClosure?

    )

```

```

func eyeTapped()

func repeatEyeTapped()

}

final class NewPasswordPresenter: NewPasswordPresenterProtocol {

    private enum Constants {

        static let eyeClosed = "ic-eye-closed-blue"

        static let eyeOpen = "ic-open-blue"

        static let textFieldFont = "SFUIText-Regular"

        static let codeError = "400"

        static let textFieldFontSize: CGFloat = 14

    }

    private weak var view: NewPasswordView?

    private(set) var router: NewPasswordRouterProtocol

    private let login: String

    private var authType: OneWayAuthPresenter.AuthType

    private let code: String

    private let navControllerSettingsUseCase: NavigationControllerSettingsUseCase

    private let hashPasswordUseCase: HashPasswordUseCase

    private let amplitudeManager: AmplitudeManagerProtocol

    private let authManager: AuthorizationManagerProtocol

    private var eyeIsOpen: Bool = false

    private var repeatEyeIsOpen: Bool = false

    init(view: NewPasswordView,

        router: NewPasswordRouterProtocol,

        login: String,

```

```

authType: OneWayAuthPresenter.AuthType,

code: String,

hashPasswordUseCase: HashPasswordUseCase,

amplitudeManager: AmplitudeManagerProtocol,

authManager: AuthorizationManagerProtocol,

navControllerSettingsUseCase: NavigationControllerSettingsUseCase) {

self.view = view

self.router = router

self.login = login

self.authType = authType

self.code = code

self.hashPasswordUseCase = hashPasswordUseCase

self.amplitudeManager = amplitudeManager

self.authManager = authManager

self.navControllerSettingsUseCase = navControllerSettingsUseCase
}

func viewDidLoad() {

view?.setTitle(with: "restore_password".localized)

view?.setLocalization()

view?.setSaveButton()

view?.setPasswordTextFields(with: "Пароль".localized,

                             repeatedText: "confirm_password".localized,

                             attributes: textFieldAttributes(),

                             eyeImage: UIImage(named: Constants.eyeClosed))

if UIDevice.isFiveSeriesModel {

```

```

        view?.setFiveSeriesModelSettings()

    }

}

func viewWillAppear() {

    view?.setKeyboardNotification()

    amplitudeManager.loadedScreen(title: "New password screen")

}

func saveButtonTapped(newPassword: UITextField, repeatedPassword: UITextField) {

    amplitudeManager.clickedSaveNewPassword()

    guard let newPassword = checkNewPassword(newPassword: newPassword,
                                              repeatedPassword: repeatedPassword) else { return }

    recoveryPassword(with: newPassword)

}

func eyeTapped() {

    eyeIsOpen.toggle()

    let eyeImage = UIImage(named: eyeIsOpen ? Constants.eyeOpen : Constants.eyeClosed)

    let repeatedEyeImage = UIImage(named: repeatEyeIsOpen ? Constants.eyeOpen :
Constants.eyeClosed)

    view?.setPasswordFieldsSecurity(isEyeSecure: eyeIsOpen, eyeImage: eyeImage,
                                   isRepeatedEyeSecure: repeatEyeIsOpen, repeatedEyeImage: repeatedEyeImage)

}

func repeatEyeTapped() {

    repeatEyeIsOpen.toggle()

    let eyeImage = UIImage(named: eyeIsOpen ? Constants.eyeOpen : Constants.eyeClosed)

    let repeatedEyeImage = UIImage(named: repeatEyeIsOpen ? Constants.eyeOpen :
Constants.eyeClosed)

```

```

view?.setPasswordFieldsSecurity(isEyeSecure: eyeIsOpen, eyeImage: eyeImage,
                                isRepeatedEyeSecure: repeatEyeIsOpen, repeatedEyeImage: repeatedEyeImage)
}

func setupNavigationBar(navigationController: UINavigationController?, navigationItem:
UINavigationControllerItem) {

    navControllerSettingsUseCase.setupNavigationBar(navigationController: navigationController,
                                                    navigationItem: navigationItem)
}

func recoveryPassword(
    with newPassword: String,
    successCompletion: EmptyClosure? = nil,
    failureCompletion: EmptyClosure? = nil
) {
    router.presentHUD()

    let (hashedPassoword, isCrypted) = hashPasswordUseCase.hash(password: newPassword)

    authManager.postRecoveryCode(
        email: login,
        code: code,
        newPassword: hashedPassoword,
        isCrypted: isCrypted
    ) { [weak self] (error, message) in
        guard let self = self else { return }

        self.router.dismissHUD()

        if error == nil, message == "" {

            self.amplitudeManager.recoveredPassword()

            let alert = AlertService.alertWithActionHandler("password_changed".localized) { _ in

```

```

        self.router.presentOneWayAuthScreen(with: self.authType)
    }

    self.router.presentAlert(with: alert)

    successCompletion?()

} else if message == Constants.codeError {

    self.router.presentAlert(with: "invalid_code_confirmation".localized)

    failureCompletion?()

} else {

    guard let message = message else { return }

    self.router.presentAlert(with: message)

    failureCompletion?()

}

}

}

private func checkNewPassword(newPassword: UITextField, repeatedPassword: UITextField) ->
String? {

    guard let password = newPassword.text, !password.isEmpty else {

        router.presentAlert(with: "fill_all_fields".localized)

        return nil

    }

    guard let repeatPassword = repeatedPassword.text, password == repeatPassword else {

        router.presentAlert(with: "password_not_match".localized)

        return nil

    }

    return password

}

```

```
private func textFieldAttributes() -> [NSAttributedString.Key: NSObject] {  
  
    guard let regularFont = UIFont(name: Constants.textFieldFont,  
  
                                     size: Constants.textFieldFontSize) else { return [:] }  
  
    let attributes = [  
  
        NSAttributedString.Key.foregroundColor: UIColor.placeholderText,  
  
        NSAttributedString.Key.font: regularFont  
  
    ]  
  
    return attributes  
  
}  
  
}
```

Додаток Б
Слайди презентації

Національний університет “Запорізька політехніка”
кафедра програмних засобів

Дипломна робота магістра
на тему

**ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ
РОЗПІЗНАВАННЯ УПАКОВОК
МЕДИКАМЕНТІВ НАЖИВО З
ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ
МАШИННОГО НАВЧАННЯ**

Виконав
Ст. гр. КНТ-110м

Харєбашвілі І.Г.

Керівник
д.т.н., доцент

Степаненко О.О.

Рисунок Б.1 - Слайд 1



Вступ

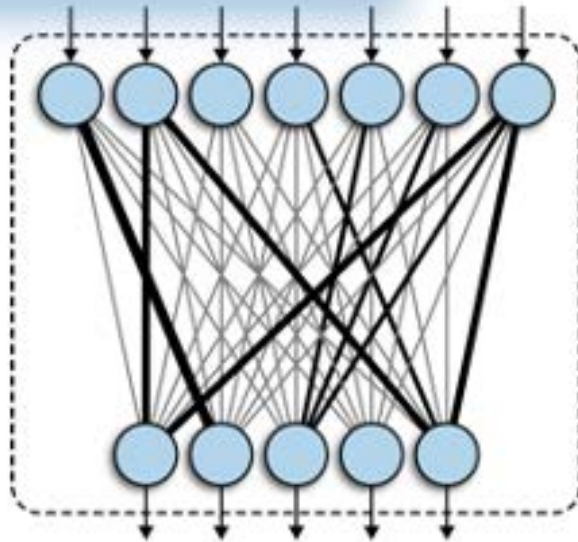
01. **Об'єкт дослідження**
Розробка системи розпізнавання упаковок медикаментів наживо з використанням технологій машинного навчання
02. **Предмет дослідження**
Застосунок, з допомогою якого можна буде сканувати упаковки з медичними препаратами та дивитись інформацію про них
03. **Задачі**
Розробка програмного забезпечення, яке допоможе простим користувачам та фармацевтам оперативно шукати інформацію про той чи інший препарат

Рисунок Б.2 - Слайд 2

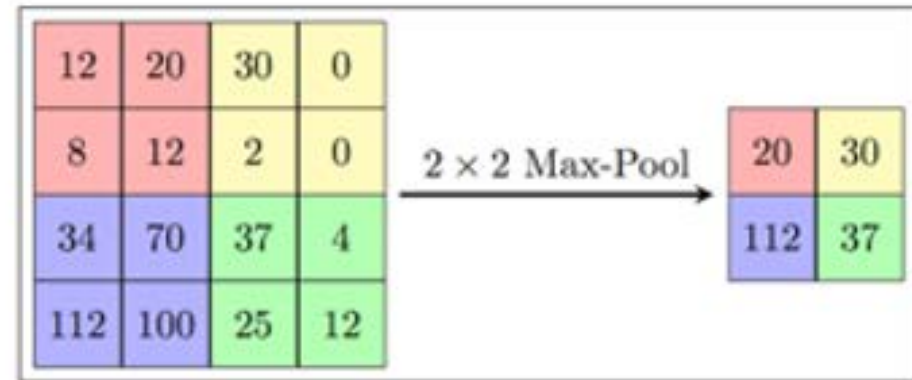


Рисунок Б.3 - Слайд 3

4 Використані алгоритми



FC NN - Працює за принципом людського зору. Певні області зорової кори активуються лише тоді, коли лінія проектується на певну частину сітківки.



CNN - Переводить вхідне зображення на набір плоских нейронів, після чого проводить порівняльний аналіз

Рисунок Б.4 - Слайд 4

5 Середовище розробки та мова проектування

Інструмент
розробки



Xcode - єдине середовище розробки для пристроїв компанії Apple та їй немає альтернатив і аналогів.

Переваги

- повністю безкоштовна;
- зручний механізм контролю версій;
- простий у використанні;
- потужний компілятор;
- НЕ ресурсомісткий, на відміну від Android Studio;
- просте конвертування; програми з iOS на macOS або iPadOS.

Недоліки

- неможливо відкрити на Windows, працює тільки на MacOS;
- не розпізнає SVG файли;
- підтримує не всі популярні мови програмування.

Мова
програмування



В якійсь мірі Swift є вдосконаленою версією Objective-C, яка містить в собі сучасні функції та підходи з інших мов.

Переваги

- спрощено синтаксис, на відміну від Objective-C;
- забезпечує безпечне управління пам'яттю;
- постійно вдосконалюється;
- простий для читання і розуміння;
- швидше в порівнянні з Objective-C.

Недоліки

- невелика популярність;
- мало стандартних бібліотек;
- складно робити низькорівневі програми;
- можна писати тільки в Xcode.

Рисунок Б.5 - Слайд 5

6 Бібліотеки машинного навчання

Під час проектування системи був вибір між двома бібліотеками, що підходять під мобільну розробку

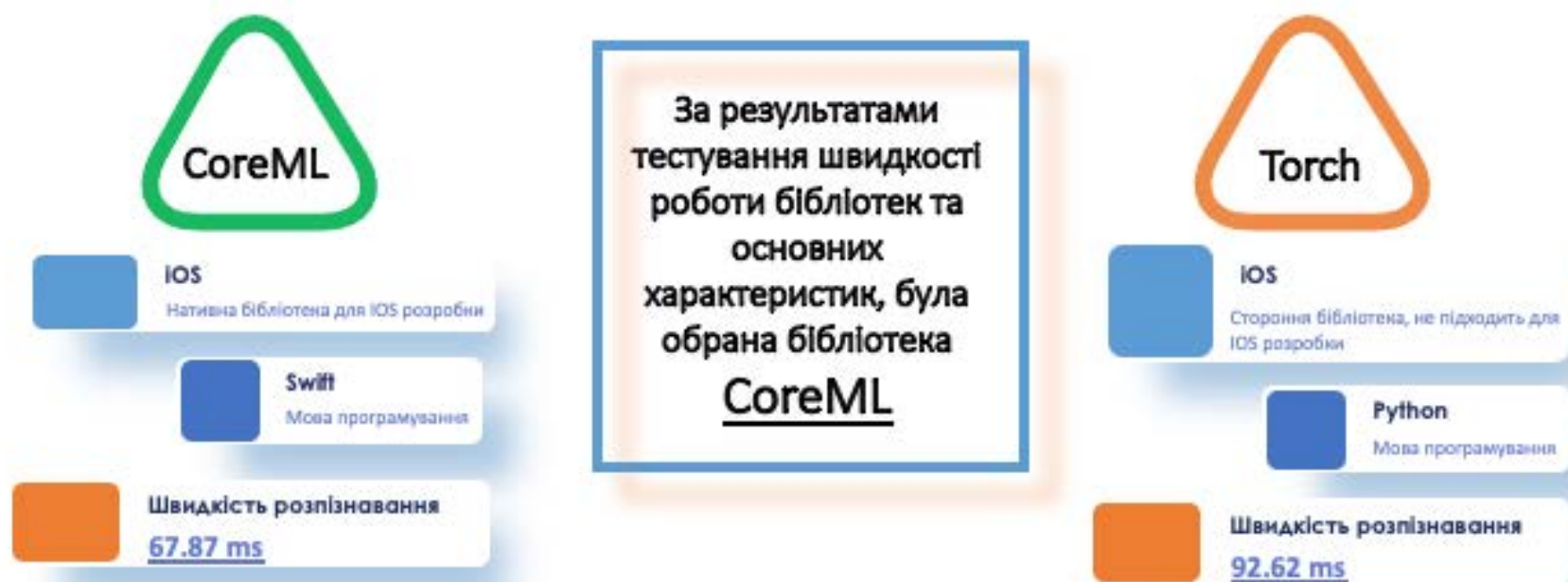


Рисунок Б.6 - Слайд 6

8 Модифікація алгоритмів машинного навчання

**Бібліотека підтримує всі класичні алгоритми машинного навчання.
Модифікація була проведена шляхом комбінування алгоритмів CNN та FC NN**

	CNN	FC NN	CNN + FC NN*
Час сканування моделі	35.2312 мс.	23.1092 мс.	25.5312 мс.
Час розпізнавання моделі	47.85 мс.	68.847512 мс.	39.5812 мс.

Алгоритм	Час всього циклу розпізнавання
CNN	83.0812 мс.
FC NN	91.9592 мс.
CNN + FC NN*	65.1124 мс.

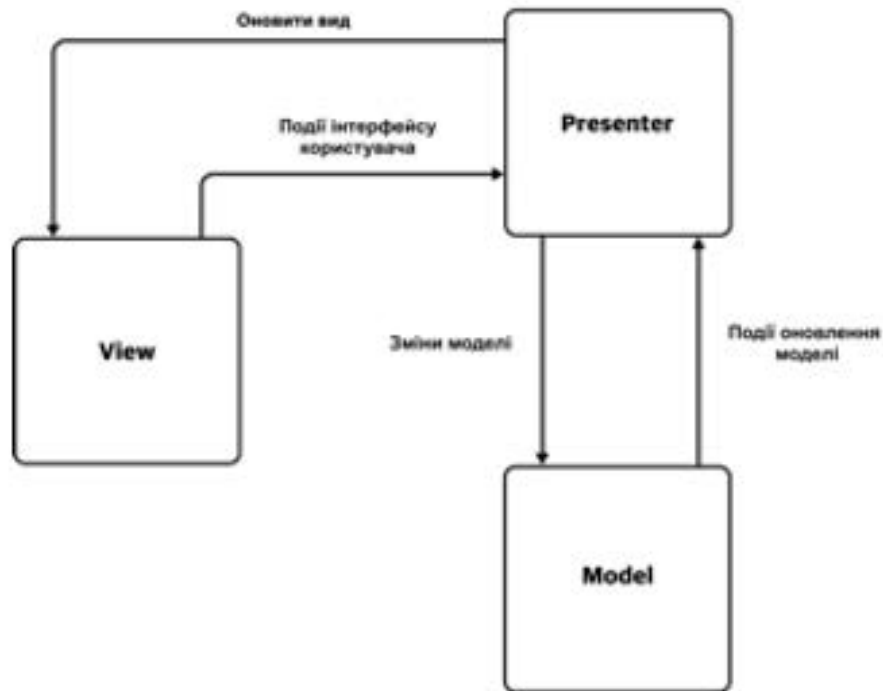
Модифікація класичних алгоритмів розпізнавання, які використовуються в бібліотеці Core ML, проводилося шляхом комбінування двох алгоритмів CNN та FC NN. В результаті ми отримали ідеальне співвідношення швидкості розпізнавання та його якості.

Нам вдалося прискорити процес розпізнавання на **36.56%**. Всі дослідження щодо покращення швидкості розпізнавання проводилися практичним шляхом.

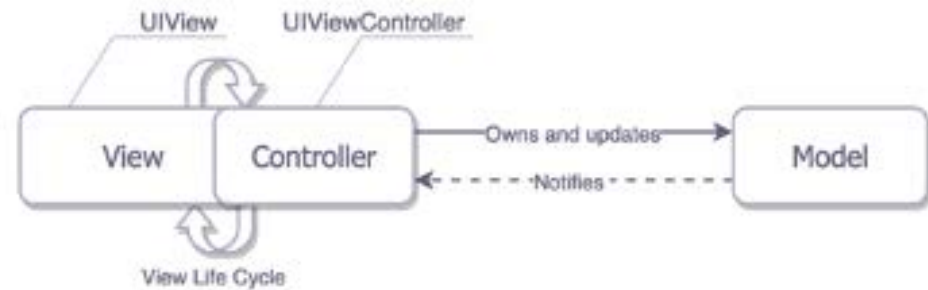
Рисунок Б.8 - Слайд 8

9

Архітектура проекту



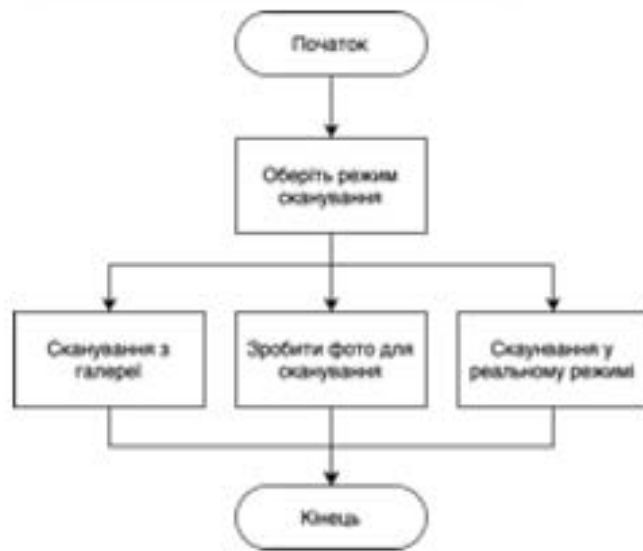
MVP



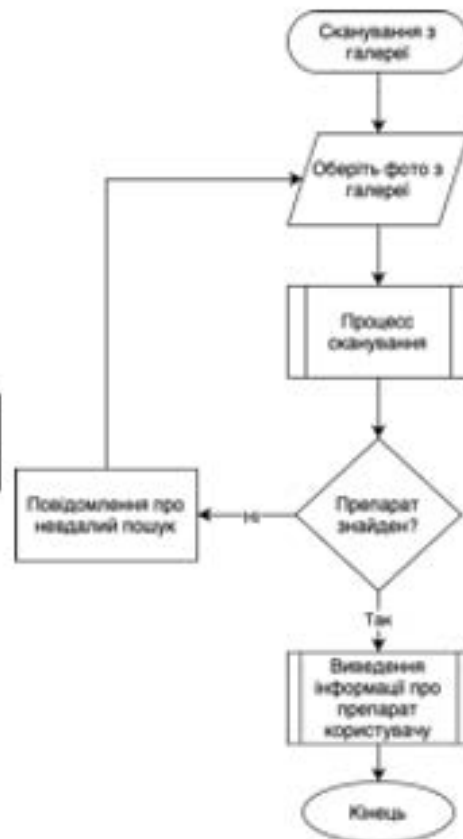
MVC

Рисунок Б.9 - Слайд 9

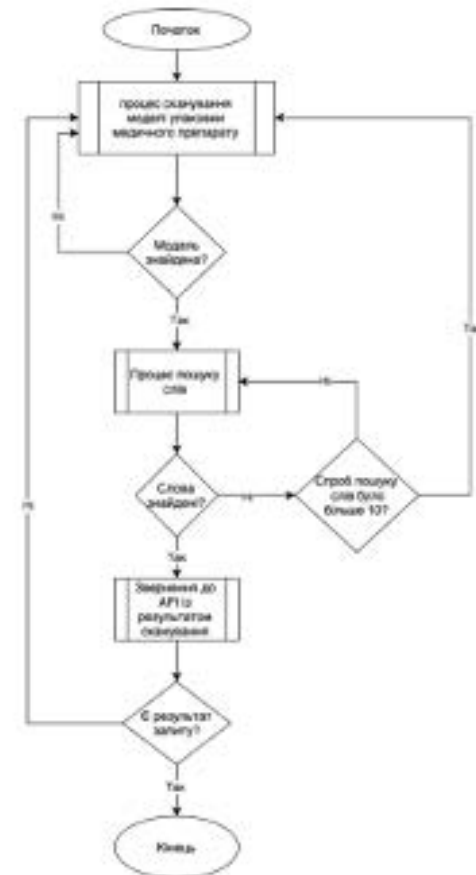
10 Блок схеми



Блок схема роботи
головної сторінки



Алгоритм роботи
сканування фото з галереї



Алгоритм процесу
сканування

Рисунок Б.10 - Слайд 10

11 UML діаграма класів

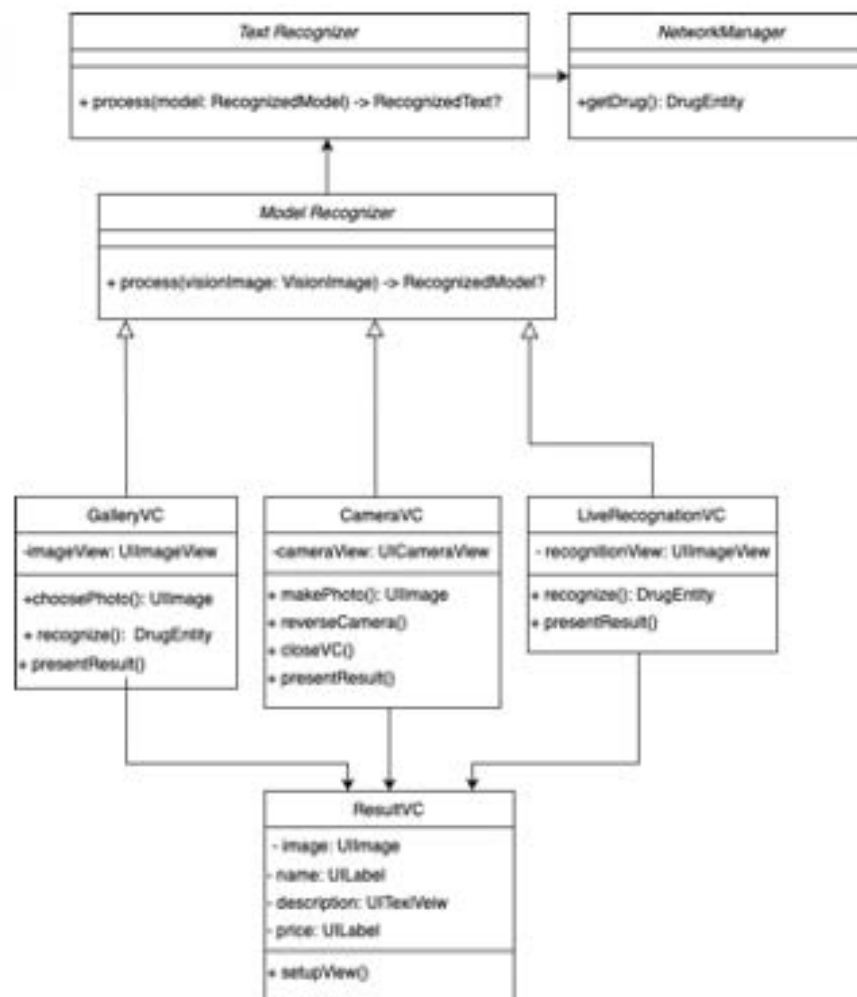


Рисунок Б.11 - Слайд 11

12 База даних

Attributes

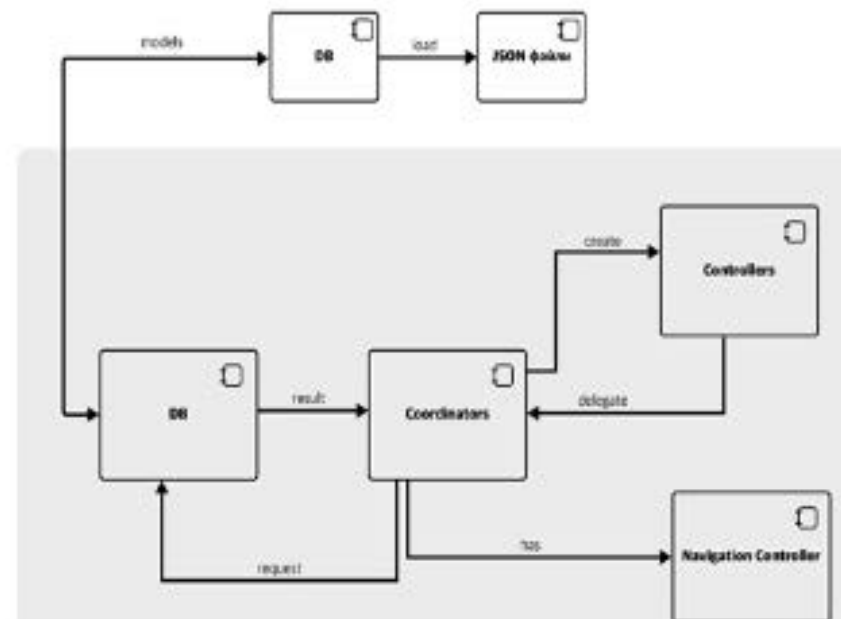
Attribute	Type	
 desc	String	⋮
 name	String	⋮
 photo	Binary Data	⋮
 price	Decimal	⋮

+ -

Relationships

Fetches Properties

Сутність DrugEntity



Структурна схема
роботи БД

Рисунок Б.12 - Слайд 12

13 Результат розробки



Процес сканування



Результат пошуку



Головна сторінка

Рисунок Б.13 - Слайд 13

Висновки

В результаті виконання магістерської роботи було створено та програмно реалізовано систему сканування та розпізнавання медичних препаратів з подальшим виводом інформації про них.

Для досягнення мети роботи було поставлено наступні задачі:

- проаналізовано існуючі аналоги;
- виявлено переваги та проблеми існуючих систем;
- сформовано вимоги для системи, що розроблялась;
- обран алгоритм розпізнавання, який працює на основі машинного навчання;
- розроблено додаток для операційної системи IOS 13.0;
- оформлено керівництво користувача і програміста;
- проведено тестування системи.

Наукова новизна полягає в тому, що для досягнення мети були використані найскладніші алгоритми машинного навчання, які дозволяють за лічені секунди обробляти дуже велику кількість інформації. Було модифіковано алгоритм розпізнавання шляхом комбінування двох алгоритмів. Завдяки чому вдалося скоротити час розпізнавання на 36.56%. Усі етапи покращення були зафіксовані у вигляді скріншотів та відображені у порівняльних таблицях

Також було закладено розширювану архітектуру програми, яка дозволить розвивати систему в майбутньому і без безлічі переробок додавати новий функціонал у додаток

Практичне значення отриманих результатів полягає в тому, що на основі розробленої моделі можна розвивати програму абсолютно в будь-який бік, а не тільки на об'єкти медицини. Для цього потрібно лише змінити пошукову модель даних з упаковок медикаментів на будь-яку іншу та обрати потрібний API сервіс для пошуку матеріалу.

Всі завдання, які були поставлені для виконання магістерської роботи, були повністю виконані.

Рисунок Б.14 - Слайд 14