

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 ТЕХНОЛОГІЙ
ІНТЕГРАЦІЇ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ ПРИ РОЗРОБЦІ
АДАПТИВНИХ ІНТЕРФЕЙСІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF TECHNOLOGY
FOR INTEGRATING AI METHODS WHEN DEVELOPING ADAPTIVE
INTERFACES

Виконав: студент 2 курсу, групи КНТ-220м
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

Нездропа І.П.

(прізвище та ініціали)

Керівник Сердюк С.М.

(прізвище та ініціали)

Рецензент Казурова А. Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
 (код і найменування)

Освітня програма (спеціалізація) Системи штучного інтелекту
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

С.О. Субботін

“ ” 20__ року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Нездропі Івану Павловичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження та програмна реалізація технологій інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів. Research and Software Implementation of Technology for Integrating AI Methods When Developing Adaptive Interfaces

керівник проекту (роботи) Сердюк Сергій Микитович к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “05” листопада 2021 року № 435

2. Строк подання студентом проекту (роботи) 01 грудня 2021 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Розробка архітектури програми. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми. 5. Додатки.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Сердюк С.М., доцент		
Нормоконтроль	Бєлова А. В., асистент		

7. Дата видачі завдання 06 вересня 2021 р.

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.	2–3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4–7 тижні	Розділ 2
4	Розробка архітектури програми.	8 тиждень	Розділ 3
5	Розробка програми.	9–10 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	11 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	12-13 тиждень	Додатки
8	Нормоконтроль та рецензування.	14–15 тижні	
9	Захист роботи.	16 тиждень	

Студент _____ Нездроп І.П. і
(підпис) (прізвище та ініціали)Керівник проєкту (роботи) _____ Сердюк С.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 99 с., 6 табл., 41 рис., 2 дод., 36 джерел.

ЛОКАЦІЯ, ПОДІЯ, РЕЛЕВАНТНА ДІЯ, НЕЙРОННА МЕРЕЖА, МЕТОД К-НАЙБЛИЖЧИХ СУСІДІВ, TENSORFLOWLITESWIFT, MATFT.

Об'єкт дослідження – методи штучного інтелекту.

Предмет дослідження – підходи та технології інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів.

Мета роботи – розробка мобільного застосуку, що дозволяє виконати передбачення подій або релевантних дій користувача залежно від локації, дня тижня та часу з використанням методів штучного інтелекту при реалізації адаптивних інтерфейсів.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування: мова програмування Swift, середовище розробки Xcode, персональний комп'ютер з процесором Core i3 під управлінням операційної системи macOS Catalina.

Результати. Проведено дослідження існуючих підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів. Створено мобільний застосунок, що вирішує задачу передбачення подій або релевантних дій користувача з використанням алгоритмів машинного навчання.

Галузь використання – прогнозування та класифікація релевантних дій користувача, отримання аналітичних даних про уподобання користувача при розробці рекомендаційних систем для маркетингових досліджень.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 99 pages, 6 tables, 41 figures, 2 appendixes, 36 sources.

LOCATION, EVENT, RELEVANT ACTION, NEURAL NETWORK, K-NEAREST NEIGHBORS METHOD, TENSORFLOWLITESWIFT, MATFT.

Object of study is a software for the methods of artificial intelligence.

Subject of study are approaches and technologies of integration of artificial intelligence methods in the development of adaptive interfaces.

The purpose of the work is to develop of a mobile application that allows to predict events or relevant user actions depending on the location and day of the week using artificial intelligence methods in the implementation of adaptive interfaces.

Materials, methods, and tools: object-oriented programming, programming language Swift, integrated development environments Xcode, a personal computer with the Core i3 processor under the control of the OS macOS Catalina.

Results. A study of existing approaches to the integration of artificial intelligence methods in the development of adaptive interfaces is conducted. A mobile application has been created that solves the problem of predicting events or relevant user actions using machine learning algorithms.

The field of use is a forecasting and classification of relevant user actions, obtaining analytical data on user preferences in the development of recommendation systems for marketing research.

ЗМІСТ

.....	C.
ABSTRACT	5
ЗМІСТ	6
ВСТУП.....	9
1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ	11
1.1 Поняття адаптивного користувацького інтерфейсу.....	11
1.2 Огляд підходів щодо розробки адаптивних інтерфейсів	13
1.3 Дослідження підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів	19
1.4 Опис алгоритму штучного інтелекту	21
1.5 Аналіз існуючих програмних продуктів	22
1.6 Висновок до першого розділу	25
2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ.....	27
2.1 Обґрунтування вибору архітектурного шаблону	27
2.2 Проектування бази даних	33
2.3 Діаграми класів та компонентів	35
2.4 Вибір мови програмування для розробки мобільного застосунку	40
2.5 Вибір середовища розробки мобільного застосунку	45
2.6 Висновок до другого розділу	46
3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ	48
3.1 Огляд бібліотек машинного навчання.....	48
3.2 Огляд використаних фреймворків, рушіїв та допоміжних бібліотек	49
3.3 Опис розроблених класів	53
3.4 Опис інтерфейсної частини	57
3.5 Висновок до третього розділу	58
4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ.....	60
4.1 Функціональні можливості системи.....	60

4.2 Порівняльний аналіз функціональних можливостей системи з відомими аналогами.....	60
4.3 Тестування компонентів розробленого мобільного застосунку.....	62
4.4 Висновок до четвертого розділу	68
ВИСНОВКИ	69
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	71
ДОДАТОК А Текст програми	74
ДОДАТОК Б Слайди презентації.....	91

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
AUI	– Adaptive User Interface;
IDE	– Integrated Development Environment;
MVC	– Model View Controller;
MVP	– Model View Presenter;
MVVM	– Model View View Model

ВСТУП

Актуальність розробки. За останній час технології розвиваються з експоненційною швидкістю. Усі пристрої об'єдані в єдину мережу, так званий Інтернет речей. Зараз мобільний телефон може розпізнавати мову людини, сканувати та розпізнавати зображення, допомогати перекладати з однієї мови на іншу, розпізнавати рослини та ін. Все це ґрунтується на використанні штучного інтелекту. Розумні програми розробляються на основі знань експертів, а потім реалізуються як вбудовані інтелектуальні функції у різних застосунках, у тому числі мобільних.

Насьогодні є актуальним використання методів штучного інтелекту для представлення потрібної інформації користувачеві або передбачення можливих дій користувача. Тому актуальним є розробка такого мобільного застосунку, що дозволить виконати інтеграцію методів штучного інтелекту при реалізації адаптивних інтерфейсів для відображення рекомендацій користувачеві.

Мета даної дипломної кваліфікаційної роботи – розробка мобільного застосуку, що дозволяє виконати передбачення подій або релевантних дій користувача залежно від локації, дня тижня та часу з використанням методів штучного інтелекту при реалізації адаптивних інтерфейсів.

Для досягнення поставленої мети в дипломній кваліфікаційній роботі потрібно вирішити наступні задачі:

- виконати огляд підходів щодо розробки адаптивних інтерфейсів;
- виконати дослідження підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів;
- описати основні функціональні вимоги до розроблюваного мобільного застосунку;
- обґрунтувати вибір технологій, середовища розробки та мови програмування;
- розробити мобільний застосунок для передбачення подій або релевантних дій користувача;

– виконати тестування компонентів мобільного застосунку.

За результатами роботи планується подати тези на науково-практичну конференцію викладачів, науковців, молодих учених, аспірантів та студентів «Тиждень науки-2022».

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Поняття адаптивного користувацького інтерфейсу

Адаптивний користувацький інтерфейс (Adaptive User Interface, AUI) – це інтерфейс користувача (User Interface), який пристосовує, тобто змінює його макет та елементи до потреб користувача або контексту, і може бути змінений таким же чином кожним користувачем. Можливість адаптації AUI властива елементам, що складають компоненти інтерфейсу; частини інтерфейсу можуть адаптуватися до інших частин інтерфейсу та впливати на них. Даний механізм зазвичай використовується для інтеграції двох логічно різних компонентів, таких як інтерактивний документ та застосунок (наприклад, веб-браузер) в єдине ціле [1].

Адаптація користувача часто є узгодженим процесом, оскільки розробники адаптивного інтерфейсу користувача одночасно надаючи засоби, за допомогою яких і дизайнери, і користувач можуть визначити їх розміщення, часто (хоча і не завжди) напіваавтоматичним, якщо не повністю автоматизованим способом. AUI в першу чергу створюється на основі характеристик системи та рівнів знань користувачів, які його використовуватимуть [1].

Переваги застосування адаптивного користувацького інтерфейсу полягають у його здатності відповідати потребам користувачів. Властивості AUI дозволяють відображати лише релевантну інформацію для поточного користувача. Це створює менше плутанини для менш досвідчених користувачів та забезпечує легкий доступ до всієї системи [1].

До недоліків застосування адаптивного користувацького інтерфейсу можна віднести наступні [1]:

- тривалий процес;
- AUI повинен розроблятися з урахуванням різних рівнів реалізації та поєднуватися зі способом вимірювання будь-яких конкретних потреб користувачів [1];

– для найефективнішої адаптації необхідно знати мету користувача. Окрім того, що це проблема сортування та інтерпретації інформації від користувача для прогнозування його потреб, це може призвести до проблем безпеки. Оскільки інформація зберігається, користувачам не вистачає конфіденційності при використанні AUI [1].

Можна виділити наступні типи адаптивного користувацького інтерфейсу в залежності від реалізації, які можуть відрізнятися залежно від обсягу інформації, доступного певного користувача або того, як користувачі використовують застосунок [1]:

– адаптивне представлення: мета адаптивного представлення є зображенням певної інформації на основі поточного користувача. Це може означати, що користувачем, що має тільки базові знання про систему, буде показано мінімальна інформація. І навпаки, користувач з розширеними знаннями отримає доступ до більш детальної інформації та можливостей. Спосіб, яким AUI може дотримуватись цієї диференціації, може бути покриттям інформації, яка буде представлена на основі рівня досвіду користувача. Друга можливість – контролювати кількість посилань на відповідних джерелах на сторінці;

– адаптивна навігація: адаптивна навігація призначена для цього, щоб налаштувати користувача до їх конкретної мети в системі, змінний спосіб, за допомогою якого система забезпечує навігацію на основі певних факторів користувача. Ці фактори можуть включати в себе рівень знань користувачів у системі / предмети, поточну мету у системі та інші відповідні фактори. Приклади адаптивної навігації можуть бути отримані різними способами, подібно адаптивному представленню. Наприклад, надання посилань, що допомагають досягти конкретної мети користувача, представлення посилання на сторінці, де знаходиться користувач, або зміна ресурсів, що доступні користувачу.

1.2 Огляд підходів щодо розробки адаптивних інтерфейсів

1.2.1 Задача побудови адаптивного користувацького інтерфейсу

Задача побудови адаптивного користувацького інтерфейсу дуже об'ємна. Для її реалізації необхідно враховувати велику кількість чинників, параметри яких можуть бути не завжди відомі [2]. Для розуміння принципів побудови АІІ розглянемо приклад його структури, та ознайомимося із способами адаптації. Адаптивний інтерфейс має включати в себе наступні основні підсистеми: систему управління інтерфейсом; діалогову систему; систему ідентифікації користувача; тестову систему; навчальну систему; базу знань про користувача; систему проектування інтерфейсу (рис.1.1) [2].

Діалогова система здійснює обмін інформацією (безпосереднє спілкування) з кінцевим користувачем.

Тестова система реалізує ряд тестів, як при реєстрації користувача в системі (примусове тестування), так і під час його роботи з програмою (оцінка результатів роботи і інтелектуальна обробка отриманих даних).

Навчальна система «навчає» інтерфейс тим знанням і правилам, які отримує тестова система в результаті оцінки і обробки даних роботи користувача з програмою.

База знань про користувача зберігає всі отримані знання про користувача і правила його роботи в системі.

Система проектування інтерфейсу, залежно від вимог користувача і його психофізичних характеристик, формує конфігурацію і тип інтерфейсу.

Система управління адаптивним інтерфейсом є ядром адаптивного інтерфейсу. Відповідно до вимог уніфікації і універсальності АІІ розділяється на незмінну і адаптивну частини.

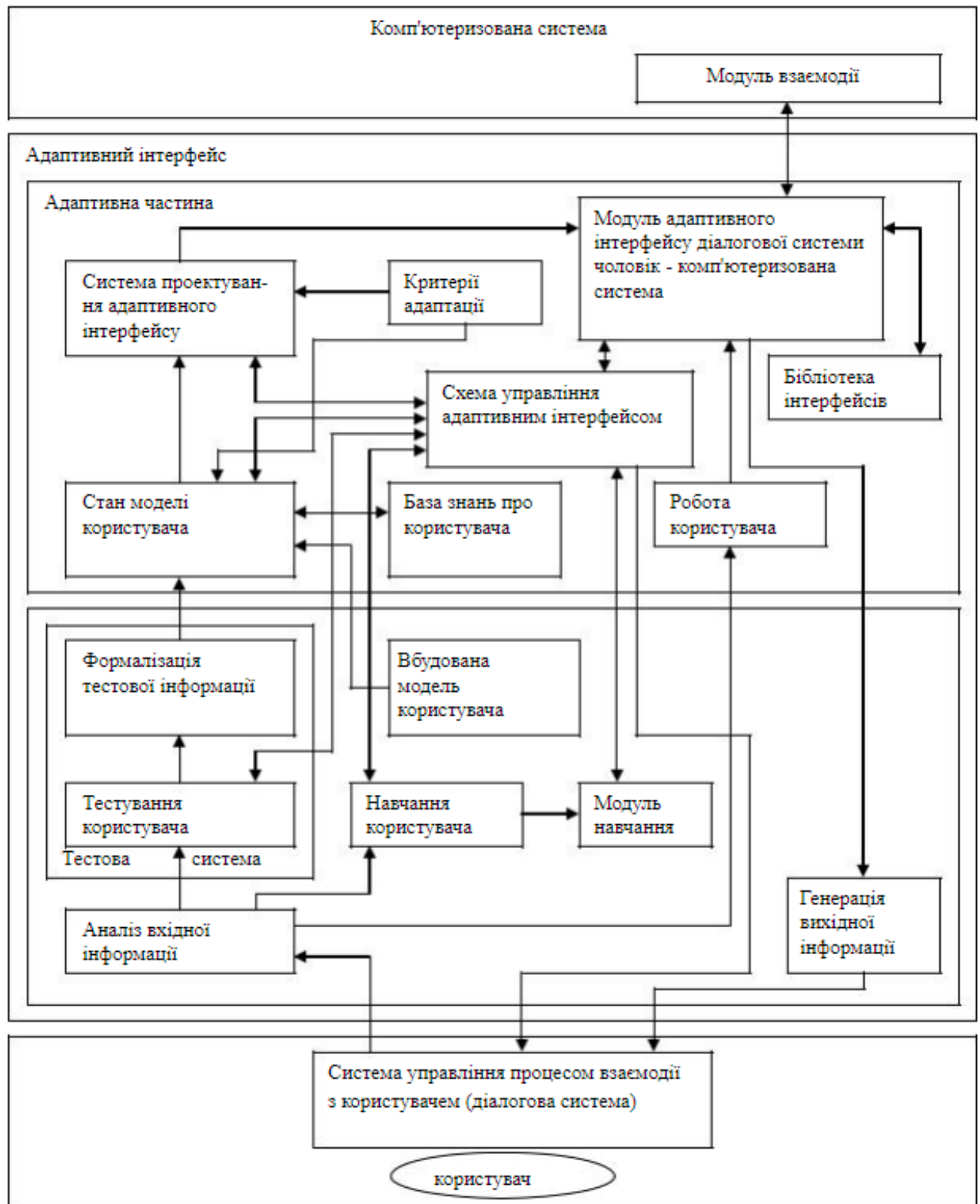


Рисунок 1.1 – Узагальнена структура адаптивного інтерфейсу [2]

Незмінна частина інтерфейсу – це частина системи, що має жорсткий сценарій роботи (рис.1.2).

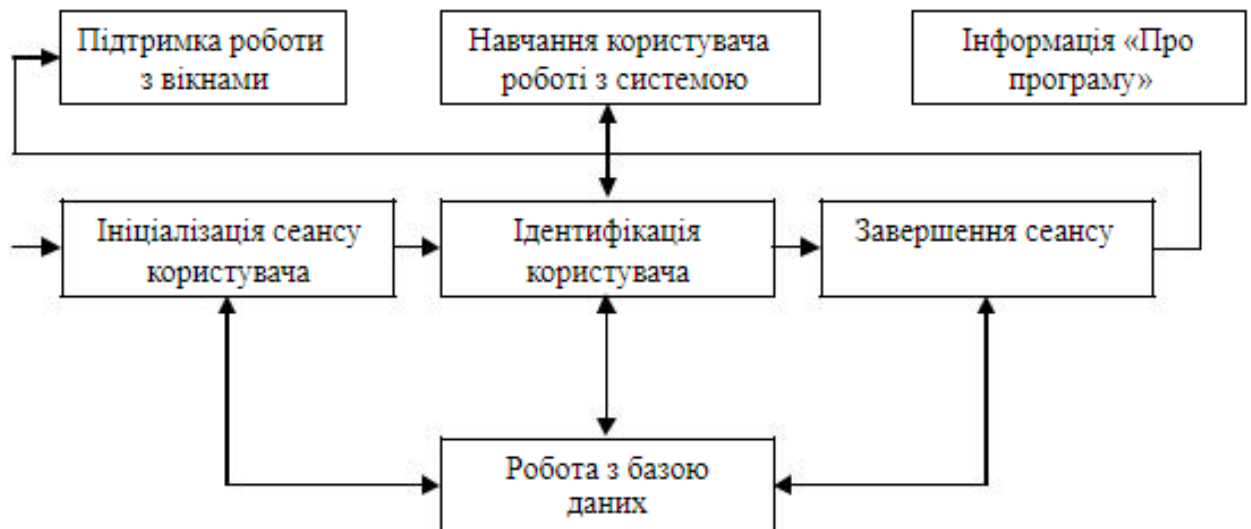


Рисунок 1.2 – Незмінна частина AI [2]

Адаптивна частина інтерфейсу є ключовою частиною АUI і відрізняється від незмінної наявністю множини різноманітних сценаріїв роботи, базується на незмінній частині і підтримується нею в своїй роботі, ініціалізується нею, бере з неї необхідні дані [2]. Приклад адаптивної частини інтерфейсу, побудованому на засадах стереотипного підходу наведен на рисунку 1.3.

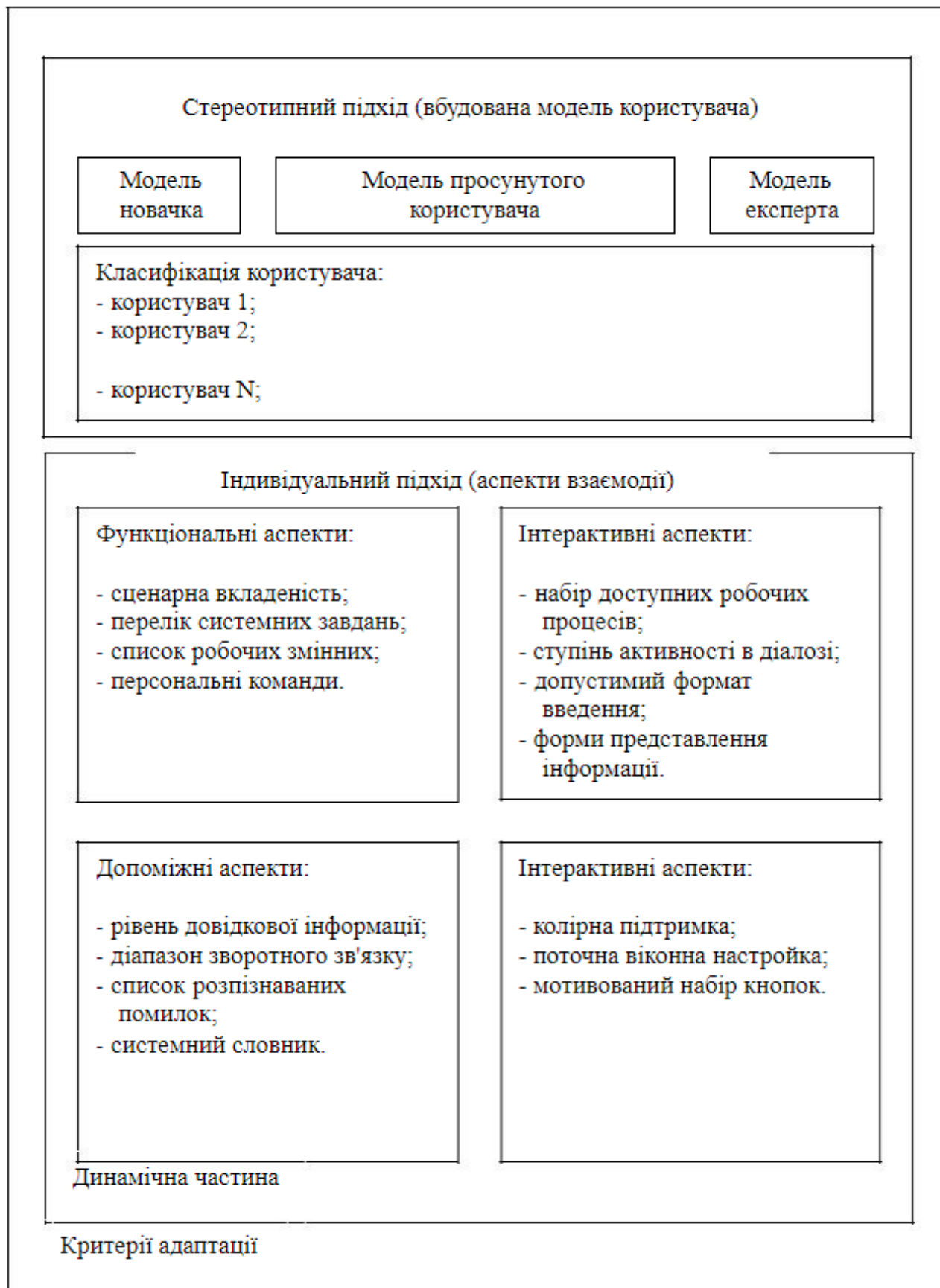


Рисунок 1.3 – Адаптивна частина АUI [2]

1.2.1 Види адаптації інтерфейсу

Основними видами адаптації інтерфейсу є параметрична і структурна адаптація [2].

Параметрична адаптація – зміна параметрів компонентів інтерфейсу, залежно від поточного стану активної моделі користувача. Тобто: розмір шрифту, розмір елементів керування, наявність піктограм в елементах керування, колір тексту різноманітних елементів керування чи елементів відображення даних тощо [2]-[3].

Структурна адаптація – зміна внутрішньої структури інтерфейсу залежно від завдань адаптації. Наприклад утворення елементів швидкого доступу до прихованих функцій, видалення чи надання додаткових елементів керування, зміна розташування функціональних панелей, тощо [2].

Приведемо приклади апіорної адаптації. Наприклад, для конкретного користувача створений інтерфейс, але у нього (користувача) виникла потреба в додаткових функціях, які не були вказані раніше. Користувач формує відповідний запит, система аналізує його і надає користувачеві можливість працювати з тим, що йому необхідне. Така адаптація апіорна тому, що відомі всі складові: вибрані користувачем функції, можливі функції користувачам даного типу, запит з боку користувача. Системі, отримавши такий сигнал, залишається за допомогою заданих алгоритмів визначити функції і запропонувати їх користувачеві. Іншим прикладом апіорної адаптації є надання можливості користувачеві змінювати зовнішній вигляд екрану або виклик довідки [2]-[3].

Прикладом апостеріорної адаптації в АUI може служити випадок збереження інформації про роботу користувача. Ситуація укладається в схему апостеріорної адаптації: якщо об'єкт є елементом списку перерахувань, замість нього система відновить його найближчого сусіда і користувачеві немає необхідності шукати необхідну позицію списку для продовження своєї роботи [2].

Адаптації з моделлю полягає в тому, що інтерфейс налаштовує свої параметри та структуру відповідно до поточної моделі користувача. Модель користувача враховує і психофізичні особливості конкретного користувача і сценарії його роботи. Якщо адекватної моделі такого користувача немає, то проводиться ідентифікація або синтез моделі стану користувача в даний момент і надається можливість пройти тестування, за результатами якого генерується модель користувача. Після проходження користувачем, наприклад, тесту Люшера йому система вибирає колірну схему зовнішнього вигляду інтерфейсу [2].

Приведемо ще приклад адаптації з моделлю – самоналагоджувальній моделі [2]-[3]. Користувач працює з множиною функцій. Логічно взаємозв'язані функції розташовуються в одному вікні. Система стежить за тим, з якими вікнами він працює зараз і до яких не звертався. Приймається рішення – не активні вікна прибрати з екрану. Тобто, якщо користувач півгодини не звертався до n -ого вікна, то воно згортається. Якщо користувач не звертався до даного згорнутого вікна протягом години, то воно видаляється взагалі [2].

Як відомо, труднощі адаптації з моделлю зводяться до синтезу моделі об'єкту, а сама адаптація зводиться до рішення оптимізаційної задачі вибору такої адаптуючої дії, яка задовольнила б цілям адаптації. Модель користувача складається з двох частин: функціональною складовою і психофізичною складовою [2].

Функціональна складова створюється шляхом аналізу інформації введеної користувачем на етапі реєстрації в системі, проходженні тестів, та аналізу його дій. Під час роботи користувача в системі вся інформація про його роботу зберігається у функціональній моделі користувача [2].

Психофізична складова моделі користувача визначається в ході тестування користувача і вибраним ним варіантом зовнішнього вигляду інтерфейсу (колірні схеми). Отримавши таку модель система генерує керуючу дію [2].

Центральним моментом організації гнучкого діалогового процесу AUI є відбір особистих характеристик користувача для включення їх до складу характеристик, що визначають необхідність внесення змін в процес інтерактивної взаємодії. В AUI використовуються характеристики користувачів [2].

Компонентом адаптації системи є також аналіз цілей і аналіз дій користувача. Досягнення цілей вимагає від користувачів здійснення певних дій. На початку роботи користувач може визначати набір функцій, необхідних для роботи в даний момент [2]-[3].

Відповідно для кожного з класів користувачів повинен бути визначений перелік можливих завдань. Окремі функції можуть повторюватися для різних класів користувачів. Функції з цього переліку і пропонуються [2].

В результаті цих кроків на екрані відображається інтерфейс, що був синтезований відповідно конкретному користувачеві. Даний інтерфейс може змінюватись під час роботи при зміні вимог, навиків і цілей роботи користувача [2].

1.3 Дослідження підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів

Виходячи з огляду підходів щодо розробки адаптивних інтерфейсів можна виділити наступні можливі варіанти інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів:

- розробка алгоритмів штучного інтелекту;
- використання спеціалізованих бібліотек штучного інтелекту в залежності від мови програмування;
- розробка набору широко використовуваних функцій або прикладного програмного інтерфейсу (Application Programming Interface, API).

1.3.1 Розробка алгоритмів штучного інтелекту

Особливістю даного підходу є те, що необхідно виконати програмну реалізацію алгоритмів штучного інтелекту при побудові адаптивного інтерфейсу користувача. Недоліком даного підходу – витрати часу розробника на написання відповідного модулю та спостерігається залежність від мови програмування або розмітки.

1.3.2 Використання спеціалізованих бібліотек машинного навчання

При реалізації адаптивного інтерфейсу користувач можна використовувати вже готові рішення, а саме спеціалізовані бібліотеки, що дозволяють скоротити процес інтеграції та зменшити витрати часу розробника. Існує багато бібліотек, наприклад TensorFlow [4], що надає бібліотеку готових алгоритмів чисельних обчислень, реалізованих через графи потоків даних (data flow graphs). Вузли в таких графах реалізують математичні операції або точки входу/виводу, в той час як ребра графу представляють багатовимірні масиви даних (тензори), які перетікають між вузлами. Вузли можуть бути закріплені за обчислювальними пристроями і виконуватися асинхронно, паралельно обробляючи разом все підходящі до них тензори, що дозволяє організувати одночасну роботу вузлів в нейронній мережі за аналогією з одночасною активацією нейронів в мозку [4].

1.3.2 Розробка прикладного програмного інтерфейсу

Для інтеграції методів штучного інтелекту можна написати серверну програму, яка буде реалізувати прикладний програмний інтерфейс, або розробити REST API з використанням відповідної мови програмування, що виконує певні https запити: відправку даних (у json форматі) та обробку їх з використанням скрипту та отримання потрібної відповіді – результату обробки.

1.4 Опис алгоритму штучного інтелекту

В роботі для реалізації адаптивного інтерфейсу буде використовуватися комбінований алгоритм нейронної мережі (автокодувальник) та методу k найближчих сусідів (рис. 1.4).

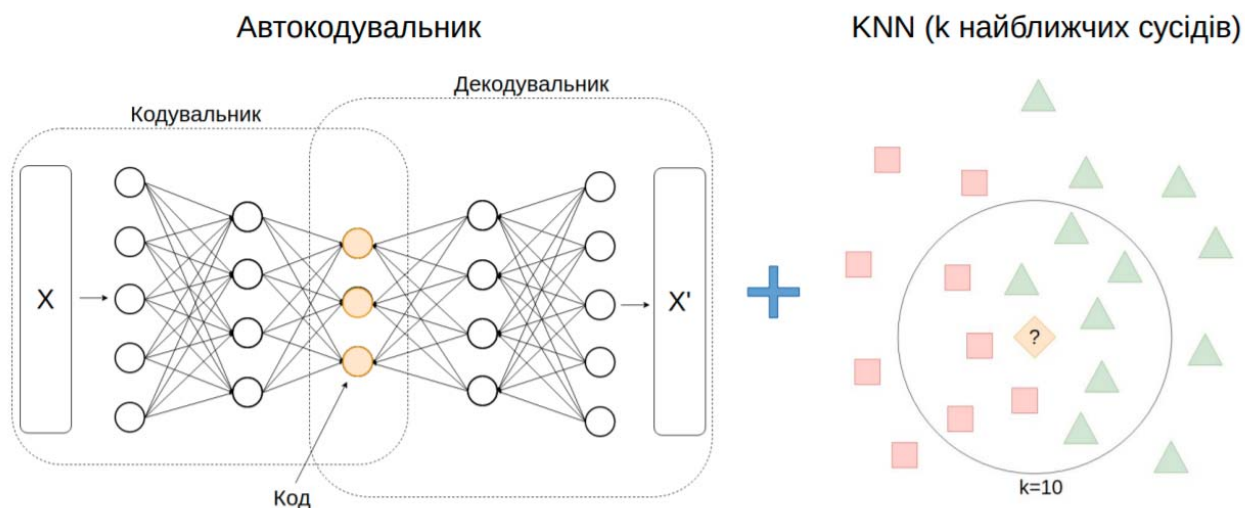


Рисунок 1.4 – Комбінований алгоритм нейронної мережі (автокодувальник) та методу k найближчих сусідів

Автокодувальник – тип штучних нейронних мереж, що дозволяє застосовувати навчання без учителя з використанням методу зворотного поширення помилки. Найпростіша архітектура автокодувальника – мережа прямого поширення, без зворотних 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 [5].

Метод k найближчих сусідів (англ. *k-nearest neighbor method*) – непараметричний класифікаційний метод, де для класифікації об'єктів у рамках простору властивостей використовуються відстані (евклідові), пораховані до усіх інших об'єктів. Вибираються об'єкти, до яких відстань найменша, і вони виділяються в окремий клас [6].

Позначення: A – поточні дані; A' – вектор-код; D – база даних виду $B - B' - Y$, де B – нормальні історичні дані, B' – закодовані дані, Y – дії користувача; N – розмір історії (останні записи); HN – останні N записів у базі даних D ; k – кількість сусідів для алгоритму kNN ; S – деяка гранична відстань для алгоритму kNN ; E – кодувальник (нейромережа).

Наведемо опис алгоритму :

Крок 1. У певний час система збирає поточні дані A про локацію, час та дату.

Крок 2. Дані A кодуються кодувальником E у вектор-код A' .

Крок 3. Алгоритмом kNN знаходиться найчастіша дія Y' з HN серед k найближчих векторів.

Крок 4. Якщо відстань до найближчого серед k вектор-коду більша, ніж S , то перейти до кроку 1.

Крок 5. Користувачу пропонується виконати дію Y' .

Крок 6. Якщо користувач відмовляється, то перейти до кроку Крок 1, інакше перейти до кроку 7.

Крок 7. У базу даних D заноситься запис вигляду $A - A' - Y'$.

1.5 Аналіз існуючих програмних продуктів

Насьогодні можна виділити такі напрями розробки програмного забезпечення для мобільних пристроїв з використанням штучного інтелекту [7]:

- розпізнавання тексту;
- біометрія;
- розпізнаванн облич та емоцій;
- обробка природної мови та розпізнавання мови;
- автоматизоване міркування.

Розглянемо приклади існуючих програмних продуктів для мобільних пристроїв, що використовують вищенаведені напрями штучного інтелекту:

Мобільний застосунок FineScanner AI, що дозволяє сканувати документи та зображення (рис. 1.5).

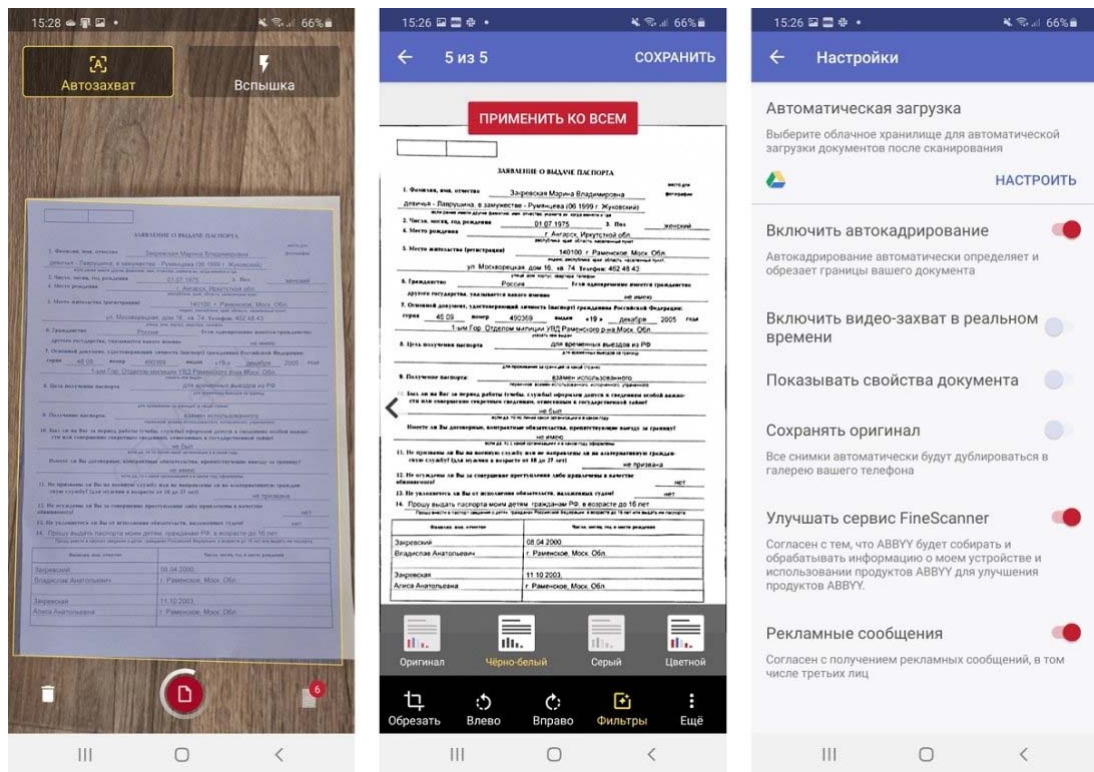


Рисунок 1.5 – Приклади інтерфейсу мобільного застосунку FineScanner AI [7]

Користуватися даним мобільним застосунком можуть власники iOS- та Android-пристроїв. Програма зможе переформатувати будь-які друковані документи 12 популярних форматів, серед яких DOCX, XLSX, PPTX, PDF і JPG. Інтегрована технологія OCR (оптичне розпізнавання символів – англ. Optical Character Recognition, OCR) розпізнає тексти на 193 мовах. Особливістю сервісу є підтримка технологій МН на базі нейронних мереж [7].

Мобільний застосунок Hemingway App працює на основі технології розпізнавання текстів і дозволяє їх швидко та якісно відредагувати (рис. 1.6). Програма запропонує синоніми, покаже речення, складні для читання, дозволить відформатувати текст – додати списки, заголовки, посилання [7].

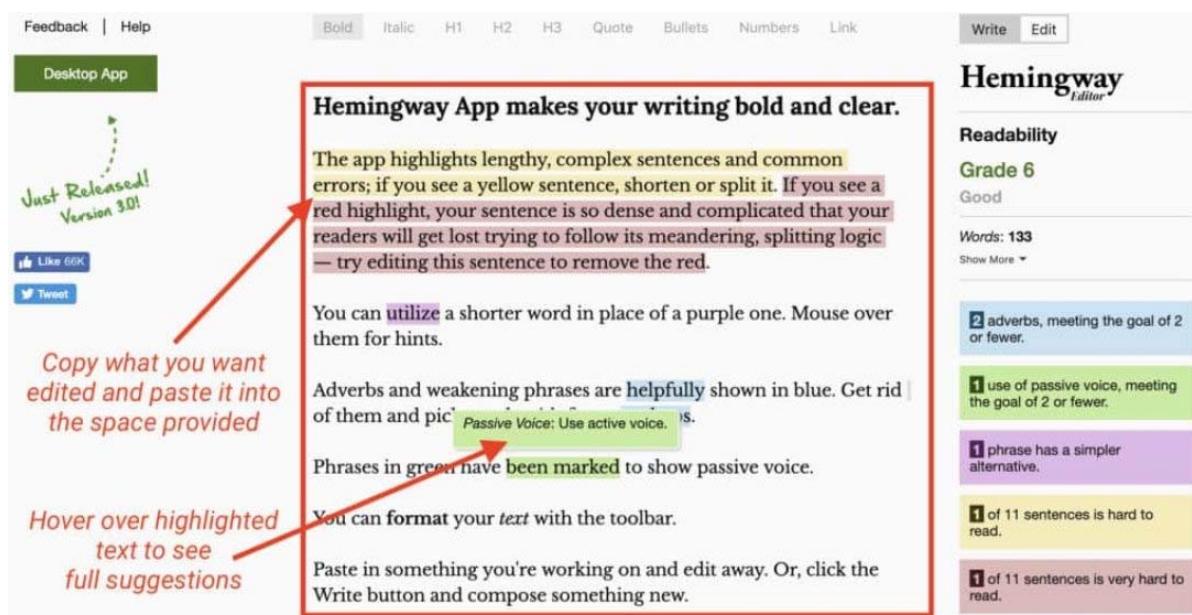


Рисунок 1.6 – Приклади інтерфейсу та принцип роботи ШІ-редактора Hemingway App [7]

Мобільний застосунок Replika (рис. 1.7) дозволяє спілкуватися не тільки на загальні теми, а на ті, які дійсно цікаві користувачеві. Спілкування програма вибудовує на базі даних, отриманих з соціальних мереж користувача. Алгоритми машинного навчання допомагають їй швидко навчатися, запам'ятовувати відповіді та реакції користувача [7].

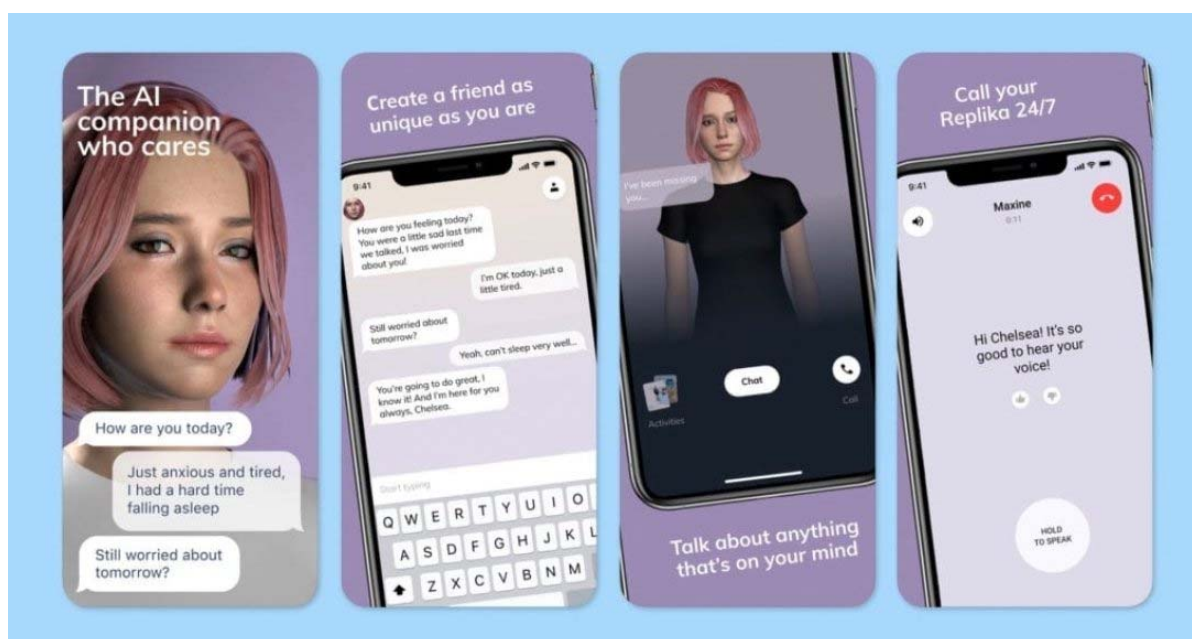


Рисунок 1.7 – Приклади інтерфейсу програми-співрозмовник Replika [7]

Google AutoDraw – сервіс [7], що здатний перетворити малюнки, зроблені від руки, на якісні кліп-арти (рис.1.8). Працює все просто: штучний інтелект аналізує малюнки, а потім пропонує аналогічні, але виконані професійними художниками й ілюстраторами [7].

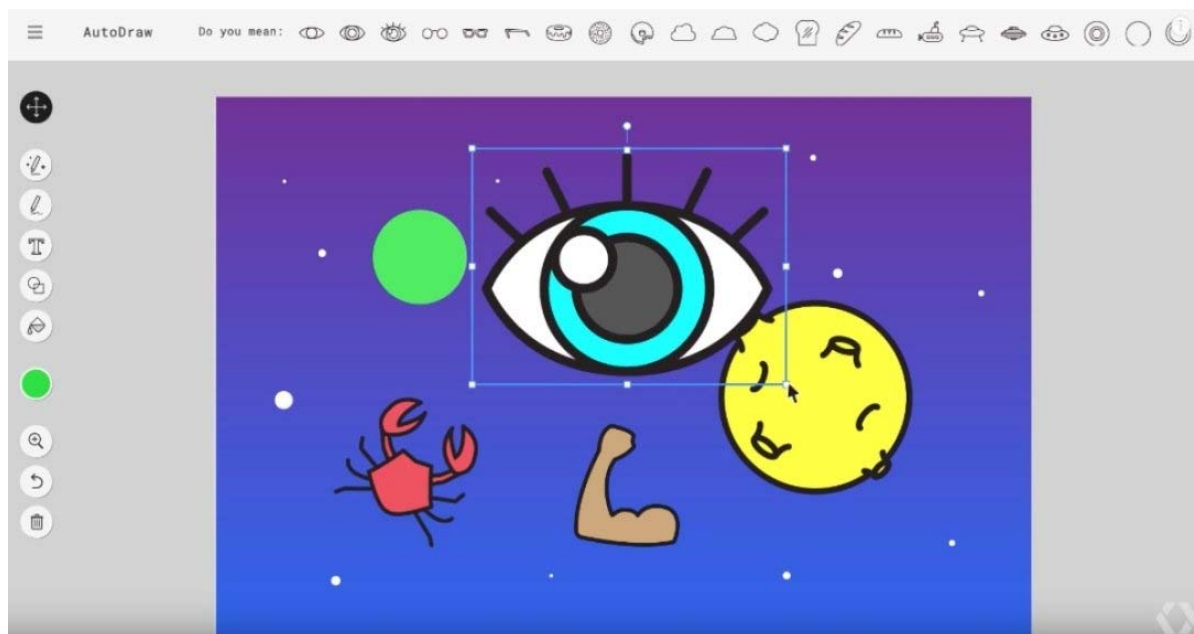


Рисунок 1.8 – Приклади інтерфейсу програми Google AutoDraw [7]

Розглянуті мобільні застосунки мають адаптивний інтерфейс та вбудований функціонал з використанням алгоритмів машинного навчання. Тому актуальною задачею є розробити програмне забезпечення, що дозволяє виконати інтеграцію методів штучного інтелекту при реалізації адаптивних інтерфейсів.

1.6 Висновок до першого розділу

В розділі було розглянуто поняття адаптивного користувацького інтерфейсу, виконано огляд підходів щодо розробки адаптивних інтерфейсів та дослідження підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів.

Було прийнято рішення виконати інтеграцію у вигляді модуля з використанням спеціалізованих бібліотек машинного навчання TensorFlow та NumPy при розробці адаптивного інтерфейсу мобільного застосунку.

В роботі для реалізації адаптивного інтерфейсу буде використовуватися комбінований алгоритм нейронної мережі (автокодувальник) та методу k найближчих сусідів.

Розроблений мобільний застосунок буде вирішувати задачу передбачення подій або релевантних дій для користувача.

Розроблена система повинна мати наступні функціональні можливості:

- збирати дані користувача, такі як: локація, дата, час;
- збирати події, в даному випадку дзвінки та історію браузера;
- при зміні локації (не координат, а саме локації) відповідну локацію та дату її зміни потрібно обробити за допомогою алгоритмів штучного інтелекту з використанням спеціалізованих бібліотек машинного навчання TensorFlow та NumPy та запропонувати найбільш релевантну дію, пов'язану з вихідними даними.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Обґрунтування вибору архітектурного шаблону

Для розробки мобільних застосунків використовують наступні архітектурні шаблони: MVC, MVP, VIP(Clean Swift), MVVM. Наведемо основні особливості наведених архітектурних шаблонів.

Модель - Представлення - Контролер (Model View Controller, MVC) – архітектурний шаблон (рис. 2.1), який використовується під час проєктування та розробки програмного забезпечення. Даний шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача [8].

Мета шаблону – гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності [8].

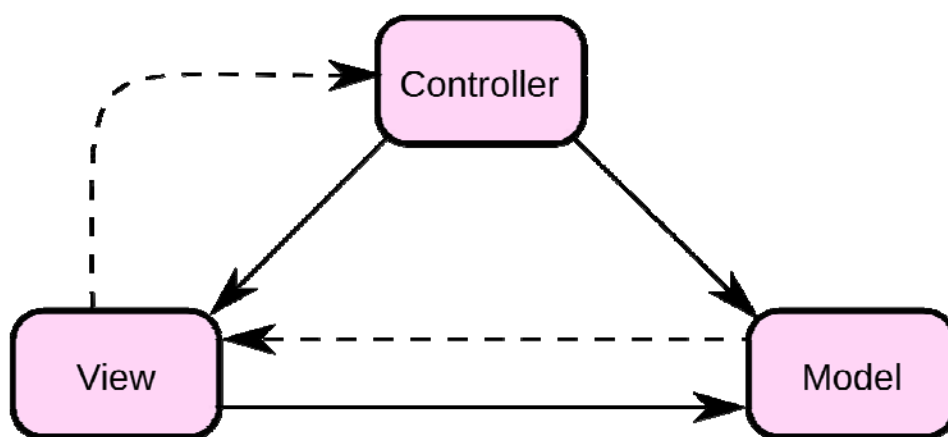


Рисунок 2.1 – Схема шаблону проєктування MVC [8]

У рамках архітектурного шаблону модель–представлення–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами [8].

Модель (Model) відповідає за зберігання даних та їх структуру. Представлення (View) відповідальний за представлення цих даних користувачеві, тобто інтерфейс програми. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель [8].

Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.

Представлення може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії. Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду. Модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних. Представлення може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані. У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля. Зареєстровані події транслуються в різні запити, що спрямовуються компонентам моделі або об'єктам, відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації. Таким чином, якщо користувач

через контролер внесе зміни до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися [8].

Модель–Представлення–Пред'явник (Model View Presenter, MVP) – шаблон проектування, похідний від MVC, що відділяє візуальне відображення та поведінку обробки подій у різні класи, а саме: Представлення (View) та Пред'явник (Presenter) [9]. На рисунку 2.2 наведено схему шаблону проектування MVP [9].

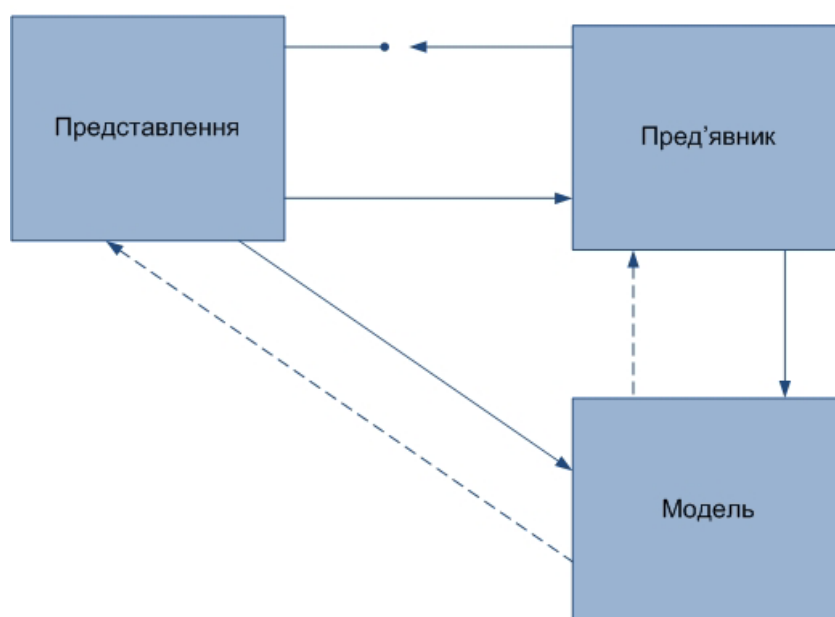


Рисунок 2.2 – Схема шаблону проектування MVP [9]

Передумови для застосування шаблону наступні:

- виникає необхідність відділення бізнес-логіки від логіки інтерфейсу користувача, щоб зробити його простішим для розуміння та підтримки;
- виникає задача надати доступ до коду різним сторінкам, що потребують однакової поведінки;
- потрібно максимізувати обсяг коду, що підлягає автоматизованому тестуванню [9].

Модель являє собою клас для визначення даних, які будуть відображатися або над якими будуть проводитися інші дії у інтерфейсі користувача.

Клас Представлення управляє елементами на сторінці, та направляє події до класу Пред'явника.

Пред'явник містить логіку реагування на події, оновлює Модель (бізнес-логіки і даних з програми) і, в свою чергу, маніпулює станом Представлення.

Clean Swift або VIP (View controller, Interactor та Presenter) – даний архітектурний шаблон був вперше введений Реймондом Лоу [10]. Ідея полягала в тому, щоб вирішити проблему контролера Massive View.

Структура шаблону складається з наступних компонент: View Controller, Interactor, Presenter, Worker, Models, Router [10].

Контролер представлення, інтерактор і пред'явник є трьома основними компонентами Clean Swift. Вони діють як вхід і вихід один для одного, як показано на наступній схемі (рис.2.3).

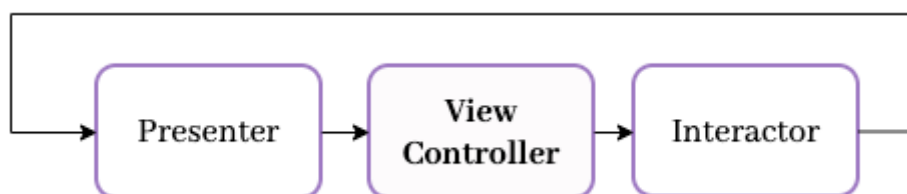


Рисунок 2.3 – Схема шаблону проектування VIP [10]

Вихід контролера перегляду підключений до входу інтерактора. Вихід інтерактора з'єднаний з входом доповідача. Вихід презентатора підключений до входу контролера перегляду. Це означає, що потік контролю завжди односпрямований [10].

Model-View-ViewModel (MVVM) – шаблон проектування, що застосовується під час проектування архітектури застосунків [11]. Публічно вперше був представлений Джоном Госсманом (John Gossman) у 2005 році як модифікація шаблону Presentation Model. MVVM орієнтований на такі сучасні

платформи розробки, як Windows Presentation Foundation та Silverlight від компанії Microsoft.

MVVM полегшує відокремлення розробки графічного інтерфейсу від розробки бізнес логіки (бек-енд логіки), відомої як модель (можна також сказати, що це відокремлення представлення від моделі). Модель представлення є частиною, яка відповідає за перетворення даних для їх подальшої підтримки і використання. З цієї точки зору, модель представлення більше схожа на модель, ніж на представлення і оброблює більшість, якщо не всю, логіку відображення даних. Модель представлення може також реалізовувати патерн медіатор, організовуючи доступ до бек-енд логіки навколо множини правил використання, які підтримуються представленням [11].

MVVM використовується для відокремлення моделі та її відображення. Необхідністю цього є надання можливості змінювати їх незалежно одну від одної. Наприклад, розробник працює над логікою роботи з даними, а дизайнер – з користувацьким інтерфейсом [11].

MVVM була створена з метою поділу праці дизайнера і програміста, що є неможливим, коли Java-розробник намагається побудувати GUI в Swing або розробник на Visual C++ намагається створити користувацький інтерфейс в MFC. Розробники кмітливі і мають безліч навичок, але створення зручних і привабливих інтерфейсів вимагає абсолютно інших талантів, ніж ті, якими вони володіють. Ця робота більше підходить для дизайнерів інтерфейсів. Хороші дизайнери інтерфейсів краще знають, чого бажають користувачі, ніж експерти в області проєктування і написання коду. Зрозуміло, буде краще, якщо дизайнер інтерфейсів створить інтерфейс, а розробник напише код, який реалізує логіку цього інтерфейсу, але технології типу Swing або MFC просто-напросто не дозволяють чинити таким чином [11].

В MVC/MVP зміни у користувацькому інтерфейсі не впливають безпосередньо на модель, а йдуть через Контролер/Presenter. У таких технологіях, як WPF та Silverlight, присутня концепція «зв'язування даних», що

дозволяє зв'язувати дані із візуальними елементами в обидві сторони. Архітектура MVVM вирішує цю проблему ясным поділом відповідальності [11]:

- розробка користувацького інтерфейсу здійснюється дизайнером інтерфейсів за допомогою технології, більш-менш природної для такої роботи (XML);

- логіка користувацького інтерфейсу реалізується розробником як компонент ViewModel;

- функціональні зв'язки між користувацьким інтерфейсом та ViewModel реалізуються через біндинги (bindings), які, по суті, є правилами типу «якщо кнопка А була натиснута, повинен бути викликаний метод onButtonAClick() з ViewModel». Біндинги можуть бути написані в коді або визначені декларативним шляхом (Android використовує обидва типи).

Архітектура MVVM використовується в тому чи іншому вигляді усіма сучасними технологіями, наприклад Microsoft WPF і Silverlight, Oracle JavaFX, Adobe Flex, AJAX [11].

Шаблон MVVM ділиться на три частини [11]:

- модель (Model), як і в класичному шаблоні MVC, Модель являє собою фундаментальні дані, що необхідні для роботи застосунку. Вигляд (View) як і в класичному шаблоні MVC;

- вигляд – це графічний інтерфейс, тобто вікно, кнопки тощо;

- модель вигляду (ViewModel, що означає «Model of View») з одного боку є абстракцією Вигляду, а з іншого надає обгортку даних з Моделі, які мають зв'язуватись. Тобто вона містить Модель, яка перетворена до Вигляду, а також містить у собі команди, якими може скористатися Вигляд для впливу на Модель. Фактично ViewModel призначена для того, щоб здійснювати зв'язок між моделлю та вікном, відслідковувати зміни в даних, що зроблені користувачем, відпрацьовувати логіку роботи View (механізм команд).

Для реалізації мобільного застосунку прийнято рішення використати архітектурний шаблон MVC тому, що проєкт невеликого обсягу, даний

архітектурний шаблон доцільно застосувати, модулі програми будуть логічно розділені на компоненти Модель, Представлення та Контроллер.

2.2 Проектування бази даних

Мобільний застосунок, що розробляється, повинен вирішувати задачу передбачення подій або релевантних дій для користувача. Тому було вирішено створити базу даних з наступними таблицями: Вибрані контакти, Події та Нещодавні дзвінки (табл. 2.1 – табл. 2.3).

Таблиця 2.1 – Вибрані контакти FavoriteContact

Атрибут	Тип
firstName	String
identifier	String
imageData	BinaryData
note	String
organization	String
phone	String
secondName	String

Таблиця 2.2 – Нещодавні дзвінки RecentCall

Атрибут	Тип
date	Date
phone	String

Таблиця 2.3 – Подія Event

Атрибут	Тип
date	Date
latitude	Double
longitude	Double
pushDescription	String
source	String
typeOfEventValue	Int16

Наведемо схему бази даних (рис.2.4).

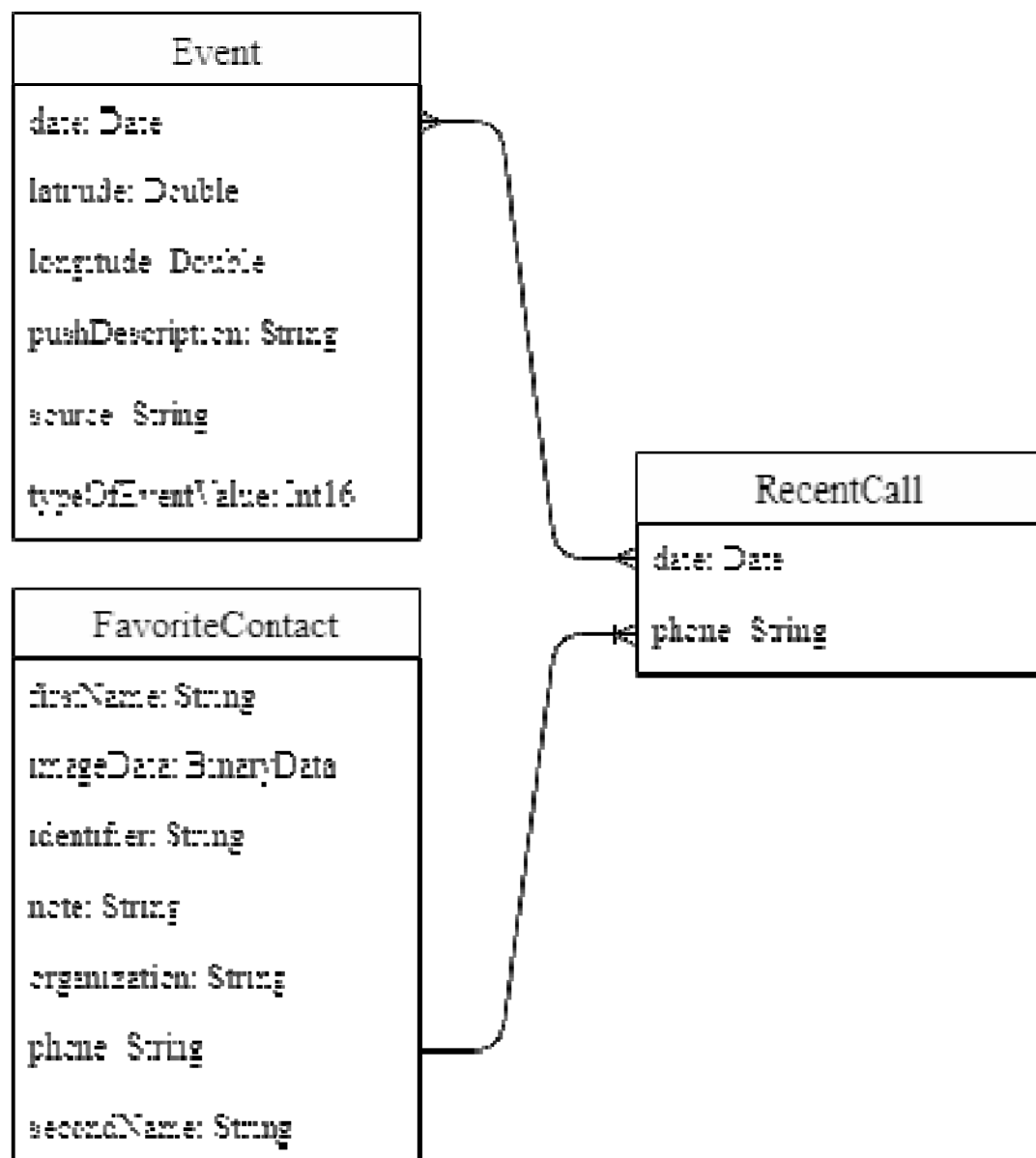


Рисунок 2.4 – Схема бази даних

Таблиця Подія Event та Нещодавні дзвінки RecentCall мають зв'язок "багато-до-багатьох" за ключем data. Таблиця Вибрані контакти FavoriteContact та Нещодавні дзвінки RecentCall мають зв'язок "один-до-багатьох" за ключем phone.

2.3 Діаграми класів та компонентів

Діаграма класів – статичне представлення структури моделі та відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Всього було спроектовано 10 класів Контролерів, 2 класи Моделі, 11 допоміжних класів. На рисунках 2.5 – 2.13 бачимо спроектовані діаграми класів.

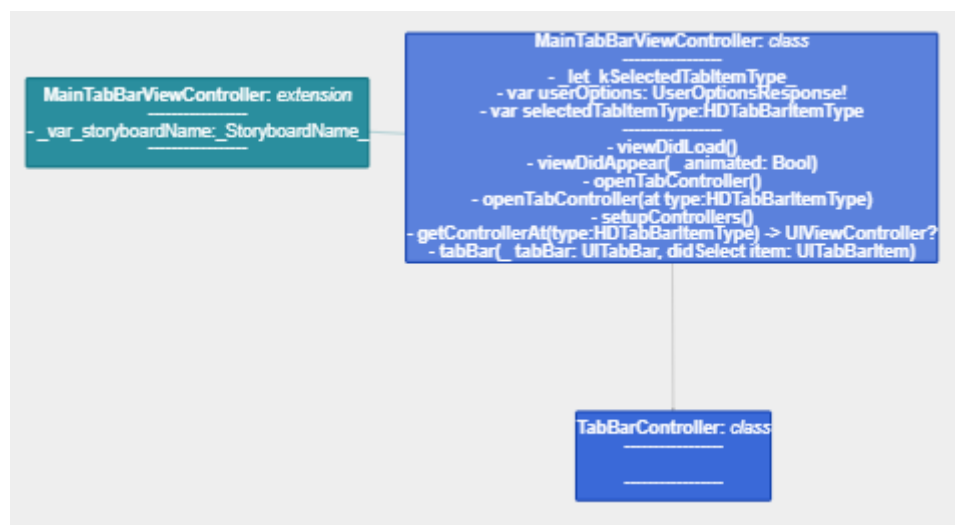


Рисунок 2.5 – Діаграма класів MainTabBarController

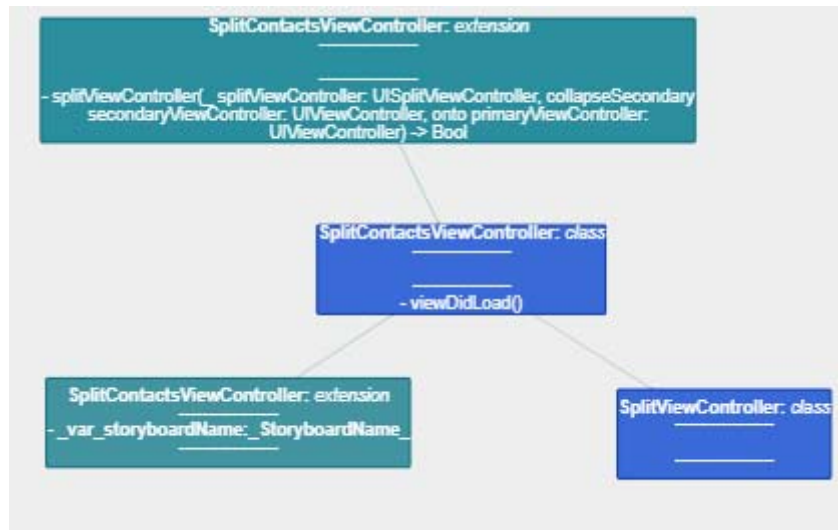


Рисунок 2.6 – Діаграма класів SplitContactsNavigationController

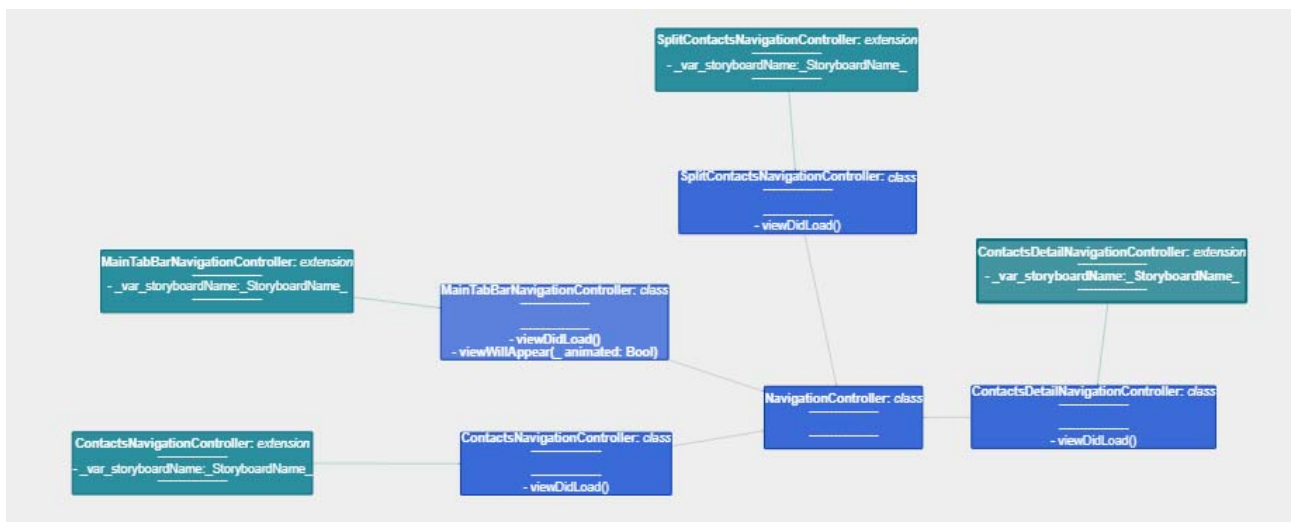


Рисунок 2.7 – Діаграма класів UINavigationController

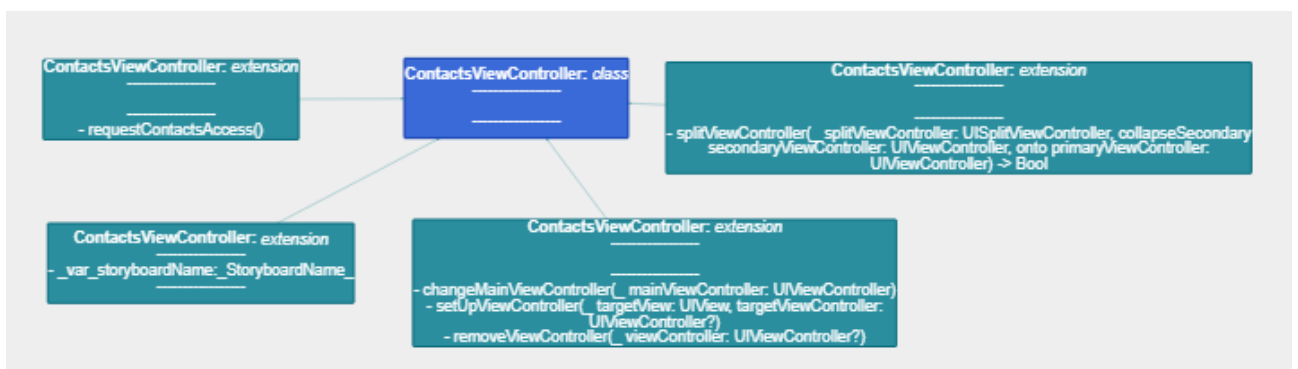


Рисунок 2.8 – Діаграма класів ContactsViewController

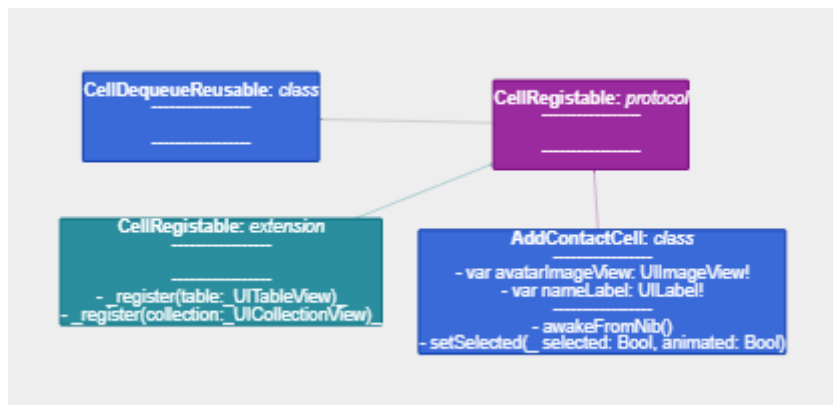


Рисунок 2.9 – Діаграма класів AddContactCell

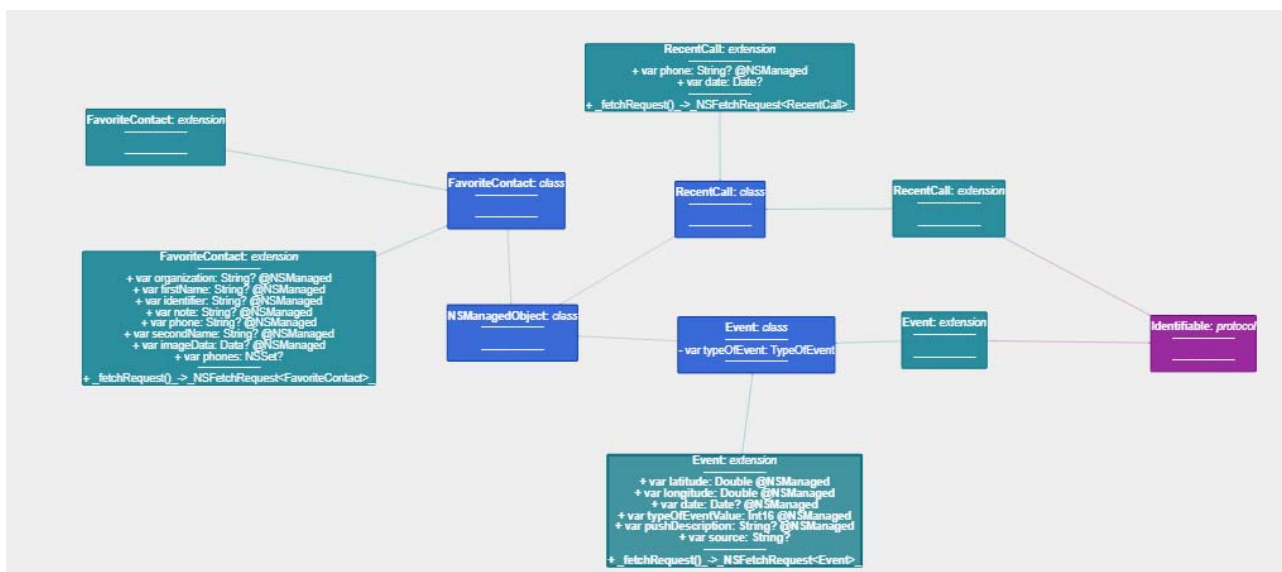


Рисунок 2.10 – Діаграма класів Event

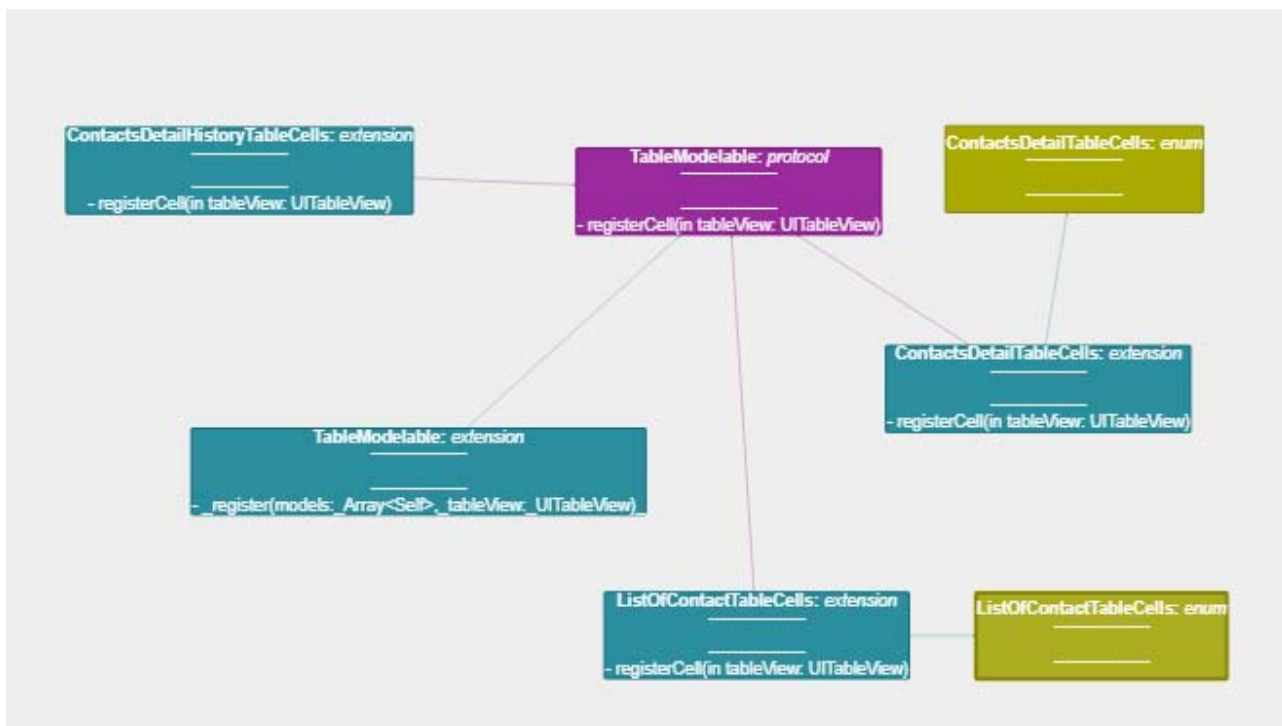


Рисунок 2.11 – Діаграма класів Contacts

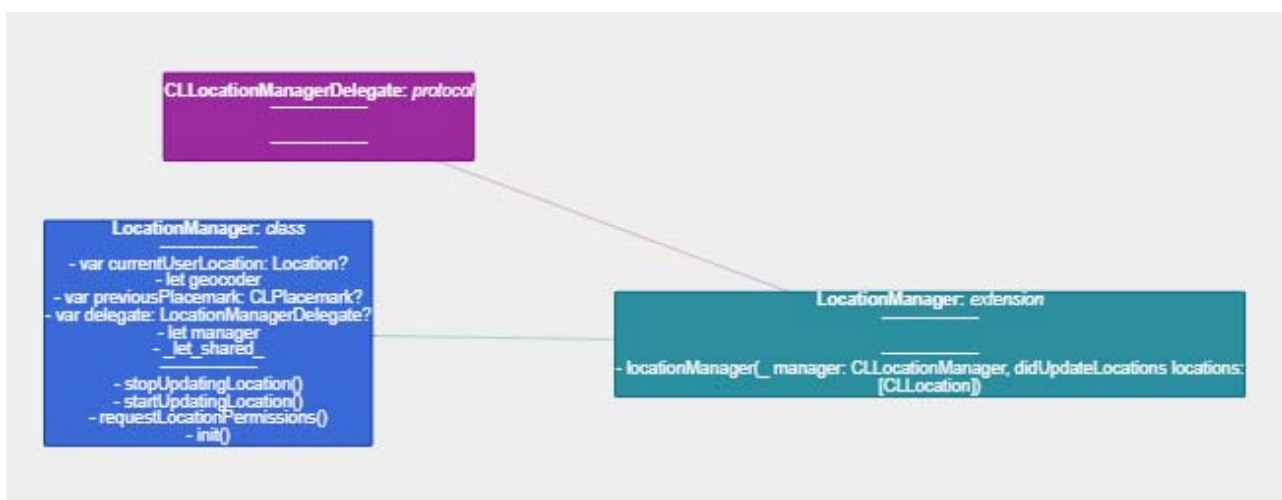


Рисунок 2.12 – Діаграма класів LocationManager

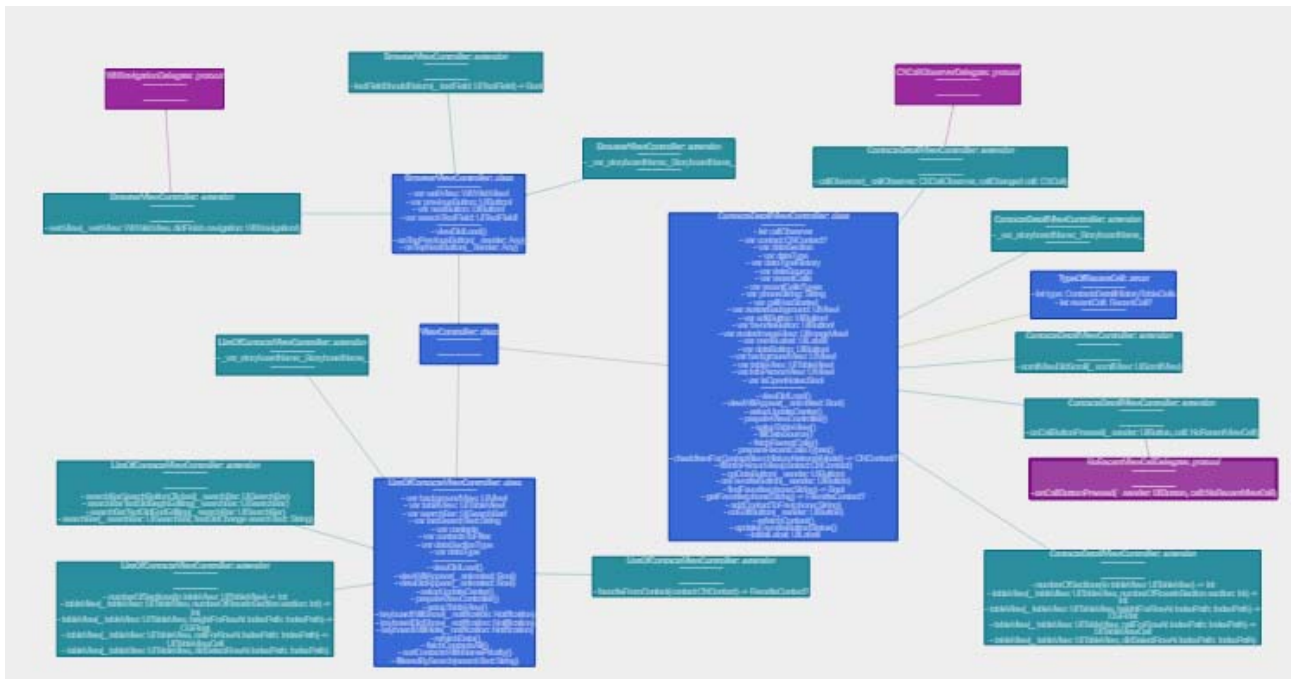


Рисунок 2.13 – Діаграма класів ContactsDetailViewController

Серед вищенаведених діаграм класів можна виділити наступні діаграму класів Event, Contacts та ContactsDetailViewController.

Діаграма класів Event описує взаємодію між класами RecentCall, FavoriteContact та Event. Діаграма класів Contacts наводить зв'язки між класами Contact та ListOfContactTableCells.

ContactsDetailViewController показує основну взаємодію між класами PredictionManager та контролерами ContactsDetailNavigationController, ContactsDetailViewController, ContactsViewController, ContactsNavigationController, ListOfContactsViewController.

Компоненти – це фізичні модулі програмного коду. Вони показують, яким образом одні компоненти взаємодіють з іншими.

На діаграмі відображаються як бібліотеки вихідного коду, так і файли, що виконуються. Для їхнього представлення використовуються різні піктограми (рисунок 2.14).

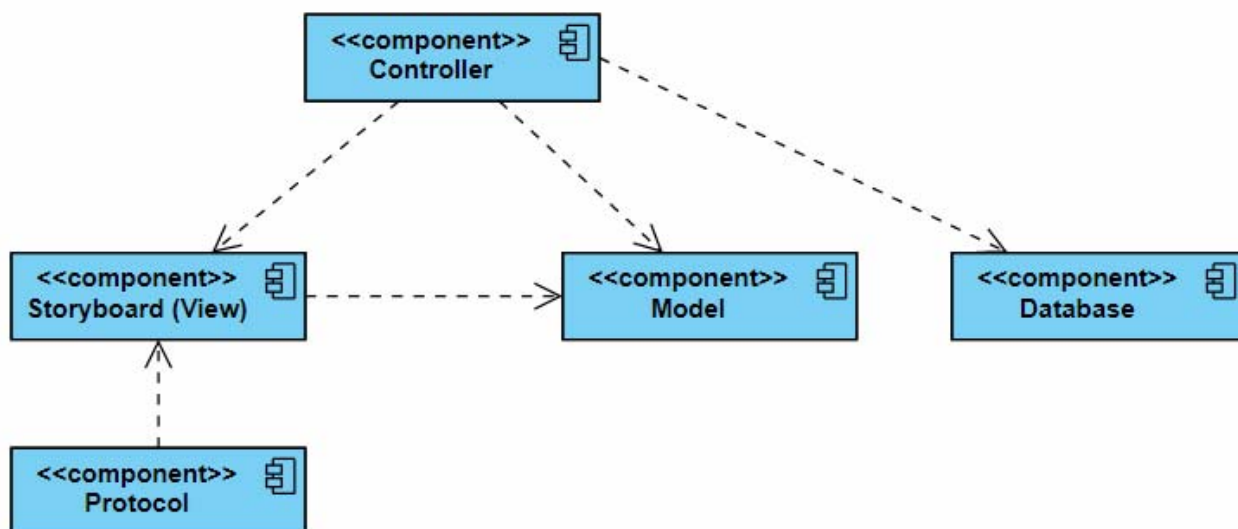


Рисунок 2.14 – Діаграма компонентів

2.4 Вибір мови програмування для розробки мобільного застосунку

Розглянемо наступні мови програмування, що використовуються для розробки мобільних застосунків: Java, Dart та Swift.

Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією Sun Microsystems як основний компонент платформи Java. З 2009 року мовою займається компанія Oracle, яка того року придбала Sun Microsystems [12].

В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Oracle надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License [12]. Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині.

Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування [12]-[13].

Dart – мова програмування, яку розробляє компанія Google, позиціонуючи як мову структурованого програмування [15].

Розробники вважали, що в довгостроковій перспективі Dart може стати прогресивною заміною JavaScript, котрий потерпає від наявних в даний час проблем з розширюваністю, продуктивністю і підтримкою розробки складних застосунків [15].

Мова має схожий на Java синтаксис, не вимагає явного визначення типів і її можна використовувати для створення серверних та клієнтських застосунків.

У березні 2015 компанія Google представила оновлену стратегію просування Dart, у котрій вирішено не прив'язувати Dart до браузеру і відмовитися від ідеї інтеграції віртуальної машини Dart у Chrome [15].

Розробку буде зосереджено на застосуванні Dart як проміжної мови, скомпільованої в JavaScript. Розвиток Dart як окремої мови, альтернативної JavaScript і безпосередньо підтримуваної у браузерах, визнано недоцільним.

Замість цього Dart рухатиметься у бік якіснішої інтеграції з JavaScript і генерації оптимального JavaScript-коду [16].

При цьому розробку віртуальної машини Dart VM буде продовжено, але вона позиціонуватиметься в основному для створення серверних і мобільних застосунків.

Особливості мови Dart:

– звичний і простий для вивчення синтаксис, природний для програмістів на JavaScript, C і Java;

- забезпечення швидкого запуску і високої продуктивності для всіх сучасних веб-браузерів і різних типів оточень, від портативних пристроїв до потужних серверів;

- можливість визначення класів і інтерфейсів, що дозволяють використовувати інкапсуляцію і повторно використовувати існуючі методи і дані;

- необов'язкове вказування типів, використовувати чи ні статичні типи вирішує розробник. Вказування типів дозволяє спростити зневадження і виявлення помилок, робить код яснішим і читаним, спрощує його доопрацювання та аналіз сторонніми розробниками;

- серед підтримуваних типів: різні види хешів, масивів і списків, черги, числові і рядкові типи, типи для визначення дати і часу, регулярні вирази (RegExp). Можливо створення своїх типів;

- для організації паралельного виконання пропонується використовувати класи з атрибутом `isolate`, код яких виконується повністю в ізольованому просторі в окремій області пам'яті, взаємодіючи з основним процесом через відправку повідомлень;

- підтримка використання бібліотек, що спрощують підтримку і зневадження великих вебпроектів. Сторонні реалізації функцій можуть підключатися у вигляді поділюваних бібліотек. Застосунки можна розбити на частини і доручити розробку кожної з частин окремій команді програмістів;

- набір готових інструментів для підтримки розробки мовою Dart, включаючи реалізацію засобів динамічної розробки та зневадження з виправленням коду на льоту ("edit-and-continue");

- можливість створювати однорідні системи, що охоплюють як клієнтську, так і серверну частину. Використання однієї мови та інструментарію для клієнтських і серверних компонентів спрощує процес кодування і позбавляє від постійної зміни контексту [17].

Swift – багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C та бути

стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta.

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду [18].

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C.

Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Мова також пропонує безліч сучасних методів програмування, таких як замикання, узагальнене програмування, лямбда-вирази, кортежі і словникові типи, швидкі операції над колекціями, елементи функційного програмування.

Основним застосуванням Swift є розробка користувацьких застосунків для macOS, iOS, tvOS, watchOS з використанням тулкіта Cocoa і Cocoa Touch. При цьому Swift надає об'єктну модель, сумісну з Objective-C.

Сирцевий код мовою Swift може змішуватися з кодом на C і Objective-C в одному проекті. Swift щільно інтегровано до власницького середовища розробки Xcode, проте може бути викликано з терміналу, що уможлиблює її використання на операційних системах, відмінних від macOS, наприклад, на Linux. Окремо варто відзначити, що Swift від компанії Apple не варто плутати з досить давно розроблюваною скриптовою мовою Swift, націленої на багатонитеве програмування і поставленого під вільною ліцензією Apache.

У таблиці 2.4 наведено порівняння мов програмування.

Таблиця 2.4 – Порівняльна характеристика мов програмування

	Java	Dart	Swift
Передбачуване використання	Мобільні застосунки, бізнес, вебпрограми, клієнтська та серверна сторони	Мобільні застосунки, вебпрограми	Мобільні застосунки, десктоп
Імперативне	+	+	+
Об'єктно-орієнтоване	+	+	+
Функціональне	—	—	—
Процедурне	—	—	—
Узагальнене	+	—	+
Рефлексивне	+	—	+
Подійно-орієнтоване	—	—	—
Інші парадигми	Конкурентне	Структурне	Конкурентне
Стандартизація	Java Language Specification	Ні	Ні

Проведено дослідження мов програмування, що використовуються для розробки мобільних застосунків: Java, Dart та Swift. Було обрано мову програмування Swift як найпоширеніший інструмент для розробки iOS застосунків.

2.5 Вибір середовища розробки мобільного застосунку

Для розробки під ОС iOS є три основні Integrated Development Environment (IDE): Xcode, Appcode та CodeRunner 2 [19]-[23].

Xcode – одна з найкращих IDE для розробки застосунків під iOS, яка має автоматичне завершення та повне виділення синтаксису для Swift. IDE інтегрована з фреймворками Cocoa Touch та має кнопку помічника, яка розділяє редактори на основний робочий документ і помічник редактора. IDE має вбудований конструктор інтерфейсу, який спрощує створення прототипу повного інтерфейсу користувача без визначення коду. Xcode містить всі необхідні інструменти, щоб створити програми для iPhone, iPad, Mac, Apple Watch і Apple TV. Xcode 8 – швидша версія IDE, яка має нові розширення редактора, які можна просто використовувати, щоб повністю налаштувати [19].

iOS накладає своєрідну «монополію» на вибір засобів для розробника. Тим не менш, Xcode по праву займає перше місце серед інструментів для розробки мобільних застосунків. Xcode використовує єдине вікно робочого простору – workplace window, що містить більшу частину необхідних для роботи даних. З Xcode можна легко перейти від написання коду до його відлагодження та розробки користувацького інтерфейсу в межах одного робочого вікна.

SDK iOS постійно розширює комплект інструментів Xcode, додаючи компілятори та фреймворки, необхідні для роботи. В якості мови для розробки виступає Objective-C або Swift [20].

Appcode – це інтегроване середовище розробки для Swift, Objective-C, C та C++, розроблене на платформі IntelliJ IDEA від JetBrains [21]-[22]. Перша загальнодоступна попередня версія AppCode стала доступною у квітні 2011 року. AppCode побудований на платформі IntelliJ IDEA, яка написана на Java та Kotlin. Користувачі можуть розширити його можливості, встановивши надбудови для платформи IntelliJ, а також вони можуть написати свої власні надбудови. Appcode – розумне середовище розробки для iOS/macOS, що

забезпечує високу якість коду за рахунок ефективної навігації, інтелектуального автодоповнення та аналізу коду.

CodeRunner 2 – просунутий редактор коду [23], що дозволить легко редагувати вихідні тексти, але й запускати їх, використовуючи вбудовані в систему компілятори/інтерпретатори, гнучко налаштовувати робоче середовище під конкретну мову, а також створювати власні шаблони для певних типів файлів.

Основні можливості CodeRunner 2 наступні [23]:

- наявність повнофункціонального редактору з вкладками, підсвічуванням синтаксису, автозавершення коду, колірні теми, інтелектуальні дужки та ін.;
- пресети коду для нових документів, що редагуються, позбавляють від необхідності продовжувати писати основні функції та ін.;
- вбудований файловий браузер для швидкого доступу до файлів;
- підтримка введення в програмах, з можливістю запуску з зумовленими вхідними даними;
- автозбереження та повний екран;
- налаштовані команди запуску та прапори компіляції, шрифти, що налаштовуються, повна підтримка кодувань та ін.

CodeRunner підтримує біля 25 мов програмування, серед яких є і мови програмування Swift та Objective-C / Objective-C++ для мобільних застосунків.

В результаті дослідження IDE для мобільних застосунків було вирішено обрати середовище розробки Xcode.

2.6 Висновок до другого розділу

У даному розділі було розглянуто архітектурні шаблони проектування мобільного застосунку: Модель–Представлення–Контролер (Model View Controller, MVC), Модель–Представлення–Пред'явник (Model View Presenter,

MVP), Clean Swift або VIP (View controller, Interactor та Presenter) та Model View View Model.

Для реалізації мобільного застосунку прийнято рішення використати архітектурний шаблон MVC тому, що проєкт невеликого обсягу, даний архітектурний шаблон доцільно застосувати, модулі програми будуть логічно розділені на компоненти Модель, Представлення та Контролер.

Було виконано проектування схеми бази даних, діаграм класів та компонентів. Проведено дослідження мов програмування, що використовуються для розробки мобільних застосунків: Java, Dart та Swift. Було обрано мову програмування Swift як найпоширеніший інструмент для розробки iOS застосунків.

В результаті дослідження IDE для мобільних застосунків було вирішено обрати середовище розробки Xcode.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

Для реалізації мобільного застосунку у розділі 2 вирішено використати архітектурний шаблон MVC тому, що проєкт невеликого обсягу, даний архітектурний шаблон доцільно застосувати, модулі програми будуть логічно розділені на компоненти Модель, Представлення та Контроллер. У розділі 1 також прийнято рішення виконати інтеграцію у вигляді модуля з використанням спеціалізованих бібліотек машинного навчання TensorFlow та Numpy при розробці адаптивного інтерфейсу мобільного застосунку. Тому проведемо огляд найпоширеніших бібліотек машинного навчання під мобільну операційну систему iOS.

3.1 Огляд бібліотек машинного навчання

В роботі були використані наступні бібліотеки машинного навчання під мобільну операційну систему iOS: TensorFlowLiteSwift та Matft.

Бібліотека TensorFlowLiteSwift – аналог відкритої програмної бібліотеки для машинного навчання TensorFlow [4], [24], розробленої під мобільну операційну систему iOS з можливістю написання вихідного коду з використанням мови Swift. TensorFlow можна використовувати для навчання величезних моделей у кластерах у хмарі, а також для запуску моделей локально у вбудованій мобільній операційній системі смартфона.

Бібліотека Matft – бібліотека, подібна до бібліотеки Numpy з можливістю написання вихідного коду з використанням мови Swift [25]-[26]. Назва функцій та її використання подібні до бібліотеки Numpy, що надає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з цими масивами.

3.2 Огляд використаних фреймворків, рушіїв та допоміжних бібліотек

В роботі були використані наступні фреймворки та рушії: WebKit, UIKit, CallKit, CoreLocation, CoreData, UserNotifications, Contacts, SVProgressHUD.

WebKit – рушій виведення для вебсторінок [27]. На базі WebKit працюють відомі браузерери Apple Safari, Google Chrome (до версії 28) та ін. WebKit забезпечує набір класів для виведення веб-інформації у вікнах, та реалізує функції браузера, такі як: механізм гіперпосилань, управління історією відвіданих сторінок. Спочатку рушій був створений як гілка від KHTML та використовувався компанією Apple у своєму браузері Safari. WebKit портується на багато платформ, зокрема зараз він підтримується у Windows та Linux. Такі частини WebKit, як WebCore та JavaScriptCore доступні під ліцензією GNU LGPL, а інші під BSD-подібними ліцензіями.

Можна виділити наступні компоненти WebKit [27]:

- WebCore – це бібліотека, що відповідає за вивід, рендерінг, та об'єктну модель документу для HTML та SVG. Код бібліотеки виданий за під LGPL (GNU Lesser General Public License). WebKit «обгортає» WebCore і JavaScriptCore, які написані на C++, програмним інтерфейсом для Objective-C. WebKit проходить тесті Acid2 та Acid3, з відмінним рендерінгом на рекомендованому обладнанні;

- JavaScriptCore JavaScriptCore – це фреймворк, що реалізує JavaScript-рушій для WebKit. JavaScriptCore був створений на базі KJS та (бібліотека для роботи з регулярними виразами). 2 червня 2008 був анонсований переписаний JavaScriptCore під назвою «SquirrelFish», інтерпретатор байткоду. Цей проєкт еволюціонував до SquirrelFish Extreme (або просто SFX), анонсований 18 вересня 2008. SFX компілює JavaScript у нативний машинний код, не потребуючи інтерпретатору і таким чином прискорюючий виконання JavaScript;

– Drosera – це відлагоджувач для JavaScript, який входить до складу нічних збірок WebKit;

– SunSpider – програмний засіб, що вимірює продуктивність JavaScript на типових запитах, як маніпуляції з текстом та шифрування.

UIKit – фреймворк, призначений для розробки користувальницьких інтерфейсів (кнопки, картинки, слайдери, текстові поля та ін.) [28],[29]. З допомогою нього, за досить короткий час, можна сформувати досить складну веб-сторінку і причому всі елементи, що входять у її структуру, будуть витримані в одному єдиному стилі, що іноді дуже важливо. Фреймворк відносно молодий, так як був випущений в липні 2013 року компанією Yootheme і вже користується гарною популярністю серед веб-розробників. Так само він володіє хорошим швидкодією і за своєю структурою складається з окремих модулів, що значно спрощує роботу і налаштування під себе. Звичайно ж, так як це сучасний фреймворк, у ньому підтримується адаптивність, тобто при необхідності можна реалізувати адаптивний інтерфейс. Так як UIKit – це більше CSS фреймворк, то звичайно ж, в основному працювати доведеться з певними стандартними іменами класів, але вони обрані так вдало, що запам'ятовуються практично миттєво і інтуїтивно зрозумілі. Ну і останнє компоненти входять до складу фреймворку, можуть працювати як окремо, так і один з одним, як би в парі, тобто Ви завжди зможете комбінувати функціонал окремих компонентів.

UIKit є повністю безкоштовним для всіх ваших особистих або комерційних проектів[28],[29].

Основні можливості UIKit[28], [29]:

- розширення та модульні компоненти;
- послідовні і безконфліктні іменування;
- адаптивний та мобільний інтерфейс;
- підтримка LESS;
- розширення темами;
- зміна в режимі реального часу;

- адаптивна, гумова та вкладена сітка;
- Off-canvas navigations;
- навігація, меню, що випадають, модулі, кнопки та ін.;
- підтримка RTL "з коробки";
- наявність повної документації.

CallKit – фреймворк, що дозволяє відобразити інтерфейс системного виклику для служб VoIP застосунку та координувати служби дзвінків з іншими програмами та системою [30], та метою якого є покращення взаємодії з VoIP, що дозволяє інтегрувати застосунки з власним інтерфейсом користувача смартфона. Застосування CallKit дозволяє використовувати власний екран вхідних викликів як у заблокованому, так і в розблокованому стані, запускати виклики з екранів «Контакти», «Вибране» та «Недавні» застосунків власного телефону, взаємодіяти з іншими викликами в системі [307].

Бібліотека CoreLocation дозволяє визначити поточну позицію або напрям, асоційований з пристроєм [31]. Дана бібліотека використовує доступне обладнання для визначення позиції та напрямку користувача. Можна використовувати класи та протоколи з цієї бібліотеки, щоб налаштувати та встановити графік для визначення розташування та подій про переміщення. Також можна використовувати даний графік, щоб визначити географічні регіони та відстежувати, коли користувач перетинає межі цих регіонів. У iOS, можна визначити область навколо Bluetooth маячку.

CoreData – фреймворк від компанії Apple, вбудований в операційну систему iOS, MacOS, що дозволяє розробнику взаємодіяти з базою даних [32]. Був представлений компанією Apple з анонсом Mac OS X 10.4 Tiger та iOS з iPhone SDK 3.0. Дозволяє цим бути організованими в Сутність-Атрибут-Значення. Управління даними можна здійснити з допомогою маніпуляцій сутностей та їх взаємозв'язків. Core Data взаємодіє безпосередньо з SQLite, ізолюючи розробника від базового SQL.

CoreData складається з наступних компонент [32]:

- Managed Object Context – компонент для збереження або перезапис даних;

- Persistent Store Coordinator – компонент для зберігання даних;

- Managed Object Model – модель бази даних;

- Persistent Store – репозиторій, де зберігаються дані.

SVProgressHUD – простий у використанні Head-Up Display, призначений для відображення прогресу поточного завдання на iOS і tvOS [33].

Foundation – фреймворк, що надає базовий рівень функціональності для застосунків та платформ, включаючи зберігання і збереження даних, обробку тексту, обчислення дати та часу, сортування та фільтрацію, а також роботу в мережі [34]. Класи, протоколи та типи даних, визначені в Foundation, використовуються в SDK macOS, iOS, watchOS та tvOS [34].

Contacts – фреймворк, що надає доступ до контактів користувача, форматування та локалізацію контактної інформації з використанням API Swift і Objective-C для доступу до контактної інформації користувача [35]. На даний час більшість програм зчитують контактну інформацію, не вносячи жодних змін, ця структура оптимізована для потокобезпечного використання лише для читання.

UserNotifications – фреймворк, що призначений для надсилання сповіщення для користувача на пристрій користувача із сервера або створення їх локально зі свого застосунка. Фреймворк надає набір API та уніфікований, об'єктно-орієнтований метод роботи з локальними та віддаленими повідомленнями на платформах iOS 10, tvOS 10 та watchOS 3, що дозволяє, порівняно з існуючим API, обробляти локальні або видалені повідомлення можна дуже схожим чином, а доступ до контенту сповіщень більше не робиться через словники [36].

3.3 Опис розроблених класів

Мобільний застосунок складається з головного файлу AppDelegate.swift проекту, що викликається при старті програми, у ньому відбуваються всі ініціалізації проекту, розроблених класів моделей, контролерів, допоміжних класів та файлів розмітки представлення.

3.3.1 Опис розроблених моделей

У розділі 2 було прийнято рішення щодо використання архітектури MVC, оскільки дана архітектура відповідає всім вимогам, що висуваються до архітектури мобільного застосунку.

Модель являє собою клас для визначення даних, які будуть відображатися або над якими будуть проводитися інші дії у інтерфейсі користувача.

Отже, до Моделі відносяться наступні класи:

- ContactDetailViewModel – модель відображення деталей контакту;
- RecentCallViewModel – модель відображення недавніх дзвінків.

3.3.2 Опис розроблених контролерів

Контролер відстежує зміни пов'язані з діями користувача та відповідно цим змінам він впливає на Модель. Були розроблені наступні класи контролерів:

- BrowserViewController – контролер, що призначений для перегляду сторінок вебсайтів(браузера);
- ContactsDetailNavigationController – контролер, що призначений для створення навігації в деталях контакту;
- ContactsDetailViewController – контролер, що призначений для відображення деталей контакту;

- `ContactsNavigationController` – контролер, що призначений для створення навігації на списку контактів;
- `ContactsViewController` – контролер, що призначений для перегляду контактів;
- `ListOfContactsViewController` – контролер, що призначений для перегляду списку контактів;
- `MainTabBarController` – контролер, що призначений для створення навігації в панелі вкладок;
- `MainTabBarViewController` – контролер, що призначений для створення в панелі вкладок;
- `SplitContactsNavigationController` – контролер, що призначений для навігації перегляду списку контактів та вибраних контактів;
- `SplitContactsViewController` – контролер, що призначений для перегляду списку контактів та вибраних контактів (служить як контейнер для них).

3.3.3 Опис розроблених допоміжних класів та файлів представлень

Також були розроблені наступні допоміжні класи та файли представлень:

- `AddContactCell.swift`, `AddContactCell.xib` – клас та файл розмітки представлення додавання контакту;
- `BaseTableViewCell.swift` – клас базового представлення чарунок таблиці;
- `CellRegistable.swift` – протокол реєстрації чарунок в таблицях;
- `CNContactExt.swift` – файл розширення класу `CNContact`, доданий метод сортування;
- `ContactManager.swift` – клас, у якому відбуваються всі маніпуляції з контактами(видалення, редагування та ін.);
- `ContactsDetailHistoryTableCells.swift` – чарунка деталей контакту;

- ContactsDetailTableCells.swift – клас чарунок таблиці деталей контакту;
- Event+CoreDataClass.swift – модель бази даних, подія;
- Event+CoreDataProperties.swift – модель бази даних, властивості події;
- FavoriteContact+CoreDataClass.swift – модель бази даних, вибраний контакт;
- FavoriteContact+CoreDataProperties.swift – модель бази даних, властивості вибранного контакту;
- ListOfContactCell.swift, ListOfContactCell.xib – клас та файл розмітки представлення чарунки контакту;
- ListOfContactMarkedCell.swift, ListOfContactMarkedCell.xib – клас та файл розмітки представлення чарунки контакту, що показується при пошуку, з підсвічуванням збігів;
- ListOfContactTableCells.swift – розширення для чарунок контакту, що показується при пошуку, з підсвічуванням збігів;
- LocationManager.swift – клас, який відповідає за локацію користувача;
- NoRecentViewCell.swift, NoRecentViewCell.xib – клас та файл розмітки представлення чарунки, яка відображається, коли немає попередніх дзвінків;
- NoteContactCell.swift, NoteContactCell.xib – клас та файл розмітки представлення чарунки заміток в контакті;
- NotificationManager.swift – клас, що відповідає за push нотифікації;
- PhoneContactCell.swift, PhoneContactCell.xib – клас та файл розмітки представлення введення телефону під час додавання/зміни контакту;
- RecentCall+CoreDataClass.swift – модель бази даних, попередні дзвінки;
- RecentCall+CoreDataProperties.swift – модель бази даних, попередні дзвінки (властивості);
- RecentSeparatorCell.swift, RecentSeparatorCell.xib – клас та файл розмітки представлення попередніх дзвінків;
- TableModelable.swift – протокол взаємодії з таблицями, для спрощення реєстрації чарунків у таблицях;

– StoryboardInstance.swift – протокол для роутингу між представленнями контролерів;

– StoryboardName.swift – протокол для роутингу між представленнями контролерів, в якому зберігаються назви усіх файлів розміток для ініціалізації / прототипування інтерфейсу користувача.

– PredictionManager.swift – клас, в якому присутні методи, що відповідають за передбачення подій.

Наведемо опис основних властивостей та функцій класу PredictionManager (рис. 3.1).

```
PredictionManager: class
-----
- _let_shared: PredictionManager_
  - var events : [Event]
  - var normalizedData: [[String]]
  - var outputs: [[Float32]]
-----
- predicate(location: Location)
  - normalizeDataSet()
- normalize(from latitude: Double, longitude: Double, date: Date) ->
  [String]
  - runScript(inputData: Data) -> [Float32]
- kNN(currentFeatures: [Float32], historyFeatures: [[Float32]], historyOutputs:
  [Event], k: Int)
  - init()
```

Рисунок 3.1 – Клас PredictionManager

Структура Prediction містить поля event (подія) та distance (відстань). Масив events містить перелік подій. Масив normalizedData необхідний для нормалізації даних. Масив outputs для отримання вихідного результату. Далі приведемо текст функції predicate, що дозволяє здійснити передбачення з використанням методу k-найближчих сусідів та нейронної мережі.

```
func predicate(location: Location) {
    normalizedData = []
    outputs = []

    normalizeDataSet()

    let k = 2
```

```

        let normalizedCurrentEvent = normalize(from: location
        .latitude, longitude: location.longitude, date: Date())

        let currentEventInputDataFloat32 = normalizedCurrentE
vent.compactMap({ Float32($0) })

        let currentEventInputData = currentEventInputDataFlea
t32.withUnsafeBytes { f in
            Data(f)
        }

        let currentEvent = runScript(inputData: currentEventI
nputData)

        for value in normalizedData {
            let inputDataFloat32 = value.compactMap({ Float32
($0) })

            let inputData = inputDataFloat32.withUnsafeBytes
{ f in
                Data(f)
            }
            self.outputs.append(runScript(inputData: inputDat
a))
        }

        let historyComponents = Array(outputs.reversed())
        let events = Array(self.events.reversed())
        kNN(currentFeatures: currentEvent, historyFeatures: h
istoryComponents, historyOutputs: events, k: k)
    }
}

```

Функції `normalizeDataSet` та `normalize` призначені для нормалізації даних. Функція `runScript` запускає роботу побудованої моделі нейронної мережі(кодувальник). Функція `kNN` релізує метод k-найближчих сусідів.

3.4 Опис інтерфейсної частини

Для формування інтерфейсної частини застосунку використовуються розмітки представлення, що розміщені у відповідних файлах тексту програми.

Представлення – компонент мобільного застосунку, що видає екран (View), та з яким користувачем користувачі можуть взаємодіяти для виконання яких-або діючих дій (наприклад, реєстрація та авторизація користувача).

- AddEdit.storyboard – файл розмітки у форматі XML для ініціалізації / прототипування інтерфейсу користувача для додавання/зміни контакту;
- Browser.storyboard – файл розмітки у форматі XML для ініціалізації / прототипування інтерфейсу користувача для перегляду сайтів(браузера) ;
- ContactDetail.storyboard – файл розмітки у форматі XML для ініціалізації / прототипування інтерфейсу користувача для перегляду деталей контакту;
- ListOfContacts.storyboard – файл розмітки у форматі XML для ініціалізації / прототипування інтерфейсу користувача для перегляду списку контактів;
- ListOfFavorites.storyboard – файл розмітки у форматі XML для ініціалізації / прототипування інтерфейсу користувача для перегляду вибраних контактів;
- SplitContacts.storyboard – файл розмітки у форматі XML для ініціалізації / прототипування інтерфейсу користувача для перегляду списку контактів та вибраних контактів.

3.5 Висновок до третього розділу

В даному розділі виконано огляд використаних бібліотек машинного навчання, фреймворків, рушіїв та допоміжних бібліотек, було описано розроблені класи моделей ContactDetailViewModel та RecentCallViewModel, класу PredictionManager.swift, в якому присутні методи, що відповідають за передбачення подій та класи контролерів:

- BrowserViewController;
- ContactsDetailNavigationController;
- ContactsDetailViewController;
- ContactsNavigationController;
- ContactsViewController;
- ListOfContactsViewController;

- MainTabBarNavigationController;
- MainTabBarViewController;
- SplitContactsNavigationController;
- SplitContactsViewController.

Також виконано опис інтерфейсної частини мобільного застосунку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Функціональні можливості системи

Розроблений мобільний застосунок вирішує задачу передбачення подій або релевантних дій для користувача.

Розроблена система має наступні функціональні можливості:

- збирає дані користувача, такі як: локація, дата, час;
- збирає події, в даному випадку дзвінки та історію браузера;
- при зміні локації (не координат, а саме локації, наприклад, при зміні супермаркету Сільпо на супермаркет АТБ) відповідна локація та дата її зміни буде оброблена моделлю штучного інтелекту з використанням бібліотеки TensorFlow та за допомогою алгоритму буде запропоновано найбільш релевантну дію, пов'язану з вихідними даними.

4.2 Порівняльний аналіз функціональних можливостей системи з відомими аналогами

З відомих аналогів, що реалізують алгоритми штучного інтелекту, можна відзначити такі програмні продукти:

- FineScanner AI;
- Hemingway App;
- Replika.

Порівняння функціональних можливостей існуючих аналогів систем, що реалізують підходи інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів та розробленої мобільного застосунку наведено в таблиці 4.1.

Таблиця 4.1 – Порівняльний аналіз систем, що реалізують підходи інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів

Системи\ Параметри	FineScanner AI	Hemingway App	Replika	Розроблений мобільний застосунок
Зручність та простота інтерфейсу користувача	+	+	+	+
Адаптивність інтерфейсу	—	—	—	+
Наявність push сповіщень	—	—	—	+
Робота в режимі реального часу	+	+	+	+
Збирання даних про користувача: локація, дата, час	—	—	—	+
Збирання подій	—	—	—	+
Запропонування релевантної дії, пов'язану з вихідними даними	—	+	—	+

На основі проведеного аналізу можна зробити висновок, що розроблена система, на відміну від існуючих аналогів, дозволяє визначати релевантну дію користувача, пов'язану з вихідними даними (зміна локації та часу, дня тижня).

4.3 Тестування компонентів розробленого мобільного застосунку

З метою відстеження коректності роботи застосунку та базою даних, відстеження обробки помилкових дій користувача, обробки стандартних дій користувача (навігація застосунком та різні етапи життєвого циклу вьюконтролерів) було проведено декілька емуляцій роботи користувача з системою.

В ході таких емуляцій для оцінки якості та стабільності застосунку були проведені наступні перевірки:

- перевірка на завантаження застосунку;
- перевірка на адаптивність інтерфейсу;
- перевірка на відновлення застосунку після згортання;
- перевірка на переходи між вкладками;
- перевірка на повернення до попереднього вью контролеру.
- перевірка на додавання контакту;
- перевірка на редагування контакту;
- перевірка на видалення контакту;
- перевірка на додавання контакту в обране;
- перевірка на видалення контакту з обраного;
- перевірка роботи пошуку контактів (рис. 4.1);
- перевірка роботи браузера;
- перевірка на переходи між вкладками в браузері (рис. 4.2);
- перевірка роботи передбачень;
- перевірка роботи push сповіщень (рис. 4.3, рис. 4.4).

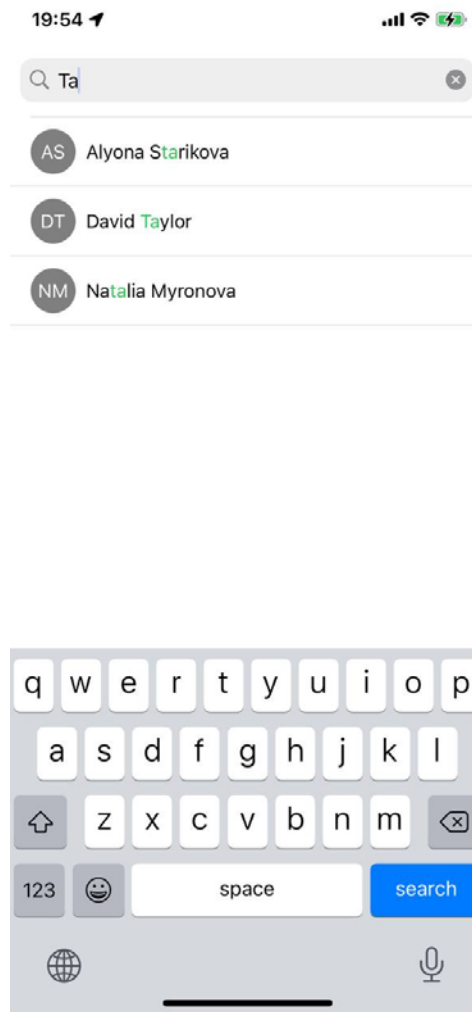


Рисунок 4.1 – Перевірка роботи пошуку контактів

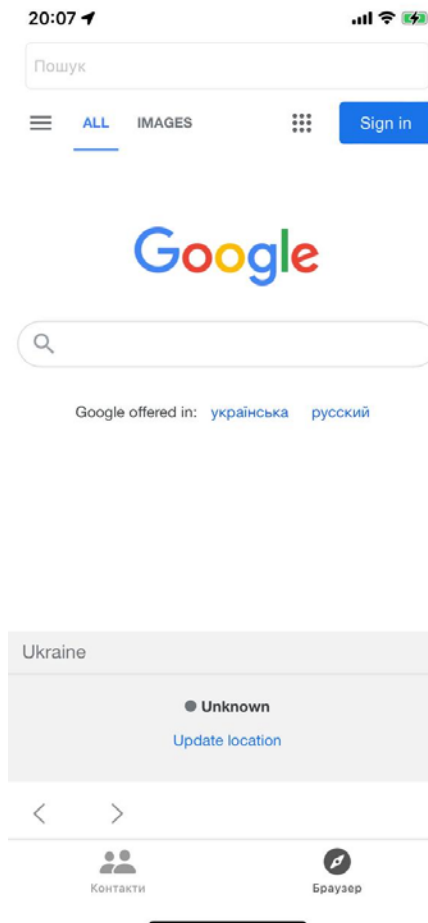


Рисунок 4.2 – Перевірка на переходи між вкладками в браузері

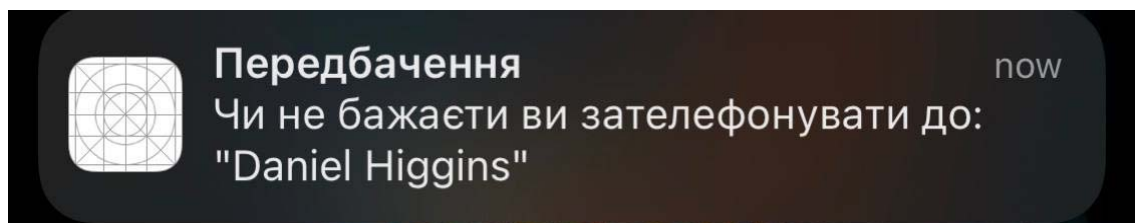


Рисунок 4.3 – Приклад 1 перевірки роботи push сповіщень

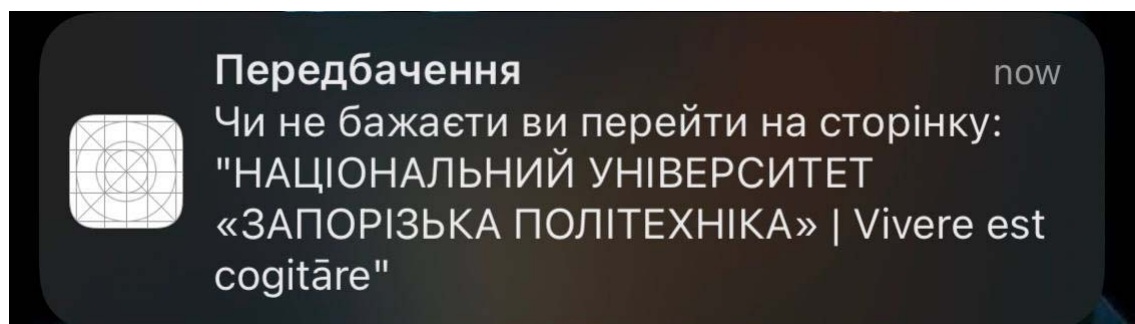


Рисунок 4.4 – Приклад 2 перевірки роботи push сповіщень

Детальний опис змісту та сенсу перевірок наведено нижче.

Перевірка на завантаження застосунку передбачає як успішність завантаження, так і допустимий час, за який виконується завантаження.

Перевірка на адаптивність інтерфейсу передбачає коректність відображення елементів керування в умовах різної орієнтації пристрою(книжна, альбомна) та за різних параметрів пристрою(розмір екрану).

Перевірка на відновлення застосунку після згортання полягає в можливості відновлення застосунку з того місця, на якому була завершена робота користувача перед згортанням.

Перевірка на переходи між вкладками полягає в перевірці часу, який необхідний застосунку для зміни та відрисовки поточного व्यюконтролера на наступне.

Перевірка на повернення до попереднього व्यюконтролера полягає в можливості відновлення роботи з попереднім व्यюконтролером при натисканні на кнопку «Back».

Перевірка на додавання контакта полягає в можливості додавання контакта при натисканні на кнопку «Зберегти».

Перевірка на редагування контакту полягає в можливості редагування контакта при натисканні на кнопку «Зберегти».

Перевірка на видалення контакту полягає в можливості видалення контакта при натисканні на кнопку «Видалити».

Перевірка на додавання контакту в обране полягає в можливості додавання контакту в обране при натисканні на кнопку «Додати в обране» (Кнопка за картинкою зірочки).

Перевірка на видалення контакту з обраного полягає в можливості видалення контакту з обраного при натисканні на кнопку «Видалити з обраного» (Кнопка за картинкою зірочки).

Перевірка на роботи пошуку контактів полягає в можливості шукати контакти на вкладці «Контакти», та бачити підсвічені співпадіння.

Перевірка на роботу браузера полягає в можливості перегляду сторінок після переходу на вкладку «Браузер».

Перевірка на переходи між вкладками в браузері полягає в можливості перегляду сторінок після переходу на вкладку «Браузер».

Перевірка на роботу передбачень полягає в перегляді передбачень в конкретній локації та в конкретний час та день тижня.

Перевірка роботи push сповіщень полягає в перегляді push сповіщень під час виникнення передбачення.

Тестування проводилось на трьох різних мобільних пристроях, основні характеристики яких наведено нижче:

- модель – iPhone 11 Pro Max, процесор – Apple A13 Bionic;
- модель – iPhone X, процесор – Apple A11 Bionic;
- модель – iPhone 8, процесор – Apple A8.

Нижче в таблиці 4.2 наведено чек-лист тестування розробленого мобільного застосунку.

Таблиця 4.2 – Чек-лист тестування мобільного застосунку

Перевірка	Модель – iPhone 11 Pro Max, процесор – Apple A13 Bionic	Модель – iPhone X, процесор – Apple A11 Bionic	Модель – iPhone 8, процесор – Apple A8
1	2	3	4
Завантаження застосунку	пройдено	пройдено	пройдено
Адаптивність інтерфейсу	пройдено	пройдено	пройдено
Відновлення застосунку після згортання	пройдено	пройдено	пройдено
Переходи між вкладками	пройдено	пройдено	пройдено

Продовження таблиці 4.2

1	2	3	4
Повернення до попереднього контролю	пройдено	пройдено	пройдено
Додавання контакту	пройдено	пройдено	пройдено
Редагування контакту	пройдено	пройдено	пройдено
Видалення контакту	пройдено	пройдено	пройдено
Додавання контакту в обране	пройдено	пройдено	пройдено
Видалення контакту з обраного	пройдено	пройдено	пройдено
Пошук контактів	пройдено	пройдено	пройдено
Робота браузерa	пройдено	пройдено	пройдено
Переходи між вкладками в браузері	пройдено	пройдено	пройдено
Передбачення	пройдено	пройдено	пройдено
Push сповіщення	пройдено	пройдено	пройдено

Завантаження застосунку відбувається в середньому за 0,2-0,5с., що є прийнятним результатом. Відновлення застосунку після згортання передбачає старт застосунку з користувацькими налаштуваннями – відкрита відповідна вкладка. Перевірка на відновлення застосунку після згортання пройдена успішно, оскільки після згортання застосунок відновлює відповідну вкладку, яка була відкрита. Переходи між вкладками виконуються за 0,2-0,6с., цей час вважається прийнятним. При відкритому діалоговому вікні натискання на кнопку «Back» повертає управління попередньому контролеру та повністю відновлює його роботу з моменту передачі управління діалоговому вікну. Інтерфейс є адаптивним, оскільки елементи керування застосунку пристосовуються як під орієнтацію мобільного пристрою, так і під розміри екрану.

4.4 Висновок до четвертого розділу

В даному розділі наведені функціональні можливості системи. Також виконано опис процесу тестування компонентів розробленої системи.

Подальші напрямки розвитку мобільного застосунку полягають в додаванні більшої кількості подій для обробки, наприклад відправлення повідомлення, проведення якихось транзакцій, а також в виконанні портування застосунку на MacOS18:11.

ВИСНОВКИ

В результаті виконання дипломної кваліфікаційної роботи магістра розглянуто поняття адаптивного користувацького інтерфейсу, виконано огляд підходів щодо розробки адаптивних інтерфейсів та дослідження підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів.

Було прийнято рішення виконати інтеграцію у вигляді модуля з використанням спеціалізованих бібліотек машинного навчання TensorFlow та NumPy при розробці адаптивного інтерфейсу мобільного застосунку.

Було розглянуто архітектурні шаблони проектування мобільного застосунку: Модель–представлення–контролер (Model View Controller, MVC)–Модель–Представлення–Пред'явник (Model View Presenter, MVP), Clean Swift або VIP (View controller, Interactor та Presenter) та Model View ViewModel. Було виконано проектування структури бази даних, діаграм класів та компонентів.

Для реалізації мобільного застосунку прийнято рішення використати архітектурний шаблон MVC тому, що проєкт невеликого обсягу, даний архітектурний шаблон доцільно застосувати, модулі програми логічно розділені на компоненти Модель, Представлення та Контроллер.

Проведено дослідження мов програмування, що використовуються для розробки мобільних застосунків: Java, Dart та Swift. Було обрано мову програмування Swift як найпоширеніший інструмент для розробки iOS застосунків.

В роботі виконано огляд використаних бібліотек машинного навчання, фреймворків, рушіїв та допоміжних бібліотек, було описано розроблені класи моделей ContactDetailViewModel та RecentCallViewModel, класу PredictionManager.swift, в якому присутні методи, що відповідають за передбачення подій та класи контролерів:

- BrowserViewController;
- ContactsDetailNavigationController;
- ContactsDetailViewController;

- ContactsNavigationController;
- ContactsViewController;
- ListOfContactsViewController;
- MainTabBarNavigationController;
- MainTabBarViewController;
- SplitContactsNavigationController;
- SplitContactsViewController.

Також виконано опис інтерфейсної частини мобільного застосунку.

Було виконано опис процесу тестування компонентів розробленої системи.

Розроблений мобільний застосунок вирішує задачу передбачення подій або релевантних дій користувача.

Розроблена система має наступні функціональні можливості:

- збирає дані користувача, такі як: локація, дата, час;
- збирає події, в даному випадку дзвінки та історію браузера;
- при зміні локації (не координат, а саме локації, наприклад, при зміні супермаркету Сільпо на супермаркет АТБ) відповідна локація та дата її зміни буде оброблена моделлю штучного інтелекту з використанням бібліотеки TensorFlow та за допомогою алгоритму буде запропоновано найбільш релевантну дію, пов'язану з вихідними даними.

Подальші напрямки розвитку мобільного застосунку полягають в додаванні більшої кількості подій для обробки, наприклад відправлення повідомлення, проведення якихось транзакцій, а також в виконанні портування застосунку на MacOS18:11.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Адаптивный пользовательский интерфейс - Adaptive user interface [Электрон. ресурс]. – Режим доступа: https://360wiki.ru/wiki/Adaptive_user_interface
2. Загальні положення адаптації інтерфейсу до користувача [Електрон. ресурс]. – Режим доступу: <https://studfile.net/preview/5065654>
3. Адаптивный пользовательский интерфейс: проблемы построения [Электрон. ресурс]. – Режим доступа: <http://aaecs.org/hodakov.html>
4. TensorFlow [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/TensorFlow>
5. Автокодуювальник [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BA%D0%BE%D0%B4%D1%83%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA>
6. Метод k-найближчих сусідів [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_k-%D0%BD%D0%B0%D0%B9%D0%B1%D0%BB%D0%B8%D0%B6%D1%87%D0%B8%D1%85_%D1%81%D1%83%D1%81%D1%96%D0%B4%D1%96%D0%B2
7. Для роботи та розваг: топ-15 мобільних додатків на базі штучного інтелекту [Електрон. ресурс]. – Режим доступу: <https://focus.ua/uk/technologies/481128-dlya-roboti-ta-rozvag-top-15-mobilnih-dodatkov-na-bazi-shtuchnogo-intelektu>
8. Модель-вид-контролер [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80>
9. Model-View-Presenter [Електрон. ресурс]. – Режим доступу:

<https://uk.wikipedia.org/wiki/Model-View-Presenter>

10. Clean Swift (VIP) iOS Architecture Pattern [Електрон. ресурс]. – Режим доступу: <https://www.netguru.com/blog/clean-swift-ios-architecture-pattern>

11. Model-View-ViewModel [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Model-View-ViewModel>

12. Гарнаєв, А. WEB-програмування на Java і JavaScript / А. Гарнаєв, С. Гарнаєв. – М.: БХВ-Петербург, 2012. – 179 с.

13. Давидов, С. IntelliJ IDEA. Професійне програмування на Java. Найбільш повне керівництво / С. Давидов, О. Єфімов. – М.: БХВ-Петербург, 2011. – 800 с.

14. Мартін, К. Соломон. Oracle. Програмування на мові Java / Мартін К. Соломон, Нірва Моріссо-Леруа, Джулі Басу. – М.: ЛОРИ, 2010. – 512 с.

15. Бакетт, К. Dart в действии / К. Бакетт. – М.: ДКМ Пресс, 2013. – 528 с.

16. Walrath, K. Dart: Up and Running. / K. Walrath, S.Ladd. – O'Reilly, 2012. – 152 p.

17. Walrath, K. What is Dart? / K. Walrath, S.Ladd. – O'Reilly Media, 2012. – 20 p.

18. Swift Programming Language. [Electronic resource]. – Access mode: <https://developer.apple.com/swift/>

19. Top 5 IDE Tools for iOS App Development [Electronic resource]. – Access mode: <https://findnerd.com/list/view/-Top-5-IDE-Tools-for-iOS-App-Development-/35743/>

20. Дейв, М. iOS 6 SDK. Розробка додатків для iPhone, iPad і iPod touch / [М. Дейв і ін.]. – Москва: РДГУ, 2013. – 672 с.

21. AppCode [Electronic resource]. – Access mode: <https://www.jetbrains.com/ru-ru/objc/>

22. AppCode [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/AppCode>

23. Coderunner [Electronic resource]. – Access mode: <https://macx.ws/mac-os-unix/8627-coderunner.html>

24. TensorFlow Lite [Electronic resource]. – Access mode:
<https://www.tensorflow.org/lite/guide>
25. Matft [Electronic resource]. – Access mode:
<https://swiftpackageregistry.com/jjjkkkjjj/Matft>
26. NumPy [Electronic resource]. – Access mode:
<https://uk.wikipedia.org/wiki/NumPy>
27. WebKit [Electronic resource]. – Access mode:
<https://uk.wikipedia.org/wiki/WebKit>
28. Введення в UIKit framework. Частина 1 [Електрон. ресурс]. – Режим доступу: <http://savelink.org.ua/vvedennya-v-uikit-framework-chastina-1-2/>
29. UIKit - модульний css фреймворк веб-інтерфейсу від yootheme [Електрон. ресурс]. – Режим доступу: <https://jak.bono.odessa.ua/articles/uikit-modulnij-css-frejmwork-veb-interfejsu-vid.php>
30. CallKit [Electronic resource]. – Access mode:
<https://developer.apple.com/documentation/callkit>
31. CoreLocation [Electronic resource]. – Access mode:
<http://macbug.ru/ios/corelocation>
32. CoreData [Electronic resource]. – Access mode:
https://en.wikipedia.org/wiki/Core_Data
33. SVProgressHUD [Electronic resource]. – Access mode:
<https://github.com/SVProgressHUD/SVProgressHUD>
34. Foundation [Electronic resource]. – Access mode:
<https://developer.apple.com/documentation/foundation>
35. Contacts [Electronic resource]. – Access mode:
<https://developer.apple.com/documentation/contacts>
36. Usernotifications [Electronic resource]. – Access mode:
<https://developer.apple.com/documentation/usernotifications>

ДОДАТОК А
Текст програми

A.1 Текст файла PredictionManager.swift

```

import Foundation
import TensorFlowLite
import Matft

class PredictionManager {

    struct Prediction: Hashable {
        var event: Event
        var distance: Float32
    }

    static let shared: PredictionManager = PredictionManager()

    var events : [Event] {
        do {
            return try AppDelegate.shared.currentContext.fetch(Event.fetchRequest())
        } catch {
            print(error.localizedDescription)
            return []
        }
    }

    var normalizedData: [[String]] = []

    var outputs: [[Float32]] = []

    private init() {}

    func predicate(location: Location) {
        normalizedData = []
        outputs = []

        normalizeDataSet()

        let k = 2

        let normalizedCurrentEvent = normalize(from: location.latitude, longitude: location.longitude, date: Date())

        let currentEventInputDataFloat32 = normalizedCurrentEvent.compactMap({ Float32($0) })
    }

```

```

    let currentEventInputData = currentEventInputDataFloat32.withUnsafeBytes { f
in      Data(f)
    }

    let currentEvent = runScript(inputData: currentEventInputData)

    for value in normalizedData {
        let inputDataFloat32 = value.compactMap({ Float32($0) })

        let inputData = inputDataFloat32.withUnsafeBytes { f in
            Data(f)
        }
        self.outputs.append(runScript(inputData: inputData))
    }

    let historyComponents = Array(outputs.reversed())
    let events = Array(self.events.reversed())
    kNN(currentFeatures: currentEvent, historyFeatures: historyComponents, historyOutputs: events, k: k)
}

func normalizeDataSet() {

    for event in self.events {
        normalizedData.append(normalize(from: event.latitude, longitude: event.longitude, date: event.date ?? Date()))
    }

}

func normalize(from latitude: Double, longitude: Double, date: Date) -> [String] {

    let calendar = Calendar.current

    var data: [String] = []

    let hour = calendar.component(.hour, from: date)

    let minutes = calendar.component(.minute, from: date)

    let weekDay = calendar.component(.weekday, from: date)

```

```

let latDms = latitude.dms
let longDms = longitude.dms

// hour
data.append(String(Double(hour)/24))

//minute
data.append(String(Double(minutes)/60))

//lat
data.append(String((latDms.degrees+90)/180))

//lat min
data.append(String(latDms.minutes/60))

//lat sec
data.append(String(latDms.seconds/60))

//long
data.append(String((longDms.degrees+90)/180))

//long min
data.append(String(longDms.minutes/60))

//long sec
data.append(String(longDms.seconds/60))

data.append("0")
data.append("0")
data.append("0")
data.append("0")
data.append("0")
data.append("0")
data.append("0")

switch weekDay {
case 1:
    data[8] = "1"
case 2:
    data[9] = "1"
case 3:
    data[10] = "1"
case 4:
    data[11] = "1"

```

```

    case 5:
        data[12] = "1"
    case 6:
        data[13] = "1"
    case 7:
        data[14] = "1"
    default:
        break
}

return data
}

func runScript(inputData: Data) -> [Float32] {
    guard
        let modelPath = Bundle.main.path(forResource: "new_model", ofType: "tflite")
    else {
        return []
    }

    do {

        // Initialize an interpreter with the model.
        let interpreter = try Interpreter(modelPath: modelPath)

        // Allocate memory for the model's input `Tensor`s.
        try interpreter.allocateTensors()

        //      let inputData: Data // Should be initialized

        // input data preparation...

        // Copy the input data to the input `Tensor`.
        try interpreter.copy(inputData, toInputAt: 0)

        // Run inference by invoking the `Interpreter`.
        try interpreter.invoke()

        // Get the output `Tensor`
        let outputTensor = try interpreter.output(at: 0)

        let data = outputTensor.data

```

```

        var output = Array<Float32>(repeating: 0, count: data.count/MemoryLayout<
Float32>.stride)
        _ = output.withUnsafeMutableBytes { i in
            data.copyBytes(to: i)
        }
        return output

    } catch {
        print(error)
        return []
    }

}

func kNN(currentFeatures: [Float32], historyFeatures: [[Float32]], historyOutputs:
[Event], k: Int) {
    let n = historyFeatures.count

    var distances : [Prediction] = []
    for i in 0..

```

```

}

extension Double {
    var dms: (degrees: Double, minutes: Double, seconds: Double) {
        var seconds = self * 3600
        let degrees = seconds / 3600
        seconds = abs(seconds.truncatingRemainder(dividingBy: 3600))
        return (degrees, seconds / 60, seconds.truncatingRemainder(dividingBy: 60))
    }
}

```

A.2 Текст файла LocationManager.swift

```

import Foundation
import CoreLocation

protocol LocationManagerDelegate: class {
    func didUpdateLocations(_ locations: [CLLocation])
    func didUpdateUserLocation(_ location: Location?)
}

class LocationManager: NSObject {
    var currentUserLocation: Location? {
        didSet {
            self.delegate?.didUpdateUserLocation(self.currentUserLocation)
        }
    }
    let geocoder = CLGeocoder()
    var previousPlacemark: CLPlacemark?
    weak var delegate: LocationManagerDelegate?
    private let manager = CLLocationManager()
    static let shared = LocationManager()
    override private init() {
        super.init()
        self.manager.delegate = self
    }
    func stopUpdatingLocation() {
        self.manager.stopUpdatingLocation()
    }
    func startUpdatingLocation() {
        self.manager.startUpdatingLocation()
    }
    func requestLocationPermissions() {
        self.manager.requestWhenInUseAuthorization()
    }
}

```

```

}

extension LocationManager: CLLocationManagerDelegate {
    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
        if let location = locations.last {
            let wrappedlocation = Location(latitude: location.coordinate.latitude, longitude: location.coordinate.longitude)
            self.currentUserLocation = wrappedlocation
            self.geocoder.reverseGeocodeLocation(location, completionHandler: { placemarks, error in
                if let placemark = placemarks?.first {
                    if self.previousPlacemark?.name != placemark.name {
                        self.previousPlacemark = placemark
                        PredictionManager.shared.predicate(location: wrappedlocation)
                    }
                }
            })
        }
    }
}

struct Location {
    var latitude:Double
    var longitude:Double
}

```

A.3 Текст файла ContactManager.swift

```

import Foundation
import CoreData
import Contacts
import UIKit

let contact_keys = [CNContactFormatter.descriptorForRequiredKeys(for: CNContactFormatterStyle.fullName),
    CNContactPhoneNumbersKey,
    CNContactEmailAddressesKey,
    CNContactBirthdayKey,

```

```
CNContactImageDataKey] as! [CNKeyDescriptor]
```

```
let contact_keysNotes = [CNContactFormatter.descriptorForRequiredKeys(for: CNC
ontactFormatterStyle.fullName),
    CNContactPhoneNumbersKey,
    CNContactNoteKey,
    CNContactEmailAddressesKey,
    CNContactBirthdayKey,
    CNContactFormatter.descriptorForRequiredKeys(for: .fullName),
    CNContactImageDataKey] as! [CNKeyDescriptor]
```

```
class ContactManager {
```

```
    fileprivate var contactStore:CNContactStore = CNContactStore()
```

```
    //Pattern Singleton
```

```
    static let main = ContactManager()
```

```
    private init() { }
```

```
    func getContactStore() -> CNContactStore? {
```

```
        switch CNContactStore.authorizationStatus(for: .contacts) {
```

```
        case .authorized:
```

```
            return self.contactStore
```

```
        case .denied:
```

```
            return nil
```

```
        case .restricted, .notDetermined:
```

```
            return nil
```

```
        }
```

```
    }
```

```
    func requestAccess(completionHandler: @escaping (_ accessGranted: Bool) -
> Void) {
```

```
        switch CNContactStore.authorizationStatus(for: .contacts) {
```

```
        case .authorized:
```

```
            DispatchQueue.main.async {
                completionHandler(true)
            }
```

```
        case .denied:
```

```
            DispatchQueue.main.async {
                completionHandler(false)
            }
```

```
        case .restricted, .notDetermined:
```

```

        self.contactStore.requestAccess(for: .contacts) { granted, error in
            DispatchQueue.main.async {
                completionHandler(granted)
            }
        }
    }
}

func userNameBy(phoneNumber: String) -> String? {
    let keysToFetch = [CNContactFormatter.descriptorForRequiredKeys(for: .fullName), CNContactPhoneNumbersKey, CNContactIdentifierKey] as! [CNKeyDescriptor]

    if let contactsStore = self.getContactStore() {
        do {
            var contacts: [CNContact] = []
            let request = CNContactFetchRequest(keysToFetch: keysToFetch)
            try contactsStore.enumerateContacts(with: request, usingBlock: { contact, pointer in
                contacts.append(contact)
            })
            for contact in contacts {
                if let _ = contact.phoneNumbers.first(where: { $0.value.stringValue.clearedNumber().contains(find: phoneNumber) || phoneNumber.contains(find: $0.value.stringValue.clearedNumber()) }) {
                    return contact.fullName
                }
            }
        } catch {
            print("Failed to fetch contact, error: \(error)")
        }
    }
    return nil
}

func setSocialNumberToContactsIfNeeded() {
    DispatchQueue.global(qos: .userInteractive).async {
        if let contactsStore = self.getContactStore() {
            var contactsArray = [CNContact]()

            let request = CNContactFetchRequest(keysToFetch: contact_keysNotes)

            do {

```

```

> Void in try contactsStore.enumerateContacts(with: request) { (contact, pointer) -

    if contact.phoneNumbers.count > 0 {
        contactsArray.append(contact)
    }
}
}
catch let error as NSError {

    let request = CNContactFetchRequest(keysToFetch:contact_keys)

    do {

        try contactsStore.enumerateContacts(with: request) { (contact, pointer)

-> Void in

            if contact.phoneNumbers.count > 0 {
                contactsArray.append(contact)
            }
        }
    }
    catch let error as NSError {
        print(error.description, separator: "", terminator: "\n")
    }
}

let saveRequest = CNSaveRequest()
for contact in contactsArray {

    guard contact.phoneNumbers.first(where: { $0.label == "Соціальний пр
офіль" }) == nil else {
        continue
    }

    let contactToUpdate = contact.mutableCopy() as! CNMutableContact

    if let phone = contact.phoneNumbers.first {
        var number = phone.value.stringValue
        if number.prefix(2) != "+1" {
            number = "+1" + number
        } else if number.first == "1" {
            number = "+" + number
        }
    }
}

```

```

        let phoneNumber = CNPhoneNumber(stringValue: number)
        let phoneLabel = CNLabeledValue(label: "Соціальний профіль", value: phoneNumber)
        contactToUpdate.phoneNumbers.append(phoneLabel)
        saveRequest.update(contactToUpdate)
    }
    try? contactsStore.execute(saveRequest)
}
}
}

func openSettingsToGiveAccess() {
    UIApplication.shared.open(URL(string: UIApplication.openSettingsURLString)!)
}

func isOpenAccess(completionHandler: @escaping (_ accessGranted: Bool) -> Void) {
    switch CNContactStore.authorizationStatus(for: .contacts) {
    case .authorized:
        DispatchQueue.main.async {
            completionHandler(true)
        }
    case .denied:
        DispatchQueue.main.async {
            completionHandler(false)
        }
    case .restricted, .notDetermined:
        DispatchQueue.main.async {
            completionHandler(false)
        }
    }
}

func authorizationStatusAccess(completionHandler: @escaping (_ authorizationStatus: CNAuthorizationStatus) -> Void) {
    DispatchQueue.main.async {
        completionHandler(CNContactStore.authorizationStatus(for: .contacts))
    }
}
}

```

```
enum ContactError: Error {
    case noteUnavailable
}

extension CNContact {

    func getNote() throws -> String? {

        if self.isKeyAvailable(CNContactNoteKey) {
            return self.note
        } else {
            throw ContactError.noteUnavailable
        }
    }

}
```

A.4 Текст файла NotificationManager.swift

```
import Foundation
import UserNotifications

class NotificationManager: NSObject, UNUserNotificationCenterDelegate {

    static let shared = NotificationManager()

    override private init() {
        super.init()
        UNUserNotificationCenter.current().delegate = self
    }

    func register() {
        UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .badge, .sound]) { success, error in
            if success {

            } else if let error = error {
                print(error.localizedDescription)
            }
        }
    }

    func createScheduleNotification(text: String) {
        let content = UNMutableNotificationContent()
```

```

        content.title = "Передбачення"
        content.body = text
        content.sound = .default

        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 0.1, repeats: false
    )

        let request = UNNotificationRequest(identifier: UUID().uuidString, content: con
tent, trigger: trigger)

        UNUserNotificationCenter.current().add(request)
    }

    //for displaying notification when app is in foreground
    func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notif
ication: UNNotification, withCompletionHandler completionHandler: @escaping (U
NNotificationPresentationOptions) -> Void) {
        completionHandler([.alert, .badge, .sound])
    }
}

```

A.5 Текст файлу AppDelegate.swift

```

import UIKit
import CoreData

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var backgroundTask: UIBackgroundTaskIdentifier = .invalid

    static let shared = UIApplication.shared.delegate as! AppDelegate

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

        self.window = UIWindow(frame: UIScreen.main.bounds)
        registerPushNotifications()
        setupNavigationBarAppearance()
    }
}

```

```

    return true
}

private func rateAppDisplayingSetup() {
    let wasRecorded = UserDefaults.standard.value(forKey: kFirstAppLaunchingDate) != nil

    if !wasRecorded {
        UserDefaults.standard.set(Date(), forKey: kFirstAppLaunchingDate)
    }
}

func setupNavigationBarAppearance() {
    UINavigationBar.appearance().shadowImage = UIImage()
    UINavigationBar.appearance().setBackgroundImage(UIImage(), for: .default)
    UINavigationBar.appearance().backgroundColor = .clear
    UINavigationBar.appearance().backIndicatorImage = UIImage(named: "backNative")
    UINavigationBar.appearance().backIndicatorTransitionMaskImage = UIImage(named: "backNative")
    UINavigationBar.appearance().backItem?.title = ""
}

func registerPushNotifications() {
    UIApplication.shared.registerForRemoteNotifications()
}

// MARK: - Core Data stack

var currentContext:NSManagedObjectContext {
    return persistentContainer.viewContext
}

lazy var managedObjectModel: NSManagedObjectModel = {
    guard let managedObjectModel = NSManagedObjectModel.mergedModel(from: nil) else { fatalError() }
    return managedObjectModel
}()

lazy var persistentMemoryContainer: NSPersistentContainer = {

    let container = NSPersistentContainer(name: "CaptionMateMemory", managedObjectModel: managedObjectModel)

```

```

let description = NSPersistentStoreDescription()
description.type = NSInMemoryStoreType
container.persistentStoreDescriptions = [description]

container.loadPersistentStores(completionHandler: { (storeDescription, error) in
    if let error = error as NSError? {

        fatalError("Unresolved error \(error), \(error.userInfo)")
    }
})
return container
}()

lazy var persistentContainer: NSPersistentContainer = {

    let container = NSPersistentContainer(name: "CaptionMateMemory", managed
ObjectModel: managedObjectModel)
    let description = NSPersistentStoreDescription()

    container.loadPersistentStores(completionHandler: { (storeDescription, error) in
        if let error = error as NSError? {

            fatalError("Unresolved error \(error), \(error.userInfo)")
        }
    })
    return container
}()

// MARK: - Core Data Saving support

func saveCurrentContext () {
    let context = currentContext
    if context.hasChanges {
        do {
            try context.save()
        } catch {
            let nserror = error as NSError
            fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
        }
    }
}

func saveContext () {

```

```

let context = persistentContainer.viewContext
if context.hasChanges {
    do {
        try context.save()
    } catch {
        let nerror = error as NSError
        fatalError("Unresolved error \(nerror), \(nerror.userInfo)")
    }
}

func application(
    _ application: UIApplication,
    didFinishRegisterForRemoteNotificationsWithError error: Error) {
    print("Failed to register: \(error)")
}

}

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»

Магістерська робота на тему:

Дослідження та програмна реалізація технологій інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів

Виконав:
Керівник:

Нездроп І.П., ст.гр. КНТ-220м
Сердюк С.М., доц., к.т.н.

Рисунок Б.1 – Слайд 1

Мета та задачі роботи

Мета роботи - розробка мобільного застосуку, що дозволяє виконати передбачення подій або релевантних дій користувача залежно від локації та дня тижня та часу з використанням методів штучного інтелекту при реалізації адаптивних інтерфейсів

Для досягнення поставленої мети в дипломній кваліфікаційній роботі потрібно вирішити наступні **задачі**:

- виконати огляд підходів щодо розробки адаптивних інтерфейсів;
- виконати дослідження підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів;
- описати основні функціональні вимоги до розроблюваного мобільного застосунку;
- обґрунтувати вибір технологій, середовища розробки та мови програмування;
- розробити мобільний застосунок для передбачення подій або релевантних дій користувача;
- виконати тестування компонентів мобільного застосунку.

2

Рисунок Б.2 – Слайд 2

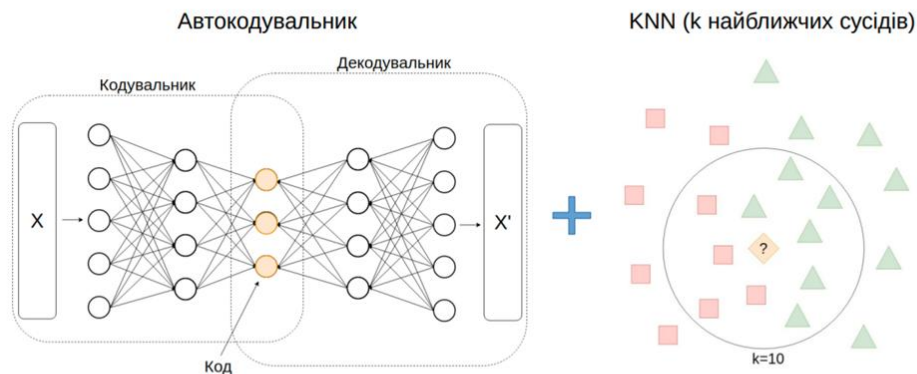
Підходи інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів

- розробка алгоритмів штучного інтелекту;
- використання спеціалізованих бібліотек штучного інтелекту в залежності від мови програмування;
- розробка набору широко використовуваних функцій або прикладного програмного інтерфейсу (Application Programming Interface, API).

3

Рисунок Б.3 – Слайд 3

Алгоритм нейронної мережі (автокодувальник) та методу k найближчих сусідів



4

Рисунок Б.4 – Слайд 4

Опис алгоритму

Позначення: A – поточні дані; A' – вектор-код; D – база даних виду $B - B' - Y$, де B – нормальні історичні дані, B' – закодовані дані, Y – дії користувача; N – розмір історії (останні записи); HN – останні N записів у базі даних D ; k – кількість сусідів для алгоритму kNN ; S – деяка гранична відстань для алгоритму kNN ; E – кодувальник (нейромережа).

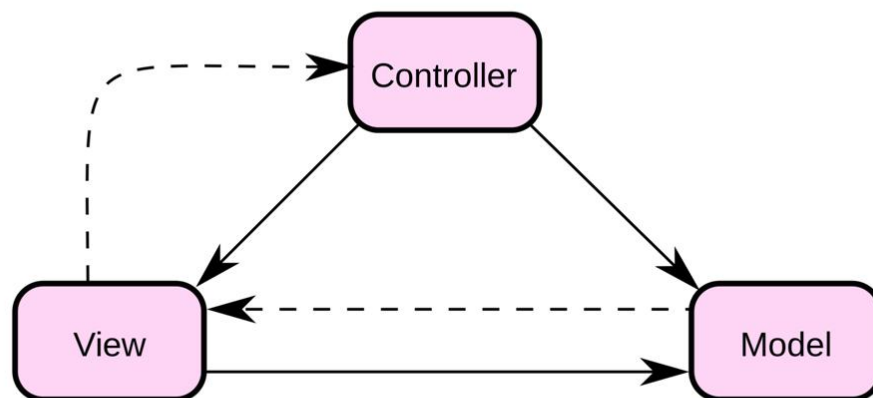
Наведемо опис алгоритму :

- Крок 1. У певний час система збирає поточні дані A про локацію, час та дату.
- Крок 2. Дані A кодуються кодувальником E у вектор-код A' .
- Крок 3. Алгоритмом kNN знаходиться найчастіша дія Y' з HN серед k найближчих векторів.
- Крок 4. Якщо відстань до найближчого серед k вектор-коду більша, ніж S , то перейти до кроку 1.
- Крок 5. Користувачу пропонується виконати дію Y' .
- Крок 6. Якщо користувач відмовляється, то перейти до кроку 1, інакше перейти до кроку 7.
- Крок 7. У базу даних D заносяться запис вигляду $A - A' - Y'$.

5

Рисунок Б.5 – Слайд 5

Архітектурний шаблон MVC

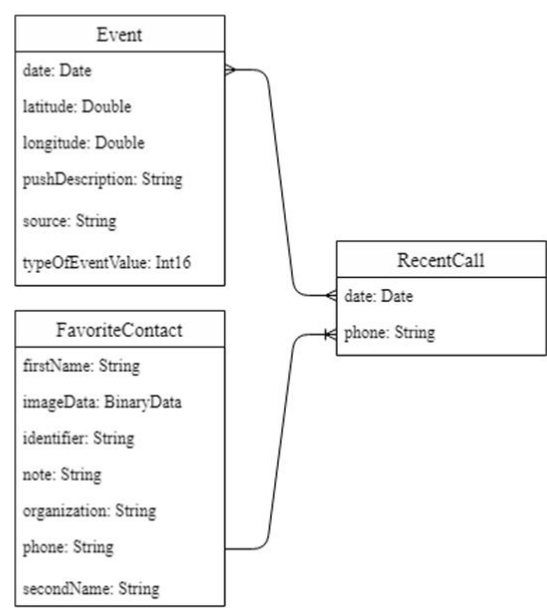


Модель-вид-контролер [Електрон. ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80>

6

Рисунок Б.6 – Слайд 6

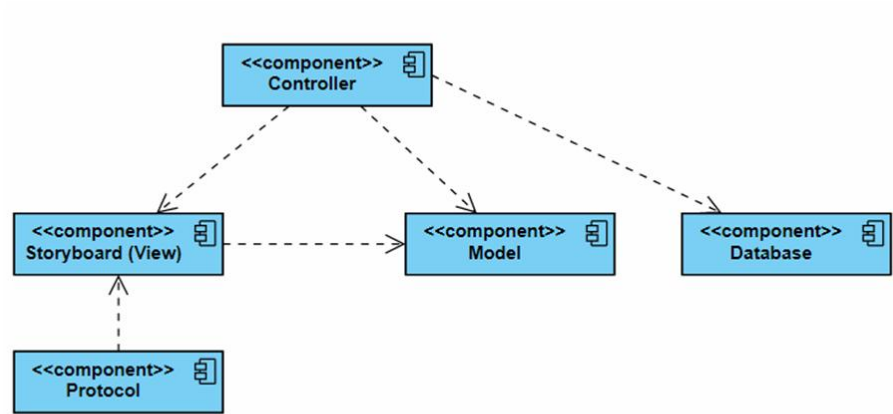
Схема бази даних



7

Рисунок Б.7 – Слайд 7

Діаграма компонентів



8

Рисунок Б.8 – Слайд 8

Порівняльна таблиця мов програмування

	Java	Dart	Swift
Передбачуване використання	Мобільні застосунки, бізнес, веб-програми, клієнтська та серверна сторони	Мобільні застосунки, веб-програми	Мобільні застосунки, десктоп
Імперативне	+	+	+
Об'єктно-орієнтоване	+	+	+
Функціональне	-	-	-
Процедурне	-	-	-
Узагальнене	+	-	+
Рефлексивне	+	-	+
Подійно-орієнтоване	-	-	-
Інші парадигми	Конкурентне	Структурне	Конкурентне
Стандартизація	Java Language Specification	Ні	Ні

9

Рисунок Б.9 – Слайд 9

Розроблені класи Моделі та Контролерів

Класи Моделі:

- ContactDetailViewModel – модель відображення деталей контакту;
- RecentCallViewModel – модель відображення недавніх дзвінків.

Класи Контролерів:

- BrowserViewController – контролер, що призначений для перегляду сторінок вебсайтів(браузера);
- ContactsDetailNavigationController – контролер, що призначений для створення навігації в деталях контакту;
- ContactsDetailViewController – контролер, що призначений для відображення деталей контакту;
- ContactsNavigationController – контролер, що призначений для створення навігації на списку контактів;
- ContactsViewController – контролер, що призначений для перегляду контактів;
- ListOfContactsViewController – контролер, що призначений для перегляду списку контактів;
- MainTabBarController – контролер, що призначений для створення навігації в панелі вкладок;
- MainTabBarViewController – контролер, що призначений для створення в панелі вкладок;
- SplitContactsNavigationController – контролер, що призначений для навігації перегляду списку контактів та вибраних контактів;
- SplitContactsViewController – контролер, що призначений для перегляду списку контактів та вибраних контактів (служить як контейнер для них).

10

Рисунок Б.10 – Слайд 10

Клас PredictionManager

```
PredictionManager: class
- _let_shared: PredictionManager_
- var events: [Event]
- var normalizedData: [[String]]
- var outputs: [[Float32]]
- predicate(location: Location)
- normalizeDataSet()
- normalize(from latitude: Double, longitude: Double, date: Date) -> [String]
- runScript(inputData: Data) -> [Float32]
- kNN(currentFeatures: [Float32], historyFeatures: [[Float32]], historyOutputs: [Event], k: Int)
- init()
```

- структура Prediction містить поля event (подія) та distance (відстань);
- масив events містить перелік подій;
- масив normalizedData необхідний для нормалізації даних;
- масив outputs для отримання вихідного результату;
- функція predicate, що дозволяє здійснити передбачення з використанням методу k-найближчих сусідів та нейронної мережі;
- функції normalizeDataSet та normalize призначені для нормалізації даних;
- функція runScript запускає роботу побудованої моделі нейронної мережі;
- функція kNN реалізує метод k-найближчих сусідів.

11

Рисунок Б.11 – Слайд 11

Чек-лист тестування мобільного застосунку

Перевірка	Модель - iPhone 11 Pro Max, процесор Apple A13 Bionic	Модель - iPhone X, процесор Apple A11 Bionic	Модель - iPhone 8, процесор Apple A8
Завантаження застосунку	пройдено	пройдено	пройдено
Адаптивність інтерфейсу	пройдено	пройдено	пройдено
Відновлення застосунку після згортання	пройдено	пройдено	пройдено
Переходи між вкладками	пройдено	пройдено	пройдено
Повернення до попереднього вью контролера	пройдено	пройдено	пройдено
Додавання контакту	пройдено	пройдено	пройдено
Редагування контакту	пройдено	пройдено	пройдено

12

Рисунок Б.12 – Слайд 12

Чек-лист тестування мобільного застосунку

Перевірка	Модель - iPhone 11 Pro Max, процесор Apple A13 Bionic	Модель - iPhone X, процесор Apple A11 Bionic	Модель - iPhone 8, процесор Apple A8
Видалення контакту	пройдено	пройдено	пройдено
Додавання контакту в обране	пройдено	пройдено	пройдено
Видалення контакту з обраного	пройдено	пройдено	пройдено
Пошук контактів	пройдено	пройдено	пройдено
Робота браузера	пройдено	пройдено	пройдено
Переходи між вкладками в браузері	пройдено	пройдено	пройдено
Передбачення	пройдено	пройдено	пройдено
Push сповіщення	пройдено	пройдено	пройдено

13

Рисунок Б.13 – Слайд 13

Висновки

- В результаті виконання дипломної кваліфікаційної роботи магістра розглянуто поняття адаптивного користувацького інтерфейсу, виконано огляд підходів щодо розробки адаптивних інтерфейсів та дослідження підходів інтеграції методів штучного інтелекту при розробці адаптивних інтерфейсів.
- Було прийнято рішення виконати інтеграцію у вигляді модуля з використанням спеціалізованих бібліотек машинного навчання TensorFlow та NumPy при розробці адаптивного інтерфейсу мобільного застосунку.
- Було розглянуто архітектурні шаблони проектування мобільного застосунку: Модель–представлення–контролер (Model View Controller, MVC)– Модель–Представлення–Пред'явник (Model View Presenter, MVP), Clean Swift або VIP (View controller, Interactor та Presenter) та Model View ViewModel. Було виконано проектування структури бази даних, діаграм класів та компонентів.
- Для реалізації мобільного застосунку прийнято рішення використати архітектурний шаблон MVC тому, що проєкт невеликого обсягу, даний архітектурний шаблон доцільно застосувати, модулі програми логічно розділені на компоненти Модель, Представлення та Контроллер.
- Проведено дослідження мов програмування, що використовуються для розробки мобільних застосунків: Java, Dart та Swift. Було обрано мову програмування Swift як найпоширеніший інструмент для розробки iOS застосунків.

14

Рисунок Б.14 – Слайд 14

Висновки

- В роботі виконано огляд використаних бібліотек машинного навчання, фреймворків, рушіїв та допоміжних бібліотек, було описано розроблені класи моделей ContactDetailViewModel та RecentCallViewModel, класу PredictionManager.swift, в якому присутні методи, що відповідають за передбачення подій та класи контролерів. Також виконано опис інтерфейсної частини мобільного застосунку.
- Було виконано опис процесу тестування компонентів розробленої системи.
- Розроблений мобільний застосунок вирішує задачу передбачення подій або релевантних дій користувача.
- Розроблена система має наступні функціональні можливості:
- збирає дані користувача, такі як: локація, дата, час;
- збирає події, в даному випадку дзвінки та історію браузера;
- при зміні локації (не координат, а саме локації, наприклад, при зміні супермаркету Сільпо на супермаркет АТБ) відповідна локація та дата її зміни буде оброблена моделлю штучного інтелекту з використанням бібліотеки TensorFlow та за допомогою алгоритму буде запропоновано найбільш релевантну дію, пов'язану з вихідними даними.
- Подальші напрямки розвитку мобільного застосунку полягають в додаванні більшої кількості подій для обробки, наприклад відправлення повідомлення, проведення якихось транзакцій, а також в виконанні портування застосунку на MacOS18:11

15

Рисунок Б.15– Слайд 15

Дякую за увагу!

16

Рисунок Б.16 – Слайд 16