Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Розробка мобільного додатку для системи автоматизованого поселення до гуртожитку

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
асистент
Франків О. О.
_____
(Підпис)
“___” _____ 2022 року
Виконала студентка ІПЗ-БП4
Ксенофонтова С.В.
“___” _____ 2022 року

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики

ЗАТВЕРДЖУЮ

Зав. Кафедри інформатики,

доцент, к.ф.-м.н

_____ С.С. Гороховський

(підпис)

“____” _____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студентці Ксенофонтовій Софії Вячеславівні факультету інформатики 4 курсу

ТЕМА: Розробка мобільного додатку для системи автоматизованого поселення до гуртожитку

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Аналіз предметної області та постановка завдання курсової роботи

Теоретичні відомості (про задачу, методи і підходи до її розв’язку тощо)

Опис реалізації додатку

Висновки

Список використаної літератури

Додатки

Дата видачі “____” _____ 2022 р.

Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання роботи

Тема: Розробка мобільного додатку для системи автоматизованого поселення до гуртожитку

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової	29.09.2021	
7.	Планування структури та архітектури практичної частини роботи, розробка макету додатку в Figma	10.11.2021-15.02.2022	
12.	Програмування застосунку	Березень-квітень 2022	
13.	Написання текстової частини роботи	10.05.2022	
15.	Здача курсової роботи на перевірку	20.05.2022	

Студентка Ксенофонтова С.В.

Керівник Франків О. О. “____” _____

ЗМІСТ

Вступ

Актуальність теми та її практичне значення

Беззаперечним фактом є те що життя сучасної людини нероздільно пов’яне з інформаційними технологіями та їх швидкісним розвитком.
Впровадження концепцій останньої четвертої промислової революції в сферу інформаційних технологій сприяло злиттю автоматизованого виробництва, обміну даних і виробничих технологій в єдину саморегульовану систему, що не потребує втручання людини. Це в свою чергу нероздільно пов’язало життя кожного з нас зі своїми смартфонами.

За наявними даними Ericsson & The Radicati Group [1], у 2022 кількість користувачів смартфонів у світі сягає 6,648 мільярдів, а це означає, що 83,72% населення світу володіє смартфоном. За прогнозами статистичної компанії Statista [2] ця кількість щороку тільки зростатиме і сягне позначки в 7.69 мільярдів користувачів вже до кінця 2027 року.

Смартфон сьогодні – це продовження особистості кожного з нас, місце для навчання, роботи, відпочинку та вирішення будь яких життєвих питань.

Особливо важливими для кожної молодої людини постають питання пов’язані з університетським життям. Щороку велика частина студентів потребує вирішення питання місця проживання впродовж навчального року. Маючи різноманітні обставини фінансового становища, родинних взаємозв’язків, попередніх житлових умов, багато хто обирає саме університетські гуртожитки як місце свого притулку на майбутній навчальний рік або роки. Проте, нажаль, враження від перших навчальних днів губиться у вирі гуртожиткової бюрократії за стосами таких вкрай необхідних довідок та бланків. Тож, актуальною постає проблема пришвидшення процесу поселення індивіда і надання йому всієї необхідної інформації про майбутні житлові умови, правила та терміни.
Адреса майбутнього гуртожитку, номер кімнати, актуальні ціни, інформація щодо переселення, терміни сплати, документи необхідні для поселення та правила проживання це мінімальний перелік інформації, який цікавитиме кожного майбутнього резидента. Безумовно, кожному хто планував би своє проживання в

гуртожитках певного університету хотілося б дізнаватися цю інформацію якомога швидше і безболісніше. Саме тому так важливо надати студентам змогу мати всі ці функції у себе під рукою, просто завантаживши програму з відповідного маркетплейсу у своєму смартфоні.

Тож, метою та завданням даної курсової роботи є розробка додатку для платформи iOS для автоматизованого поселення в університетські гуртожитки, в даному конкретному випадку в гуртожитки НаУКМА.

При створенні системи було використано:

- Swift версії 5.5.1, Xcode
- Бібліотеки Alamofire, MSAL
- Сервіс Firebase Cloud Messaging
- Apple Push Notification Service
- Figma, UIKit
- Git

Структура роботи

Робота складається з вступу та трьох основних розділів, висновків, списку джерел та додатків.

- У першому розділі проводиться аналіз та обґрунтування предметної області.
- У другому розділі розглянуті основні технології та рішення, що були використані під час розробки.
- У третьому розділі та висновку описані складові частини готового застосунку, плани щодо подальшого вдосконалення та висновки за результатами проведеної роботи.

Постановка задачі

1. Обґрунтувати актуальність розробки мобільного додатку за умови наявності повноцінної працюючої веб версії сервісу автоматизованого поселення в гуртожитки
2. Дослідити ринок на наявність додатків-аналогів заданої предметної області
3. Проаналізувати актуальні принципи розробки мобільних додатків

4. Дослідити технології та порівняти наявні рішення для створення iOS додатків безпосередньо
5. Використовуючи мову Swift та досліджені технології, реалізувати iOS додаток для автоматизованого поселення в гуртожитки

РОЗДІЛ 1 Аналіз предметної області. Постановка завдання курсової роботи

1.1 Аналіз сучасного стану питання, огляд існуючих аналогів та обґрунтування теми

З огляду на описану мною раніше важливість та щорічну актуальність питання забезпечення студентів місцями для проживання в гуртожитках та за відсутності на ринку аналогів, на момент 2020 року, які можна було б з легкістю інтегрувати та адаптувати під цей процес, стараннями студентської організації НаУКМА Fido, світ побачив сайт DMS(Dormitory Management System), що являє собою веб клієнт для електронного поселення в гуртожитки (див. Додаток А). DMS реалізовує автоматизований процес поселення/переселення, оплати проживання студентів у гуртожитках НаУКМА. Дозволяє користувачу швидко і зручно переглядати документи необхідні для поселення/переселення та отримувати листи на корпоративну пошту, щодо оновлення статусів попередніх запитів до системи.

У зв’язку з зростанням функціональних можливостей продукту DMS та беззаперечним щорічним збільшенням використання всесвітнього інтернет трафіку саме з мобільних носіїв [3], постало питання розробки мобільного додатку DMS, котрий би значно пришвидшував та полегшував процес взаємодії з системою поселення для студентів.

В питанні порівняння мобільного додатку з веб версією, відкритою через мобільний браузер, серед переваг мобільного додатку можна виділити такі [4]:

- Простота надсилання сповіщень

Маючи доступ до вбудованого в смартфон центру сповіщень додатки мають змогу надсилати миттєві, ненав’язливі сповіщення, що є бажаною опцією в разі необхідності інформувати користувача про певні події пов’язані з вашим продуктом і однією з ключових переваг додатку над мобільною версією сайту.
Використання функцій мобільного пристрою
Мобільні програми мають перевагу використання функцій мобільного пристрою, таких як камера, список контактів, телефонні дзвінки, календар, GPS та служби локації, акселерометр, гіроскоп, тощо. Хоч деякі сайти також де-юре мають змогу

використовувати функції вашого смартфону, проте де-факто список цих функцій буде значно зменшений, матиме технологічні та конфіденційні обмеження і вимагатиме щоразового залучення від користувача на надання дозволу, наприклад, на використання камери або GPS свого телефону. Тож, використання функцій мобільного пристрою “з коробки” значно скорочують час, необхідний для виконання певного завдання користувачем у додатку і покращують враження від взаємодії з ним.

- Можливість працювати в автономному режимі

Одна з фундаментальних переваг додатку. Хоча, програми як і веб-сайти, також можуть вимагати підключення до Інтернету для виконання більшості функцій, але ключова різниця полягає в можливості програми надавати користувачу основний вміст і функціональність в автономному режимі. У той час як мобільні веб-сайти, за відсутності інтернет підключення найчастіше обмежується лише доступністю перегляду попередньо кешованого контенту веб-сторінки.

- Повна свобода в дизайні

Незважаючи на плеяду технологічних досягнень у веб-дизайні, мобільні веб-сайти все ще покладаються на браузери для виконання навіть найпростіших функцій. Мобільні веб-сайти у своєму дизайні залежать від функцій браузера, таких як «кнопка назад», «кнопка оновлення» та «адресний рядок». А мобільні додатки у свою чергу, не мають жодного з цих обмежень. Додатки можуть використовувати звичайні та просунуті жести для взаємодії з сенсором, щоб запропонувати інноваційні функції, для пришвидшення та покращення виконання завдань у додатку користувачами. Додаток, до прикладу, може дозволяти користувачам перейти до наступного або попереднього кроку за допомогою жесту гортання. Беручи до уваги описану актуальність обраної предметної області, відсутність аналогів та ринку та переваги у використанні додатку, безумовним є факт необхідності реалізації мобільного клієнта DMS при наявності уже існуючого рішення для браузеру.

1.2 Постановка задачі

Головною задачею даної курсової роботи є розробка додатку під ОС iOS, котрий являтиме собою мобільний клієнт системи автоматизованого поселення до гуртожитків DMS. Клієнт мобільного додатку спілкуватиметься з сервером DMS через існуючу API.

Функціонал додатку включатиме такі можливості:

- Аутентифікацію через сервіси Microsoft, а саме через корпоративну пошту НаУКМА
- Перегляд інформації про аутентифікованого користувача
- Додавання та перегляд користувачем пільг
- Поселення користувача (поки що без можливості оплати)
- Поселення користувача з пільгою
- Надсилання користувачу сповіщень про спливання терміну оплати, про зачинення гуртожитку в якому проживає користувач за допомогою сервісу Firebase Cloud Messaging
- Перегляд та завантаження документів необхідних для поселення, завантаження пільг, тощо.

Також, метою написання даної курсової роботи є персональне бажання доповідача поглибити знання в мобільній розробці, а саме в таких навичках як:

- мова програмування Swift останньої доступної версії,
- робота з користувацьким інтерфейсом за допомогою UIKit
- робота з REST API та перегляд існуючої WebAPI документації Swagger
- робота з HTTP запитами безпосередньо за допомогою бібліотеки Alamofire

А також набути навички роботи з :

- бібліотекою MSAL (Microsoft Authentification Library) [5]
- технологією Apple Push Notification Service[6]
- сервісом Firebase Cloud Messaging [7]

РОЗДІЛ 2 Теоретичні відомості (про задачу, методи і підходи до її розв'язку тощо)

2.1 Основні поняття про REST API

Описаний і представлений у своїй дисертації науковцем Роєм Філдінгом у 2000 році [8] архітектурний підхід REST (Representational State Transfer) , що в перекладі означає «передача репрезентативного стану», дозволяє проектувати розподілені системи за допомогою певного набору обмежень:

- Client-Server

Система має бути побудована з використанням клієнт-серверної архітектури, що міститиме такі компоненти як: клієнти, сервер, і ресурси. Поведінка кожного з окремих компонентів, у свою чергу, керуватиметься через HTTP.

- Stateless

Взаємодія між компонентами має відбуватися без збереження стану. Сервер не повинен зберігати будь-яку інформацію про клієнтів, адже вона у разі необхідності ідентифікації клієнта повинна зберігатися у запиті.

- Cache

Можливість кешування даних. Для запобігання повторного використання клієнтами застарілих або некоректних даних у відповідь на подальші запити.

- Uniform Interface

Обмеження щодо наявності єдиного логічного інтерфейсу для зв'язування компонентів між собою.

Останнє REST обмеження, щодо наявності єдиного логічного інтерфейсу, буде вважитися виконаним, якщо інтерфейс у своїй побудові відповідатиме таким критеріям:

- Identification of resources (заснований на ресурсах)

У REST архітектурі ресурсом є все, чому можна дати назву. Цей ресурс повинен бути стабільно ідентифікований за допомогою ідентифікатора, що не змінюватиметься при зміні стану ресурсу. Ідентифікатором у REST є URI.

- Manipulation of resources through representations (Маніпуляції над ресурсами через представлення)

Для виконання дій над ресурсами підхід REST послуговується представленням, що презентує поточний чи бажаний стан ресурсу. Наприклад, якщо ресурс це користувач, то поданням може бути опис цього користувача у форматі XML, HTML або JSON.

- Self-descriptive messages (самодокументовані повідомлення)

Під самодокументованістю повідомлень мається на увазі, що запит і відповідь на нього повинні зберігати в собі всю необхідну інформацію для власної обробки.

- HATEOAS (Hypermedia as the engine of application state)

Статус ресурсу передається через запит, а саме через безпосередній вміст його тіла (body), рядку параметрів (query), заголовків (headers) та endpoint`у URI (ім'я ресурсу). Усі перелічені складові називаються гіпермедіа (або гіперпосилання з гіпертекстом).

REST на сьогоднішній день практично витіснив решту підходів, у тому числі дизайн заснований на SOAP [9] і WSDL [10]

Щодо API (Application Programming Interface) [11], то це набір певних визначень і протоколів для створення та інтеграції прикладного програмного забезпечення. API за своє суттю є посередником та дозволяє взаємодію одних програм з іншими в попередньо визначеному форматі. Програми, зберігаючи безпеку, контроль та аутентифікацію, мають змогу обмінюватися ресурсами та інформацією, визначаючи, хто до чого отримує доступ. До прикладу, програма клієнт має змогу доступатися до функцій програми веб-ресурсу, не поглиблюючись в особливості кешування, адже ще одна перевага API полягає в тому, що програмі яка з ним взаємодіє не потрібно знати як витягується ресурс або з якого конкретного місця на веб-ресурсі він надходить. API дає можливість різним клієнтам використовувати той самий веб-ресурс і читати потрібні їм дані, що в загальному вигляді описує основну мету використання API, як таку, що має об’єднати роботу різних мобільних або веб додатків в єдину органічно-симбіотичну систему.

Отже, підсумувавши все вище сказане та з’єднавши ці два поняття: архітектурний підхід та інтерфейс, отримуємо — REST API (також відомий як

RESTful API) — інтерфейс програмування прикладних програм (API або веб-API), який відповідає описаним вище обмеженням архітектурного стилю REST і дозволяє взаємодіяти з веб-сервісами RESTful. Робота з REST API по суті складається з обміну повідомленнями JSON через HTTP, адже JSON є домінуючим форматом в REST API, оскільки він є компактним, легким для розуміння та має гнучкий формат, який не вимагає попередньо оголошеної схеми [12].

2.2 Основні поняття про Apple Push Notifications Servise

Базовою і першочерговою задачею, вперше представленого для iOS 3 в 2009 році [13], Apple Push Notifications Servise (APN) є надання можливості розробникам сторонніх додатків надсилати сповіщення до програм, встановлених на пристроях Apple. Використання цього сервісу дозволяє створювати та надсилати сповіщення локально зі свого додатка або ж віддалено з керованого сервера. Для надсилання віддалених сповіщень сервер створює push-повідомлення, а APN у свою чергу обробляє безпосередньо сам процес доставки цих сповіщень на пристрої користувача. Сервіс APN дозволяє реалізовувати такі функції:

- визначати типи сповіщень, підтримувані розробленим додатком;
- визначати будь-які спеціальні дії, пов’язані з необхідними для вашого додатку типами сповіщень;
- обробляти сповіщення що вже доставлені;
- відповідно реагувати на дії користувача, щодо сповіщень;

2.3 Основні поняття про мобільну розробку під iOS

Компанія Apple славиться тим що всі додатки написані під її операційні системи (скор. ОС) iOS, macOS, watchOS, tvOS повинні мати певний, підпорядкований певному набору правил та принципів, зовнішній вигляд. Відрізняючись безпосередньо для кожної ОС цей набір правил називається Human Interface Guidelines (далі HIG).

Оскільки темою даною курсової роботи є розробка додатку саме для ОС iOS, пропоную розглянути характеристики [14] що відрізняють дану ОС з поміж інших:

- Ясність розуміння компонентів

У всій iOS системі текст є розбірливим у будь-якому своєму розмірі, значки та іконки програм точні й чіткі, прикрашання інтерфейсу є витонченим й доречним, а гостра зосередженість на функціональності є основним рушієм дизайну під цю ОС. Усе в системі від вільного простору, до кольору, шрифтів, графіки та елементів інтерфейсу має свою особливу роль, в тонкому виділенні важливого вмісту інтерфейсу і інтерактивної передачі користувачу.

- Повага до своїх користувачів.

Плавні рухи для взаємодії з ОС і чіткий та красивий інтерфейс допомагають людям розуміти безпосередній вміст цього інтерфейсу, ніколи не змагаючись з ним задля виконання певної дії, як це нерідко буває в ОС інших компаній. Контент зазвичай займає весь екран, тоді як напівпрозорість і розмиття часто натякають на більше. Мінімальне використання рамок, градієнтів і тіней робить інтерфейс легким і летючим, надаючи при цьому першорядне значення контенту на екрані.

- Гра з глибиною

Чіткі візуальні шари та реалістичні рухи передають ієрархію, оживляють та полегшують розуміння інтерфейсу. Дотик і гра з видимістю контенту підвищують задоволення від взаємодії, надають доступ до функцій та додаткового вмісту без втрати контексту. Переходи під час навігації по контенту забезпечують користувачу саме це відчуття глибини.

Нижче наведу принципи проектування додатків, які виділені HIG [14] саме для ОС iOS:

- Естетична цілісність додатку

Естетична цілісність показує, наскільки добре інтерфейс користувача поєднується з основною функцією яку має виконувати додаток. Наприклад, програма, яка вимагатиме від користувача виконання серйозного завдання, повинна тримати

його зосередженим, використовуючи тонку, ненав’язливу графіку, мати стандартні елементи керування інтерфейсом та передбачувану поведінку. У той же час, захоплюючий вид програм, наприклад ігри, повинні мати динамічний вигляд, що обіцятиме користувачеві веселощі під час використання, водночас заохочуючи його до нових ігрових відкриттів.

- Послідовність у використанні

Додаток розроблений під iOS має реалізувати знайомі стандарти та парадигми, використовуючи надані системою елементи інтерфейсу, добре відомі піктограми, стандартні стилі тексту та єдину термінологію. Додаток повинен включати та природньо реалізовувати функції та поведінку, так, як того очікують його користувачі.

- Пряма маніпуляція вмістом екрану

Безпосереднє маніпулювання вмістом на екрані залучає людей і полегшує розуміння. Послуговуючись прямими маніпуляціями, такими як: фізичне обертання пристрою або використання природніх жестів користувачі повинні побачити негайні, видимі результати своїх дій.

- Зворотній зв'язок

Зворотній зв’язок наявний у створеному розробником додатку повинен підтверджувати дії та показувати результати, для інформування свого користувача. Вбудовані програми iOS, до прикладу, забезпечують відчутний зворотний зв’язок у відповідь на кожну дію користувача: інтерактивні елементи при натисканні коротко підсвічуються, індикатори прогресу повідомляють про стан довготривалих операцій, а звук та анімація допомагають уточнити результати дій.

- Метафори в інтерфейсі

Загальновідомим є те, що люди швидше навчаються, коли віртуальні об’єкти та дії програми є метафорами знайомого досвіду, будь це досвід реальний або цифровий. Саме тому повсюди в iOS наявні добре працюючі метафори. Люди

перетягують вміст, перемикають перемикачі, переміщують повзунки, прокручують списки, гортають сторінки – усі ці дії є метафорами з реального життя кожного з нас.

- Контроль у користувача

Контроль над усім в iOS повинен лежати десь посередині між користувачем і додатком. Хоча додаток може запропонувати спосіб дій або попередити про небезпечні наслідки, помилковим є перекладання на програму повного прийняття рішень. Найкращі програми балансують між активністю користувачів і уникненням небажаних результатів.

2.4 Патерн MVVM для розробки під iOS

Мабуть найрозповсюдженішим шаблоном проектування для архітектури програмних додатків iOS є Model-View-Controller, або скорочено MVC, котрий використовується системними вбудованими програмами. Також, велика кількість Apple фреймворків написані саме з використанням цього шаблону. Проте, в руках недосвідченого розробника такий шаблон може призвести до перенавантаження UIViewController'а (який в ОС iOS відповідає за компоненту Controller'а патерна MVC) обов'язками і виникнення так званої проблеми Massive View Controller'а [15]. Тож, у зв'язку з цим у спільноті iOS розробників останніми роками набирає обертів використання шаблону Model-View-ViewModel, скорочено MVVM. Плануючись як модифікація шаблону Presentation Model шаблон MVVM побачив світ у 2005 році завдяки Джону Госсману [16]. Основною метою MVVM є розмежування моделі та представлення(див. Рисунок 2.4.1). Для зв'язку між рівнями в такому випадку використовується binding, що буквально означає "зв'язування даних". Основна відмінність даного архітектурного шаблону від MVC, полягає в тому що він вводить четверту компоненту – модель представлення (View-Model). Модель представлення відповідає за керування моделлю та передачу її даних до представлення через контролер. Отже, не зважаючи на свою назву, MVVM складається з таких основних компонент: Model, View, View-Model, Controller. У той час як реалізації Model і View, звично для

патерну MVC, відповідають за управління даними і їх представлення користувачу відповідно, реалізація View-Model у цьому шаблоні зазвичай проста, адже основна її задача — це перекладання даних з моделі до значень, що можуть відображатися у View. Що дозволяє розвантажити UIViewController та уникнути вищеописаної проблеми його перевантаження.

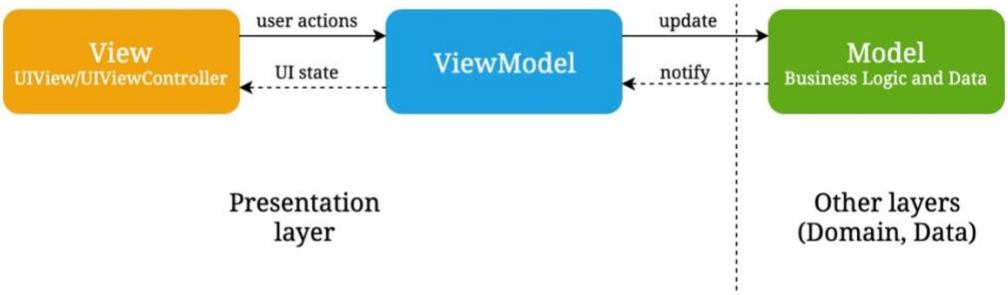

Рисунок 2.4.1 структура шаблону MVVM

РОЗДІЛ 3 Опис реалізації додатку

3.1 Порівняння використання інструментів UIKit та SwiftUI для розробки під iOS

Оскільки однією з заповідей розробки під ОС компанії Apple, описаною у попередньому розділі, є наявність у програми приголомшливого інтерфейсу користувача (UI), то наразі на ринку можна виділити два ключові фреймворки, що допомагають у цьому питанні: UIKit та SwiftUI. Пропоную провести порівняння даних фреймворків за наведеними нижче параметрами:

- Конструктор інтерфейсу

У той час як UIKit надає можливість визначити весь UI в коді або через Interface Builder, використовуючи storyboards, SwiftUI не має Interface Builder і надає перевагу конструюванню інтерфейсу прямо з коду.

- Оновлення інтерфейсу користувача

За допомогою UIKit розробники повинні визначити, що відображатиметься на екрані, коли екран оновиться, та в який саме момент повинне відбуватися дане оновлення. Вони також повинні визначити, як відбуватиметься перехід між різними станами UI. На відміну від цього, SwiftUI є реактивним, що означає динамічну зміну інтерфейсу користувача, в залежності від змін в коді. Це дає SwiftUI перевагу в швидкому доступу до перегляду зміненого інтерфейсу, через внесення в код змін, що пришвидшує процес його побудови.

- Документація та підтримка

Оскільки SwiftUI є досить молодим рішенням, що було представлене компанією Apple у 2019 році [17], то підтримує даний фреймворк лише iOS починаючи з 13 версії, а Xcode починаючи з 11. Це дає значну перевагу UIKit, адже він може підтримувати значно старіші версії, починаючи з iOS 9, що дозволяє використовувати його для розробки програм під більш давні пристрої компанії Apple. Також, на відміну від UIKit, через свою незрілість, SwiftUI використовується розробниками досить недавно, що прямо впливає на кількість доступних і описаних іншими розробниками рішень проблеми, з якою може стикнутися новоспечений розробник що пише на SwiftUI.

- Швидкість розробки програми

Загалом, наразі SwiftUI , за думкою багатьох розробників дає можливість найшвидшого створення функцій у програмі. За допомогою SwiftUI, розробник може досягти тих самих результатів що і за використання UIKit, проте з значно меншою кількістю написаного коду. Крім того, значно прискорює розробку описана вище, функція попереднього перегляду в реальному часі.

- Функціональність анімації

Використання красивої анімацій та переходів значно легше саме в SwiftUI, адже він дає користувачу доступ до великої кількості безкоштовних анімацій. Крім того, набагато легшими стають екранні переходи. Використовуючи UIKit програміст повинен визначати усі анімації вручну у своєму коді, що надає йому більший контроль проте відчутно збільшує час розробки.

- Мультиплатформенна сумісність

Хоч в документації SwiftUI і міститься відомості про те, що додаток написаний з використанням даного фреймворку буде гнучко адаптуватися до будь-якого розміру екрану пристроїв Apple, включно з Apple Watch, iPhone або MacBook, на практиці це не зовсім так. Тож, коли розробник потребує перенесення коду написаного на SwiftUI для iPhone, на macOS, йому необхідно буде переконатися, що все працює належним чином у враховуючи налаштування нового пристрою.

- Віджети

З появою віджетів для систем включно з iOS 14 і вище, розробка додатків під iOS зазнала значних змін. Адже, тепер розробники потребують фреймворків або ж технологій, що дозволятимуть зручно і красиво адаптувати написані ними програми саме під віджети. З цією функцією чудово справляється SwiftUI, на відміну від UIKit, котрий не може надати розробнику необхідний для даного завдання функціонал.

Тож, враховуючи все вище сказане і відповідаючи на питання кращості того чи іншого фреймворку, хочу підсумувати, що вибір напряму залежатиме від задач проєкту, обмежень у часі, необхідності підтримувати старіші версії ОС та персонального бажання розробника самостійно опановувати новий матеріал,

будучи при цьому значно обмеженим у відповідях від спільноти на свої виникаючі питання.

У даній курсовій роботі було обрано використовувати саме UIKit, першочергово саме через наявність попереднього досвіду роботи з ним у доповідача, доступності документації та опису великої кількості проблем, з якими попередньо стикались інші розробники, в мережі.

3.2 Опис обраних засобів розробки

Оскільки дана курсова робота являє собою по суті ще одного клієнта для вже розробленої системи електронного поселення в гуртожитки DMS, то мною було використано спільне з веб версією системи API. Зручний перегляд якого зумовлювався використанням інструменту для документування REST сервісів Swagger, який вже містив описані ендпоінти, параметри запитів, та структурні моделі. Надсилання HTTP запитів з розробленого додатку відбувалося за допомогою бібліотеки з відкритим кодом Alamofire, котра забезпечує елегантне представлення інтерфейсу мережевого стеку Foundation, являючись обгорткою над URLSession, та спрощує роботу з запитами. Для надсилання сповіщень користувачу мною було використано нативну технологію Apple Push Notification Centre та сервіс Firebase Cloud Messaging, котрий спрощує обмін сповіщеннями між мобільними додатками та серверними програмами.

Розробка додатку здійснювалася в середовищі Xcode, котре містить увесь необхідний для розробки мобільних додатків функціонал. Інструменти розробки Xcode включають:

- можливість програмування мовами Swift та Objective-C;
- тестування за допомогою симулятора, що емулюють роботу великої кількості пристроїв компанії Apple, з широким рядом доступних ОС;
- тестування на реальному фізичному пристрої Apple;
- розробку графічних інтерфейсів з використанням фреймворку для зручного конструювання інтерфейсів UIKit, або ж SwiftUI.

Інтерфейс в даній курсовій роботі був розроблений за допомогою UIKit, через описаний в попередньому пункті ряд причин.

Для аутентифікації користувача в розробленому додатку DMS була використана бібліотека MSAL. Для підключення перелічених бібліотек використовувався інструмент CocoaPods.

Нижче наведена більш детальна інформація про певні вищезгадані інструменти:

- *Firebase Cloud Messaging*

Firebase Cloud Messaging (FCM) [7] — це безкоштовне кросс-платформенне рішення для обміну повідомленнями, яке дозволяє надсилати повідомлення надійно і підтримує ОС iOS з 2015 року. Це рішення, раніше знане як Google Cloud Messaging, застаріло і припинило своє існування в 2019, трансформувавшись у FCM.

- *MSAL (Microsoft Authentification Library)*

MSAL [5] –дозволяє розробникам отримувати токени з платформи Майкрософт, для доступу до захищених веб-API. Використовується для безпечного доступу до Microsoft Graph, інших Майкрософт API, веб-API сторонніх ресурсів, або власного веб-API. MSAL надає безліч способів отримання маркерів з використанням узгодженого, для кількох платформ API. Використання MSAL надає наступні переваги:

  - Забезпечує отримання токенів від імені користувача або програми (коли застосовується до платформи).
  - Не потребує безпосередньо використання OAuth бібліотеки або написання коду, що потребує певний протокол.
  - Підтримує кешування та оновлення токенів, з спливаючим терміном ді. Тож розробнику немає необхідності регулювати закінчення терміну дії токена, вручну.
  - Надає доступ до входу під аккаунтом певної організації, груп організацій, особистих облікових записів Майкрософт, посвідчення

соціальних мереж за допомогою Azure AD B2C, користувачі в незалежних та національних хмарах).

- Допомагає настроїти програму з конфігураційних файлів.

- *Cocoapods*

Це менеджер залежностей з відкритим кодом на рівні програми для Swift та Objective-C, розроблений в 2011 році,[18] що надає стандартний формат для керування зовнішніми бібліотеками. CocoaPods дає змогу встановлювати залежності для програми за допомогою специфікації залежностей, а не методом ручного копіювання вихідних файлів, та зосереджується на розповсюдженні коду сторонніх розробників на основі вихідних кодів та автоматичної інтеграції в проекти Xcode.

3.3 Опис структури програми

Оскільки реалізований додаток побудований на основі архітектурного шаблону MVVM, то містить усі характерні для нього розбиття на модулі(Див рис. 3.3.1)

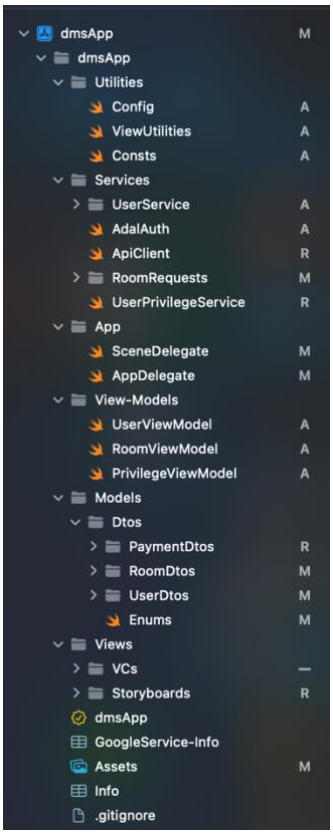

Рис. 3.3.1 Структура проєкту DMSapp

- *Models*

Містить структурні моделі об’єктів необхідні для подальшого використання в програмі та поточного кодування/декодування в JSON об’єкти відповідних HTTP запитів до API. Структурні моделі об’єктів представлені за допомогою шаблону проектування DTO (Data Transfer Object) [19], котрий визначає який вигляд матимуть передані між сервером і клієнтом структури.

- Views

Містить розбиття на представлення графічного інтерфейсу користувача, використовуючи файли .storyboards (папка storyboards), та viewController’и, що відповідають за обробку логіки представлення кожного окремого екрану.

- View-Models

Відповідають за, як було описано в розділі 2.4, зв’язування об’єктів моделі з їх майбутнім представленням спочатку у viewController’і, а потім, враховуючи обробку viewController’а, у View.

- Services

Тут міститься вся бізнес логіка роботи з HTTP запитами з безпосереднім розбиттям по папкам ендпоінтам.

- App

Містить файли що відповідають за поведінку додатку в перші секунди завантаження, та обробку поведінки додатку в неактивних станах.

- Utilities

Папка містить файли з конфігураційними налаштуваннями для аутентифікації користувача, з константами, загальними для всієї програми, і файл з загальними налаштуваннями View об’єктів

3.4 Створення об'єктів і розробка головної програми

Після аналізу веб версії системи електронного поселення DMS та предметної області був розроблений макет майбутнього додатку в Figma (див. Додаток Б). Необхідно було зберегти впізнаваний стиль, до якого користувачі встигли звикнути за роки користування веб-версією, адаптувавши його під HIG [14].

Наступним кроком розробки стало обрання інструментів, методів та архітектурних рішень, котрі гарно виконували б поставлені перед ними і додатком в цілому задачі. Таким чином було обрано описані раніше бібліотеки, MSAL, Alomofire, та сервіс Firebase Cloud Messaging. Далі необхідно було впровадити аутентифікацію за допомогою бібліотеки MSAL (див. Додаток В).

Наступним кроком було написання DTO структур, необхідних для запитів, в папці Models, розбитій на відповідні до оброблюваних API структур підпапки. Створення DTO дозволяло б енкапсулювати в HTTP запитах, для надсилання або отримання, необхідні для роботи структури об’єкти, такі як AboutUserDto або RoomRequestDto (див. Додаток Г).

Щодо класу ApiClient, спроектованого за допомогою шаблону Singleton [20] то його створення було наступним етапом розробки (див. Додаток Д). В класі ApiClient, з використанням методу бібліотеки Alomofire, реалізовано метод req (див. Додаток Е), що надає клієнтському коду зручний інтерфейс надсилання запиту. В цьому ж класі та з використанням вище зазначеного методу, мною було реалізовано загальні методи GET та POST HTTP запитів (див. Додаток Є), що по суті являються обгорткою над методом req та відправляють, відповідно до своїх назв конкретні запити на сервер. Саме ці методи, використовуються в усій програмі при подальшій необхідності відправляти запити на конкретні ендпоінти. Мої методи GET та POST, як і загальний метод req, потребують передачі в свої параметри лише ендпоінту, колбеків та хедерів, що значно спрощує подальшу реалізацію запитів в папці Service, не потребуючи щоразового виклику методів Alomofire з коробки, що досить часто мають занадто громіздку структуру. Щодо, вищезгаданої папки Service, то там містяться запити для конкретних ендпоінтів DMS API.

Наступним кроком стала розробка екранів користувацького View, що містяться в однойменній, описаній раніше папці та конфігурація відповідних для кожного екрану UIViewController’ів.

3.5 Опис роботи готової програми

1. Користувач аутентифікується за допомогою корпоративної пошти викликаючи Microsoft Graph Api, та отримує токен який підставлятиметься в HTTP заголовки майбутніх запитів користувача
2. Надалі користувачу доступні 3 екрани (зліва направо) : екран Переселень, екран Поселень, екран Оплат. Екран Переселень в реалізованому, в даній курсовій роботі, додатку наразі як описувалося раніше, є недоступними, проте планується в найближчому релізі.
3. На сторінці Поселень користувачу доступні такі функції як: перегляд документів для поселення та безпосереднє створення запиту на поселення
4. На сторінці Оплат користувач може створити оплату для вже попередньо створеного відповідного room request'у. Змога безпосереднього переходу в клієнт банкінг з подальшою оплатою також буде доступна в наступному релізі.
5. З усіх екранів додатку(Переселення, Поселення, Оплата) користувачу доступне меню, появи якого він може досягти кліком по іконці юзера справа від лого додатку. В меню користувачу доступні такі функції як перегляд інформації про себе, перегляд та додавання пільг та вихід із поточного аккаунту.
6. На сторінці інформації про себе користувач має змогу переглядати ПІБ, стать, факультет, спеціальність, рік навчання, та тип навчання(державне/контракт).
7. Сторінка пільг же, у свою чергу, дає можливість переглядати/видаляти наявні пільги та додавати нові з попередньо окресленого переліку.
8. Пункт меню Вихід з аккаунту, поверне користувача на екран логіну, даючи змогу обрати новий аккаунт для входу.

Висновки

Результатом написання даної курсової роботи є MVP додаток DMS, що являє собою мобільний клієнт однойменної системи автоматизованого поселення до гуртожитків, яка до цієї роботи мала лише веб клієнт, для ОС iOS. З реалізованою можливістю аутентифікації користувача через Microsoft Authentification Library MSAL, надсиланням Push Notification користувацьких сповіщень за допомогою Firebase Cloud Messaging. Реалізований проєкт складається з Xcode репозиторію, що представлений на GitHub (див. Додаток Ж).

Першочерговою задачею в майбутній роботі над проєктом, що я для себе ставлю, є розширення існуючого обмеженого функціоналу додатку до рівня веб-версії DMS, шляхом впровадження таких ключових, наявних у веб-версії, дій як “Переселення”, “Оплата за проживання” та пов’язаних з їх роботою функцій. Також, бажаним варіантом подальшого розширення функціоналу додатку є знову ж таки додавання, наявної у веб-версії DMS, авторизації користувача. Авторизації, яка в свою чергу давала б змогу, за наявності у користувача певної ролі, такої як: член ради гуртожитку, комендант, декан, адмін системи, розширити функціонал додатку до можливості перегляду користувачем адмін-панелі з відображенням в ній відповідних до ролі дій. Ці дії включали б переключення режимів системи(поселення/переселення), схвалення/відхилення студентських запитів на переселення, менеджмент будівель/кімнат гуртожитків, редагування існуючих користувачів, тощо.

Оскільки такою омріяною для кожного iOS розробника є подія публікації розробленого застосунку в AppStore з подальшою підтримкою і оновленням до нових версій, тож і реалізований мною iOS додаток DMS не стане виключенням і в моїх планах на його майбутнє міститься пункт про успішне розміщення на вищезгаданій платформі.

Список використаної літератури

1. Turner A. HOW MANY SMARTPHONES ARE IN THE WORLD? [Електронний ресурс] / Ash Turner – https://www.bankmycell.com/blog/how-many-phones-are-in-the-world - sources
2. S. O'Dea. Number of smartphone subscriptions worldwide from 2016 to 2027 [Електронний ресурс] / S. O'Dea. – 2022. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
3. Mobile Vs. Desktop Internet Usage (Latest 2022 Data) [Електронний ресурс] – https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics
4. Sagara Technology Idea Lab. Mobile Apps vs Web Apps: Which is the Better Option? [Електронний ресурс] / Sagara Technology Idea Lab. – 2019. – https://sagaratechnology.medium.com/mobile-apps-vs-web-apps-which-is-the-better-option-868106c88730
5. Microsoft. Overview of the Microsoft Authentication Library (MSAL) [Електронний ресурс] / Microsoft. – 2019. – https://docs.microsoft.com/uk-ua/azure/active-directory/develop/msal-overview
6. Framework User Notifications [Електронний ресурс] – https://developer.apple.com/documentation/usernotifications
7. Firebase Cloud Messaging [Електронний ресурс] – https://firebase.google.com/docs/cloud-messaging
8. Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
9. Simple Object Access Protocol (SOAP) [Електронний ресурс] – 2000. – https://www.w3.org/TR/soap/
10. XML WSDL [Електронний ресурс] – https://www.w3schools.com/xml/xml_wsdl.asp
11. A Brief, Opinionated History of the API [Електронний ресурс] – https://www.infoq.com/presentations/history-api/

12. What is a REST API? [Електронний ресурс] – https://www.redhat.com/en/topics/api/what-is-a-rest-api
13. iPhone push notification service for devs announced [Електронний ресурс] – https://www.engadget.com/2008-06-09-iphone-push-notification-service-for-devs-announced.html
14. Design Principles [Електронний ресурс] // Apple https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/
15. "Massive" View Controllers or bad coding style? [Електронний ресурс] // Medium Ivan Zmerzlyi. – 2019. https://medium.com/flawless-app-stories/massive-view-controllers-or-bad-coding-style-bf2b0d57c268
16. Gossman J. Introduction to Model/View/ViewModel pattern for building WPF apps [Електронний ресурс] / John Gossman. – 2005. – https://web.archive.org/web/20110612072619/http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx
17. Apple Announces Declarative SwiftUI Framework for Leaner, Faster, Interactive App Development [Електронний ресурс] – https://www.infoq.com/news/2019/06/swiftui-announcement-wwdc-2019/
18. CocoaPods [Електронний ресурс] – https://github.com/CocoaPods/CocoaPods
19. The DTO Pattern (Data Transfer Object) [Електронний ресурс] // by baeldung – https://www.baeldung.com/java-dto-pattern
20. Одинак на Swift [Електронний ресурс] – https://refactoring.guru/uk/design-patterns/singleton/swift/example#example-1

Додатки

Додаток А. Посилання на веб-версію системи електронного поселення до гуртожитків “Dormitory Management System” DMS

https://dms.ukma.edu.ua/

Додаток Б. Посилання на макет проєкту в Figma

https://www.figma.com/file/YJaB0UEdHDItDg1zAWFhre/DMS?node-id=0%3A1

Додаток В. Код з використанням MSAL автентифікації для отримання користувачем токену

```
func acquireTokenInteractively() {

    guard let applicationContext = self.applicationContext else { return }
    guard let webViewParameters = self.webViewParameters else { return }

    // #1
    let parameters = MSALInteractiveTokenParameters(scopes: Config.kScopes, webviewParameters: webViewParameters)
    parameters.promptType = .selectAccount

    // #2
    applicationContext.acquireToken(with: parameters) { (result, error) in

        // #3
        if let error = error {

            self.updateLogging(text: "Could not acquire token: \(error)")
            return
        }

        guard let result = result else {

            self.updateLogging(text: "Could not acquire token: No result returned")
            return
        }

        // #4
        self.accessToken = result.accessToken
        self.updateCurrentAccount(account: result.account)
        Config.keychain.set(self.accessToken, forKey: "token", withAccess: .accessibleWhenUnlocked)
        self.getContentWithToken()

    }
}
```

Даний фрагмент коду та впровадження аутентифікації за допомогою бібліотеки MSAL в даній курсовій роботі були виконані за допомогою туторіалу від Майкрософт (посилання: https://docs.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-ios)

Додаток Г. Приклади DTO структур AboutUserDto та RoomRequestDto

- ***UserInfo***

```
struct AboutUserDto: Codable {

    let id: Int
    let email: String
    let dateBirth: String
    let degree: EnumDto
    let gender: EnumDto
    let faculty: EnumDto
    let speciality: EnumDto
    let educationType: EnumDto
    let privileges: [UserPrivilegeDto]

    enum CodingKeys:String, CodingKey {

        case id
        case email
        case dateBirth
        case degree
        case gender
        case faculty
        case speciality
        case educationType
        case privileges
    }

     init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy: CodingKeys.self)
        self.id = try values.decode(Int.self, forKey: .id)
        self.email = try values.decode(String.self, forKey: .email)
        self.dateBirth = try values.decode(String.self, forKey: .dateBirth)
        self.degree = try values.decode(EnumDto.self, forKey: .degree)
        self.gender = try values.decode(EnumDto.self, forKey: .gender)
        self.faculty = try values.decode(EnumDto.self, forKey: .faculty)
        self.speciality = try values.decode(EnumDto.self, forKey: .speciality)
        self.educationType = try values.decode(EnumDto.self, forKey: .educationType)
         self.privileges = try values.decode([UserPrivilegeDto].self, forKey: .educationType)
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(id, forKey: .id)
        try container.encode(email, forKey: .email)
        try container.encode(dateBirth, forKey: .dateBirth)
        try container.encode(degree, forKey: .degree)
```

```
try container.encode(gender, forKey: .gender)
try container.encode(faculty, forKey: .faculty)
try container.encode(speciality, forKey: .speciality)
try container.encode(educationType, forKey: .educationType)
try container.encode(privileges, forKey: .privileges)
```

- ***RoomRequestDto***

```
struct RoomRequestDto : Codable{

    let cancellable: Bool
    let evictable: Bool
    let extendable: Bool
    let id: Int
    let paymentRequestable: Bool
    let payments : [PaymentDto]
    let room: RoomDto
    let status : EnumDto
    let validSince : String
    let validTo : String

    enum CodingKeys:String, CodingKey {
        case cancellable
        case evictable
        case extendable
        case id
        case paymentRequestable
        case payments
        case room
        case status
        case validSince
        case validTo
    }

    init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy: CodingKeys.self)
        self.cancellable = try values.decode(Bool.self, forKey: .cancellable)
        self.evictable = try values.decode(Bool.self, forKey: .cancellable)
        self.extendable = try values.decode(Bool.self, forKey: .cancellable)
        self.id = try values.decode(Int.self, forKey: .id)
        self.paymentRequestable = try values.decode(Bool.self, forKey: .paymentRequestable)
        self.payments = try values.decode([PaymentDto].self, forKey: .payments)
        self.room = try values.decode(RoomDto.self, forKey: .room)
        self.validSince = try values.decode(String.self, forKey: .validSince)
        self.validTo = try values.decode(String.self, forKey: .validTo)
        self.status = try values.decode(EnumDto.self, forKey: .status)
    }


    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(cancellable, forKey: .cancellable)
try container.encode(evictable, forKey: .evictable)
try container.encode(extendable, forKey: .extendable)
try container.encode(id, forKey: .id)
try container.encode(paymentRequestable, forKey: .paymentRequestable)
try container.encode(payments, forKey: .payments)
try container.encode(room, forKey: .room)
try container.encode(validSince, forKey: .validSince)
try container.encode(validTo, forKey: .validTo)
try container.encode(status, forKey: .status)
```

Додаток Д. Реалізація шаблону Singleton (виділений рядок) для ApiClient класу

```
class ApiClient {
    private let decoder = JSONDecoder()
    private static let shared = ApiClient()
    private init() { }
    static func getInstance() -> ApiClient{
        self.shared
    }
    private func createHeaders() -> HTTPHeaders? {
        if let token = Config.keychain.get("token") {
            return ["Authorization" : "Bearer " + token]
        }else {return nil}
    }
    private func req<T:Decodable>(url:String,
                                  params:[String : Any]?,
                                  method:HTTPMethod,
                                  encoding:ParameterEncoding,
                                  onSuccess:@escaping (T) -> Void,
                                  onFail: @escaping (MessageResponse) -> Void) {
        let headers = createHeaders()
        AF.request(Config.BASE_URL + url,
                   method: method,
                   parameters: params,
                   encoding: encoding,
                   headers: headers!,
                   interceptor: nil).responseDecodable { (response : DataResponse<T, AFError>) in
       if let status = response.response?.statusCode  {
                if 200..<300 ~= status {
                    var res: T
                    do{
                        res = try self.decoder.decode(T.self, from: response.data!)
                    }
                    catch {
                        print("Error in onSuccess part", error)
                        return
                    }

                    onSuccess(res)
```

Додаток Е. Реалізація методу req класу ApiClient

```
private func req<T:Decodable>(url:String,
                          params:[String : Any]?,
                          method:HTTPMethod,
                          encoding:ParameterEncoding,
                          onSuccess:@escaping (T) -> Void,
                          onFail: @escaping (MessageResponse) -> Void) {

    let headers = createHeaders()
    AF.request(Config.BASE_URL + url,
               method: method,
               parameters: params,
               encoding: encoding,
               headers: headers!,
               interceptor: nil).responseDecodable { (response : DataResponse<T, AFError>) in

        if let status = response.response?.statusCode  {
            if 200..<300 ~= status {
                var res: T
                do{
                    res = try self.decoder.decode(T.self, from: response.data!)
                }
                catch {
                    print("Error in onSuccess part", error)
                    return
                }
                onSuccess(res)
            } else {
                var res: MessageResponse

                do{
                    res = try self.decoder.decode(MessageResponse.self, from: response.data!)
                }
                catch {
                    print(error)
                    return
                }
                onFail(res)
            }
        }
    }
}
```

Додаток Є. . Реалізація методів get та post класу ApiClient

```
 func get<T:Decodable>(url:String,
                       params:[String : Any]?,
                       onSuccess: @escaping (T) -> Void,
                       onFail: @escaping (MessageResponse) -> Void) {
    self.req(url:url, params:params, method:.get, encoding:URLEncoding.default, onSuccess:onSuccess, onFail:onFail)
}

 func post<T:Decodable>(url:String,
                              params:[String : Any]?,
                              onSuccess: @escaping (T) -> Void,
                              onFail: @escaping (MessageResponse) -> Void) {
    self.req(url:url, params:params, method:.post, encoding:JSONEncoding.default, onSuccess:onSuccess, onFail:onFail)
}
```

Додаток Ж. Посилання на репозиторій проєкту на GitHub

https://github.com/SofiXeno/DmsApp