Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Мобільний додаток для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту»

Виконав: студент 2 курсу, групи 1АКІТ-21м, спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології» .

_____ Середюк Г.В.

Керівник: к.т.н., проф.. кафедри АІІТ

_____ Паламарчук Є.А.

«____» _____ 2022 р.

Опонент: _____

_____

«____» _____ 2022 р.

Допущено до захисту
Завідувач кафедри АІІТ

_____ д.т.н., проф. Бісікало О.В.
«____» _____ 2022 р.

Вінниця ВНТУ - 2022 рік

АНОТАЦІЯ

УДК 004.9

В даній роботі розглянуто питання, пов’язане з розробкою програмного забезпечення для мобільних пристроїв, взаємодії такого роду програмного забезпечення з камерами 360 град. для збирання даних у вигляді фотографій та обробкою їх з використанням штучного інтелекту в сфері будівельного бізнесу для удосконалення існуючих процесів будівництва, підвищення ефективності роботи працівників на будівельних майданчиках, комунікації між забудовником та замовником.

Досліджено актуальність використання мобільного програмного забезпечення в сфері будівельного бізнесу, як автоматизованої системи для контролю будівництва. Здійснено аналіз аналогів на торгових платформах та приводяться приклади їх впровадження.

Проаналізовано існуючі методи, підходи та інструменти для розробки мобільних додатків.

Розроблено власний підхід для вирішення досліджених проблем, алгоритми, функції, класи, методології, які лягли в основу розробленого програмного забезпечення. Проведено тестування та оптимізацію роботи розробленої програми на основі отриманих даних з тест кейсів.

Висновок відповідає на питання актуальності використання програмного забезпечення в сфері будівництва та описує вирішені задачі.

ABSTRACT

UDC 004.9

This paper considers the issue related to the development of software for mobile devices, the interaction of such software with 360-degree cameras for collecting data in the form of photographs and processing them using artificial intelligence in the field of construction business to improve existing construction processes, increase the efficiency of workers on construction sites, communication between the developer and the customer.

The relevance of using mobile software in the construction business as an automated system for construction control is investigated. The analysis of analogues on trading platforms is carried out and examples of their implementation are given.

Existing methods, approaches and tools for developing mobile applications are analyzed.

Developed own approach to solve the studied problems, algorithms, functions, classes, methodologies that formed the basis of the developed software. Testing and optimization of the developed program based on the data obtained from the test cases was carried out.

The conclusion answers the question of the relevance of the use of software in the field of construction and describes the solved problems.

ЗМІСТ

ВСТУП

Актуальність. Будівельна галузь визнана життєво важливим елементом в економіці та інших сферах будь-якої країни. Будівництво - це галузь народного господарства, що забезпечує зведення та реконструкцію житлових, громадських і виробничих будівель і споруд, створює базу для розвитку всіх галузей народного господарства. Можна виділити декілька головних загальних етапів будівництва:

- Вибір земельної ділянки.
- Проектування майбутньої споруди.
- Старт виконання робіт з фундаменту.
- Додавання комунікацій(водопостачання, опалення, електроенергія).
- Будівельні і монтажні роботи.
- Етап дизайну приміщень.

Попри стрімкий розвиток галузі та технологій, які використовуються на будівництвах, все ще актуальні такі проблеми, як дефіцит кваліфікованої робочої сили, висока конкуренція, низька рентабельність, якість робіт та збільшений ризик перевищення термінів будування. Раніше наведені проблеми, відтворюються в таких аспектах та ситуаціях, як перевищення лімітів всіх термінів та бюджету, дані із будівництва замовник отримує із запізненням у вигляді великої кількості звітів у паперовому вигляді, якість роботи перевіряється, тільки якщо є змога потрапити на будівельний майданчик, після закінчення будівництва, замовник може витратити надлишкові кошти на матеріали, зарплату для працівників.

Таким чином питання розробки нових підходів до вирішення вище наведених проблем є актуальним на сьогодні. Рішенням цих проблем може бути використання апаратно-програмного забезпечення для автоматизації процесів на будівельних майданчиках та швидкого доступу до актуальної інформації об’єктів на базі отриманих даних. Також існує розрив між існуванням актуального програмного забезпеченням та його використанням. Тому, метою тут є не лише створення нового програмного забезпечення, але й повне використання існуючого

Мета і завдання дослідження є побудова ідеології, алгоритму та програмно-апаратної реалізації системи для швидкого збирання, зберігання інформації про параметри стану будівельного об’єкту, обробки цих даних з використанням штучного інтелекту та надсилання цих даних до серверної частини для покращення процесів різного виду робіт на будівельному майданчику.

Для досягнення поставленої мети необхідно розв’язати наступні задачі:

- Проаналізувати існуюче програмне забезпечення в сфері будівництва та розглянути відомі підходи розробки програмного забезпечення на мобільні пристрої.
- На основі аналізу відомих існуючих аналогів та загальних підходів розробки програмного забезпечення на мобільні пристрої запропонувати власний підхід до вирішення актуальної проблеми і обґрунтувати його ефективність.
- Розробити програмне забезпечення на основі запропонованого підходу.
- Провести тестування та дослідження розробленого ПО.

Об'єктом дослідження є процес вирішення актуальних проблем в процесі будівництва та на будівельних майданчиках використовуючи апаратно-програмне забезпечення для мобільних пристроїв, а також його розробка згідно тенденцій сучасного програмного забезпечення та роботи з камерами 360 град. для збирання даних безпосередньо з будівельного майданчика з подальшою обробкою цих даних з використанням нейронної мережі.

Предметом дослідження є актуальні методи, засоби та технології створення додатків для мобільних пристроїв.

Методи дослідження. В процесі дослідження застосовуються методи поширення мобільних додатків на торгових платформах, методи та методології розробки ПО, методи використання штучного інтелекту а саме нейронної мережі, методи використання локальних сховищ, методи взаємодії додатку із сервером, методи інтегрування зовнішніх бібліотек, методи роботи додатку з камерами 360 град., методи розробки користувальницького інтерфейсу та методи тестування програм.

Науково - технічний результат.

1. Запропоновано удосконалений підхід до реалізації системи для швидкого збирання, зберігання інформації про параметри стану будівельного об’єкту, обробки цих даних з використанням штучного інтелекту та надсилання цих даних до серверної частини.
2. На основі запропонованого підходу розроблено апаратно-програмне забезпечення, яке на відміну від існуючих дає змогу покращити ефективність робіт на будівництві майже в усіх його аспектах, підвищити рентабельність та спростити процес контролю за будівництвом.

Апробація результатів. Результати досліджень були представлені на конференції: «XV міжнародної науково-практичної конференції “ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА АВТОМАТИЗАЦІЯ – 2022”(жовтень 2022) [1].

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасний стан та тенденції будівельної сфери

Ефективне функціонування реального сектору економіки та вирішення питань соціального розвитку неможливе без розв’язання системних проблем, які впливають на усі складові національної економіки – зокрема будівництво. У результаті впливу кризових процесів відбулися істотні зміни, які, в першу чергу, стосуються скорочення обсягів будівництва житлових та промислових об’єктів, зменшення кількості інвестицій в галузь, брак робочої сили та навичок [4].

Тенденції світової будівельної галузі у 2022-2023 роках включають в себе в цілому сприятливі економічні перспективи, а також деякі проблеми, які, як очікується, збережуться. Згідно з прогнозами Oxford Economics і ConstructConnect, у США Американські інститути архітекторів прогнозують зростання нежитлового будівництва на 4,6% у 2022 році, в той час як у житловому секторі очікується зростання на 9%. У Європейському союзі будівництво зросте з 1,5% в Іспанії до 3% у Франції та Нідерландах. Середнє зростання по ЄС прогнозується на рівні 2,7%, а в Сполученому Королівстві - 6,3%. За прогнозами Центру будівельної аналітики, у 2022 році будівництво в усьому світі зросте на 3,7%, причому найбільша частка зростання припаде на Азіатсько-Тихоокеанський регіон і Китай. Найшвидше зростання очікується в країнах Африки на південь від Сахари, включно з Ефіопією та країнами Східної Африки.[5].

1.2 Опис процесів, що протікають на будівництві

Процес будівництва - це не тільки те, як щось будується, це всі етапи, пов'язані зі створенням чого-небудь, від вибору майданчика до остаточної перевірки. Такий процес вимагає організації кількох підрядників, дорогого і небезпечного обладнання, великого будівельного майданчика, власника і графіка [6], [7].

Процес будівництва можна розділити на 6 основних частин:

- Концепція.

  Планування і розробка - це початок процесу будівництва. Задача цього етапу полягає в описі клієнтом будівлі або об'єкта, який він хоче. Планування - це етап, на якому ідеї найбільш мінливі, але він також створює основу для процесу будівництва. Частини цього кроку включають у себе пошук нерухомості для будівництва, попередні проєкти первісної концепції та вибір архітектора і, можливо, генерального підрядника.
- Дизайн.

  Процес проектування - це коли неможливі (або принаймні неймовірно дорогі) мрії клієнта зустрічаються з тим, що насправді реально та може бути зроблено. Вперше можна побачити, який вигляд матиме проєкт, коли його буде завершено. Після складання ескізного проєкту доводиться долати додаткові обмеження та регламенти, щоб задовольнити ідеї замовника. Наприклад, стандарти екологічної сертифікації для зеленої будівлі.
- Підготовка до будівництва.
- Вихід в попередню конструкцію. Це етап "підготовки до будівництва". Прийнято пропозицію від будівельної компанії або підрядника і почато проєкт.
- Закупки.

  Закупівлі - це просто купівля (або оренда) всього необхідного для будівельного проєкту. У будівництві це означає пошук робочої сили, обладнання та будівельних матеріалів.
- Будівництво.

  Проєкт, переходить із паперу (креслення САПР) у фізичний стан. На цьому етапі потрібно узгодити більшість робочих частин і термінів. Кожен залучений підрядник і субпідрядник повинен бути вчасно і слідувати планам, щоб все працювало. Саме на етапі будівництва клієнти можуть обдумувати використання цифрових інструментів управління процесами. Створення

робочих процесів проєкту для кожної команди та етапу процесу дуже важливе, але ще більш необхідно забезпечити безперебійну роботу.

- Здача об’єкта.

Усе побудовано, але ще не закінчено. Потрібно все оглянути. Остаточний контрольний список, званий списком недоліків проєкту, має бути підписаний до того, як клієнт в'їде. Ця остаточна покрокова перевірка перевіряє, що все в кінцевому підсумку було завершено правильно. Для цього архітектор видає сертифікат про суттєве завершення, потім урядова перевірка. Тільки після того, як остаточну державну перевірку буде завершено, проєкт може бути закінчено.

1.3 Аналіз управління будівництвом

Управління будівельним проєктом можна визначити як напрямок, регулювання та нагляд за проєктом від раннього будівництва до завершення. Кінцевою метою управління будівельним проєктом є повне задоволення вимог клієнта до життєздатного проєкту як з точки зору функціональності, так і бюджету. Існує широкий спектр типів будівельних проєктів, таких як комерційні, житлові, промислові та важкі цивільні. Основна концепція управління будівельними проєктами тісно пов'язана з технічними параметрами, такими як бюджет і виконання, але також вимагає міцного зв'язку між усіма учасниками (зацікавленими сторонами, підрядниками, спільнотою) [8], [9].

1) На першому етапі визначають мету та здійсненність проекту, створюється документ про ініціацію проєкту (PID). Документ про ініціацію проєкту забезпечує основу для плану будівництва і є одним із найважливіших артефактів в управлінні проєктом.

2) На етапі планування проєкту команда виділяє всю роботу, яку необхідно виконати. Це безперервна діяльність майже до кінця проєкту. Основним пріоритетом на етапі планування є планування часу, витрат і ресурсів для проєкту. На основі цих вимог команда розробляє стратегію, якої необхідно дотримуватися.

Це також відомо як управління областю дії. Іншим важливим документом, який необхідно підготувати, є структура розподілу робіт (WBS) [10], контрольний список, який ділить усю необхідну роботу на більш дрібні більш функціональні категорії. Щойно бюджет, графік і роботу визначено, проєкт практично готовий до запуску.

3) Наступним кроком є управління ризиками. На цьому етапі команда повинна вивчити всі потенційні загрози для проєкту і знайти надійні рішення. Також необхідний план комунікації, оскільки він встановить ефективний потік інформації між зацікавленими сторонами проєкту.

4) На етапі виконання план управління будівельним проєктом приводиться в дію. Як правило, ця фаза ділиться на два основні процеси: виконавчий і контрольний. Команда проєкту стежить за тим, щоб необхідні завдання виконувалися. При цьому відстежується прогрес і вносяться відповідні зміни. Насправді менеджер проєкту проводить більшу частину часу на етапі моніторингу і залежно від одержуваної інформації переспрямовує завдання і зберігає контроль над проєктом [11]

Зібрати будівельний проєкт - це дійсно складне завдання. Існує безліч параметрів і елементів, які необхідно ретельно проаналізувати. Ось чому так важливо довірити управління проєктом програмному забезпеченню для управління будівництвом, яке полегшить життя і водночас надасть можливість вивести план будівництва на абсолютно новий рівень.

1.4 Сучасний стан діджиталізації будівельної сфери

Сфера будівництва - це одна з тих галузей, де діджиталізація поки що запізнюється. Опитування CIO 2021 повідомляє, що будівельні компанії, яким вдалося впровадити програмне забезпечення для управління будівництвом, спостерігали зростання ефективності операцій на 12%, якості обслуговування клієнтів на 14%, довіри клієнтів на 12% і досвіду співробітників на 11% [5], [13]. Проте більшість будівельних компаній продовжують ігнорувати цифрову трансформацію з наступних причин:

- Застарілі процеси. Будівельний бізнес - це історично розрізнена галузь, у якій бере участь безліч зацікавлених сторін, у яких є свої зони відповідальності та підходи до роботи. Ось чому дуже складно з'єднати різні сторони за допомогою єдиного наскрізного цифрового рішення.
- Людський фактор. Важко переконати будівельників, що використання цифрових інструментів значно полегшить виконання їхніх завдань. Люди, які звикли виконувати свою роботу вручну, не довіряють будівельному програмному забезпеченню, бо не мають базових знань про його використання у своїй професії.
- Вартість проєкту. Генпідрядники завжди шукають способи скоротити витрати й уникнути накладних витрат, а впровадження цифрових технологій означає додаткові витрати. Програмне забезпечення для управління будівництвом має чітко показувати, чому варто витрачати стільки грошей.
- Високі ризики. Кожен окремий будівельний проєкт пов'язаний із високими ризиками, оскільки безліч зовнішніх чинників можуть несподівано завадити розвитку і будівництву об'єкта. Оцифровка виконання будь-якої пов'язаної діяльності ризикована подвійно, тому що підрядник не може бути впевненим, що будівельне програмне забезпечення справді ефективне і не викличе жодних проблем.

Проте програмне забезпечення для будівельників дає змогу істотно розширити можливості компанії, поліпшити управлінські процеси й автоматизувати рутинні завдання. Крім того, сучасні технології дають змогу масштабувати компанію, приводять нових клієнтів і дають змогу випередити конкурентів [13], [14].

Програмне забезпечення в будівництві істотно розширює можливості компанії.

Що дає програмне забезпечення для будівництва ?

- Оптимізацію робочих процесів. Завдяки тому, що всі процеси видно і легко керовані, виходить робити більше за менший проміжок часу. У будівництві оптимізація процесів призводить до зростання ефективності та зниження витрат. Коли ставиться план завдань і контролюється його виконання в діджитал просторі, ефективність роботи підвищується.
- Підвищення продуктивності команди. Якщо кожен учасник процесу чітко знає, що йому потрібно виконати, які перед ним завдання і як досягти результату, робота виконується швидше, краще і дешевше для компанії. Зменшується ймовірність помилок, співробітники отримують додаткову мотивацію, а головне, взаємодія всередині команди правильно організована.
- Детальну звітність і можливість бачити всі процеси. Автоматизація системи управління будівництвом передбачає докладні звіти і контроль за цифрами. У цій ніші цифрові показники відіграють ключову роль. Складно на словах пояснити замовнику, чому потрібно розширити бюджет на 15%. Якісна система звітностей дасть змогу надати всі необхідні документи і цифри, наочно вкаже, куди витрачається бюджет. Крім того, ще на етапі планування проєкту обчислення виходять більш точними.
- Контроль ресурсів. Правильна організація ресурсів - запорука успішного ведення бізнесу. Важливо опрацювати систему, в якій машини і водії не будуть простоювати, навантаження розподіляється рівномірно, а бюджет максимально точно вираховується під проєкт. У цьому допомагає автоматизація системи управління будівництва, яка коректно розподіляє

людські ресурси з огляду на зовнішні чинники, починаючи від заторів на дорозі, закінчуючи обліком часу застигання цементу.

- Контроль бюджету. Система не тільки веде облік, а й допомагає зробити правильні розрахунки, прогнозувати витрати і враховувати неочевидні витрати. Контроль бюджету для будівельної компанії дає можливість розвивати бізнес і комфортно працювати з клієнтами, вибудовуючи прозорі відносини.
- Інструменти комунікації між партнерами, клієнтами та працівниками. Діджитал рішення дозволяють налагодити комунікацію між учасниками процесу. Застарілі підходи вимагають більше часу на ухвалення рішень, а сама комунікація не є якісною та ефективною. IT-рішення допомагають уникати непорозумінь і дають інструменти комунікації між командами.
- Автоматизацію роботи з документами. Управління проєктуванням і будівництвом неможливе без документообігу. Це один із головних і складних процесів. Діджиталізація документообігу виконує одразу кілька завдань: спрощує ведення звітностей, прискорює процеси ухвалення рішень і мінімізує ризики помилок через людський фактор. Більшість документів можна звести до шаблону, що спрощує рутинні завдання. Автозаповнення даних, електронні підписи, зберігання проєктної документації в цифровому вигляді та інші інструменти роботи з документацією роблять цей процес комфортним і безпечним.
- Автономність управління процесами. Управління проєктуванням і будівництвом може здійснюватися з мобільного пристрою, комп'ютера або ноутбука. Не обов'язково бути на місці будівництва, щоб контролювати робочі процеси. Розподіл обов'язків дає можливість автономно керувати робочими процесами і контролювати результат. Управління стає гнучким, навіть якщо один або кілька учасників процесу не перебувають на місці роботи, це не перериває робочі процеси [17].
- Ефективну оцінку роботи. Фінансові звіти, документообіг, управління бюджетом, розподіл ролей - усе веде до одного: оцінки ефективності роботи. Саме від неї залежить успіх компанії, як фінансовий, так і іміджевий. Програмне забезпечення для будівництва дає змогу оцінити результат

роботи, ґрунтуючись на попередньому досвіді, і побудувати систему роботи для поліпшення оцінки надалі. Так компанія буде постійно розвиватися і покращувати якість роботи. Приклад процесів, які можна покращити на будівництві з допомогою програмного забезпечення для вирішення актуальних проблем, зображено на рисунку 1.1.

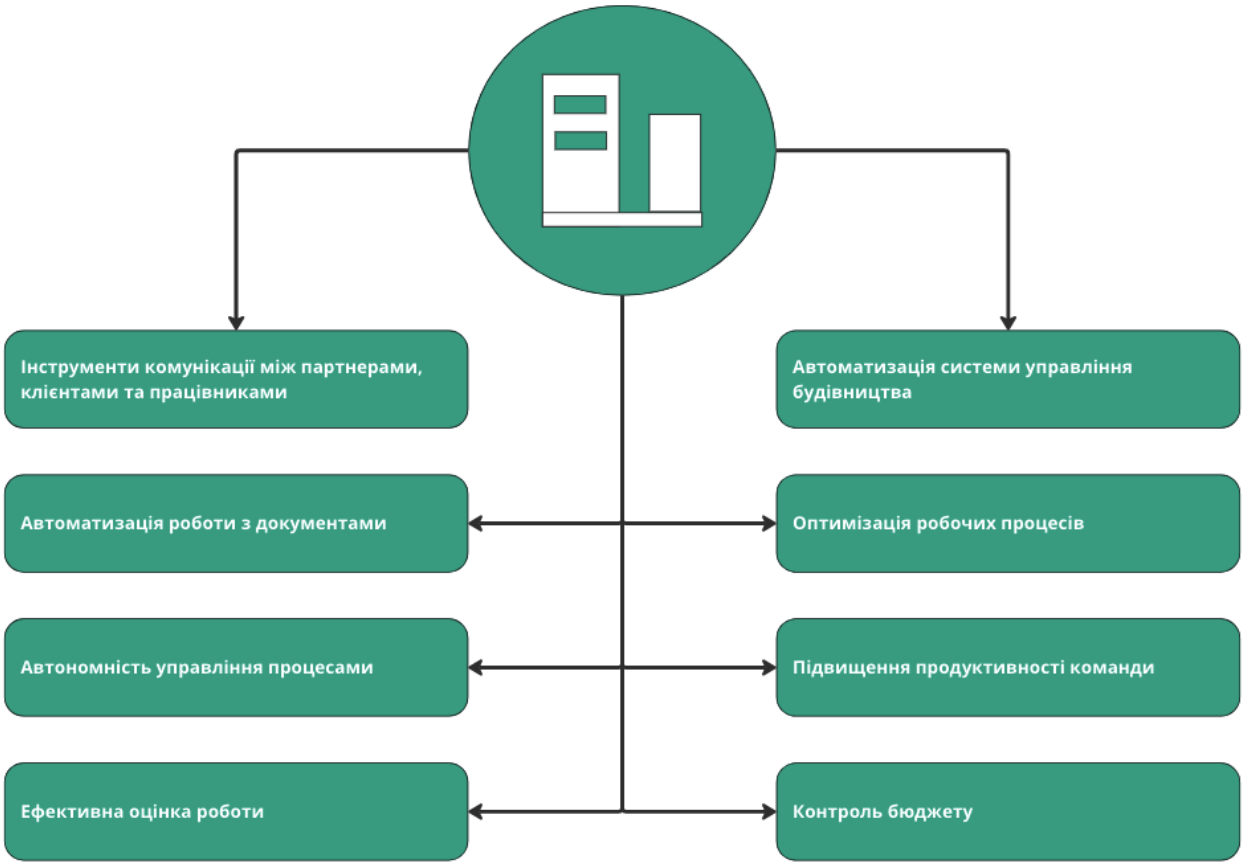


Рисунок 1.1 - Потенційні процеси будівництва для інтегрування ПЗ з метою їх удосконалення.

Тому впровадження сучасних технологій у будівельний бізнес значно розширює можливості компанії [16]. Надалі це стане невід'ємним процесом. Без програмного забезпечення компаніям буде дуже важко конкурувати. Сьогодні ж ви можете випередити конкурентів, впроваджуючи IT інструменти у свою сферу роботи [15].

1.5 Дослідження аналогів

Пошук аналогів проводився на спеціальному сервісі для дослідження мобільних застосунків AppMagic [2] за наступними критеріями:

- Ціна.
- Локальне сховище.
- Швидкість роботи.
- Передача даних в реальному часі.
- Швидкість роботи з великими об’ємами даних.

У таблиці 1.1 наведена порівняльна характеристика проаналізованих аналогів додатку, що розробляється

Таблиця 1.1 - Порівняння характеристик аналогів.

Критерії	**Timeero**	**MagicPlan**	**Plan Grid**	**GoCanvas**	**Drone Deploy**
Ціна	Середня	Висока	Середня	Середня	Низька
Локальне сховище.	+	+	+-	+-	-
Швидкість роботи.	Середня	Низька	Середня	Середня	Середня
Передача даних	+	+-	+-	+-	-
Швидкість роботи з великими об’ємами даних.	Середня	Середня	Низька	Середня	Низька

Такі аналоги мають свої недоліки та переваги.

Переваги Timeero:

- Автоматичний вхід/вихід і годинник входу з обмеженням за геозоною: співробітники автоматично реєструються, коли вони приходять або залишають робоче місце. Їм також може бути заборонено працювати, якщо вони не перебувають на робочому місці.
- Ще одна корисна функція - "Хто працює" - робота з мобільними командами може бути складним завданням, тому Timeroo показує, де знаходяться співробітники в будь-який час і над чим вони працюють. Це також дає змогу користувачам ділитися своїм місцем розташування, тож як менеджери, так і інші співробітники можуть бути в курсі того, хто на годиннику і де саме вони перебувають.

Недоліки Timeero:

- Збирається велика кількість приватних даних особи, яка використовує додаток.
- При відключення системи GPS на пристрої, більша частина функціоналу стає марна, тому що втрачається можливість моніторингу співробітників на робочих об’єктах, їх задачі та прогрес.

Переваги MagicPlan:

- Надає змогу швидко накреслити план будівлі, поверхів чи апартаментів.

- Можливість поширити іншому користувачу плани за короткий час.

Недоліки MagicPlan:

- Процес креслення стає незручним на телефонах з малим розширенням екрану, що в подальшому може призвести до хибних креслень чи розрахунків.

Після вище приведених переваг і недоліків програм, можна зробити висновок, що ідеальної програми не існує, але є ті, які максимально наблизились то вирішення поставлених задач, щоб покращити ті чи інші процеси на будівництві.

1.6 Висновок

В даному розділі підводяться підсумки по вивченню теоретичного матеріалу. Розкрито поняття будівництва, його стан та тенденціїї. Описані основні процеси будівельних робіт, методи та процеси управління, та стан використання програмного забезпечення на будівництві. Також були проаналізовані аналоги системи що використовуються сьогодні це складний процес, то важливо пам’ятати про вище наведені пункти. Вони здатні пришвидшити розробку, покращити якість і надійність і полегшити розробку загалом.

2 ОБҐРУНТУВАННЯ МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Обґрунтування вибору платформи реалізації

З пункту 1.5 можна зробити висновки, що аналоги систем які розробляються мають такі спільні недоліки як локальне сховище, передача даних в реальному часі, швидкодія обробки інформації.

Щоб вирішити вище наведені недоліки для розробки системи була вибрана форма сучасного нативного додатку на платформі iOS. Використовуючи нативний додаток для iOS ми отримаємо систему яка буде якісно, швидко працювати з спеціально розробленими нативними інструментами від компанії Apple, які постійно підтримуються та оновлюються, щоб не залежати від сторонніх компаній, які розробляють SDK чи фреймворки. Робота iOS системи з нативними інструментами, завжди швидша ніж додаток із сторонніми фреймворками, тому що інструменти спеціально розробляються на платформу iOS з дотриманням всіх специфікацій, правил, недоліків, переваг чи навіть версій системи [41], [42].

Також буде забезпечена швидкодія роботи з великою кількістю даних за рахунок інтегрування у систему нейронної мережі. Зберігання даних у великій кількості з використанням локального сховища. Передача даних на сервер з використанням REST API та приємного і інтуїтивно зрозумілого користувальницького інтерфейсу за рахунок використання нової технології SWIFTUI [22] , [23].

2.2 Обґрунтування вибору операційної системи

На сучасному ринку смартфонів є дві домінуючі платформи. Однією з них є платформа iOS від Apple Inc. Платформа iOS - це операційна система, на якій працює популярна лінійка смартфонів Apple iPhone. Другою - Android від Google. Операційна система Android використовується не лише пристроями Google, а й багатьма іншими OEM-виробниками для створення власних смартфонів та інших інтелектуальнихпристроїв.

Хоча між цими двома платформами є деяка схожість при створенні застосунків, розробка для iOS і розробка для Android припускають використання різних комплектів розроблення програмного забезпечення (SDK) і різних наборів інструментів розробки. Тоді як Apple використовує iOS виключно для своїх пристроїв, Google надає Android іншим компаніям за умови, що вони відповідають певним вимогам, таким як включення певних застосунків Google на пристрої, які вони постачають [19]. Розробники можуть створювати додатки для сотень мільйонів пристроїв.Порівняльні характеристики iOS та Android [32] , [33] :

1)Продуктивність. За рахунок високої оптимізації апаратних та програмних процесів, iPhone старших поколінь мають тенденцію перемагати нові флагмани Android в тестах. В цілому, в той час як телефони Android майже завжди мають чудові апаратні характеристики на папері, iPhone пропонують кращу продуктивність в переважній більшості випадків.

2) Дисплей. Більшість сучасних телефонів будь то Android або iOS, бюджетний або флагманський, використовують IPS LCD. Включають в себе всі iPhone до iPhone 8 , а також iPhoneXR , при цьому iPhone починаючи з X до 14 моделі є оснащеними OLED. Що стосується телефонів Android, більшість OEM-виробників вважають за краще IPS замість OLED для більшості своїх пристроїв, за винятком Samsung.

IPS LCD, і OLED мають свої переваги і недоліки. З технічної точки зору OLED має ряд переваг:

1. Це більш енергоефективні. При використанні OLED-дисплеїв кожен піксель підсвічується індивідуально, це означає, що для дисплея не потрібне активне підсвічування всього екрану при його включенні.

2. В OLED кращі кути огляду. Дисплеї IPS більш високої якості, такі як ті, що можна побачити в iPhone, можуть майже відповідати OLED в цьому відношенні, хоча різниця все ще помітна.

3. OLED краще відображає світло ніж IPS. Особливо це важливо для людей, які проводять багато часу на вулиці і не хочуть, щоб відблиски заважали їм.

3) Програми. Android безумовно бере гору над iOS, коли мова йде про кількість додатків, доступних в App Store. Цей розрив пов'язаний головним чином з природою Android — відкритим вихідним кодом і слабкою політикою Google Play. Крім того, пристрої Android також дозволяють користувачам отримувати доступ до сторонніх магазинів. Пристрої Android на відміну від iOS також дозволяють користувачеві завантажувати додатки вручну, вставляючи «файли apk», що може привести до встановлення шкідливого програмного забезпечення.

Хоча Android в порівнянні з iOS може лідирувати в плані кількості, проте строгий контроль Apple над додатками гарантує, що всі вони функціонують належним чином, та є повністю безпечними [24]. [35].

4) Оновлення. Як згадувалося раніше, пристрої Apple відомі своєю довговічністю, і при цьому продуктивність практично не знижується. Основна причина цього терміну служби полягає в тому, що кожен пристрій iOS отримує близько 5 років постійних оновлень ОС після свого початкового випуску.

Спираючись на вищенаведені характеристики для вирішення задач швидкодії та оптимізації процесів, високої якості програми, було обрано систему iOS .

2.3 Обґрунтування вибору середовища розробки мобільних додатків

Оскільки було вирішено, що додаток буде розроблятись на платформі iOS, яка працює на обладнанні iPhone, iPad, iPod Touch. Apple надає інструменти та ресурси для створення додатків iOS та аксесуарів для цих пристроїв, а саме нативне IDE Xcode.

Xcode - це IDE - інтегроване середовище розробки, створене Apple для розробки програмного забезпечення, для macOS, iOS, watchOS і tvOS. Це єдиний офіційно підтримуваний інструмент для створення та публікації додатків у магазині додатків Apple, призначений для використання початківцями та досвідченими розробниками [44]. Xcode включає в себе всі інструменти, необхідні для створення додатка в одному програмному пакеті; а саме:

- Текстовий редактор.
- Компілятор.
- Система складання.

За допомогою Xcode можна писати, компілювати і налагоджувати застосунок, а після закінчення надіслати його в магазин додатків Apple.

Як редактор коду Xcode підтримує величезну кількість мов програмування - C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit і Swift. Він використовує моделі програмування Cocoa, Carbon і Java.

Враховуючи вищенаведені можливості для розробки програмного забезпечення та вирішення поставлених задач, його тестування, аналізу, моніторингу, було обрано Xcode [23], [37].

2.4 Обґрунтування вибору архітектури системи

Шаблони проєктування справили величезний вплив на розробку програмного забезпечення. Подібно до веб-додатків, упровадження мобільних додатків також установило деякі перевірені шаблони та стандарти для подолання проблем і обмежень у розробленні мобільних додатків. Більшість мобільних застосунків було розроблено з використанням низькоякісного коду і не ґрунтуються на шаблонах архітектурного проектування. Розробка мобільного застосунку з правильним шаблоном проєктування може ефективно пов'язати користувацький інтерфейс із моделями даних і бізнес-логікою. Це вплине на те, який вигляд матиме ваш вихідний код [25] , [28].

Основні шаблони проєктування:

1) Apple MVC. MVC - це перший підхід до опису, а також реалізації розробки програмного забезпечення на основі їхніх обов'язків. Спочатку розроблений для настільних комп'ютерів, він широко використовувався як архітектура для веб-додатків основними мовами програмування. MVC включає в себе три компоненти для кожного об'єкта, а саме:
    - Модель - відповідає за дані або рівень доступу до даних.
    - Вид - відповідає за графічне відображення даних.
    - Контролер - слугує сполучною ланкою між представленням і моделлю. Контролер не відіграє роль посередника між поданням і моделлю. Він змінює модель на основі активності користувача в поданні, а також оновлює зміни за допомогою подання. Приклад роботи MVC зображено на рисунку 2.1[9].

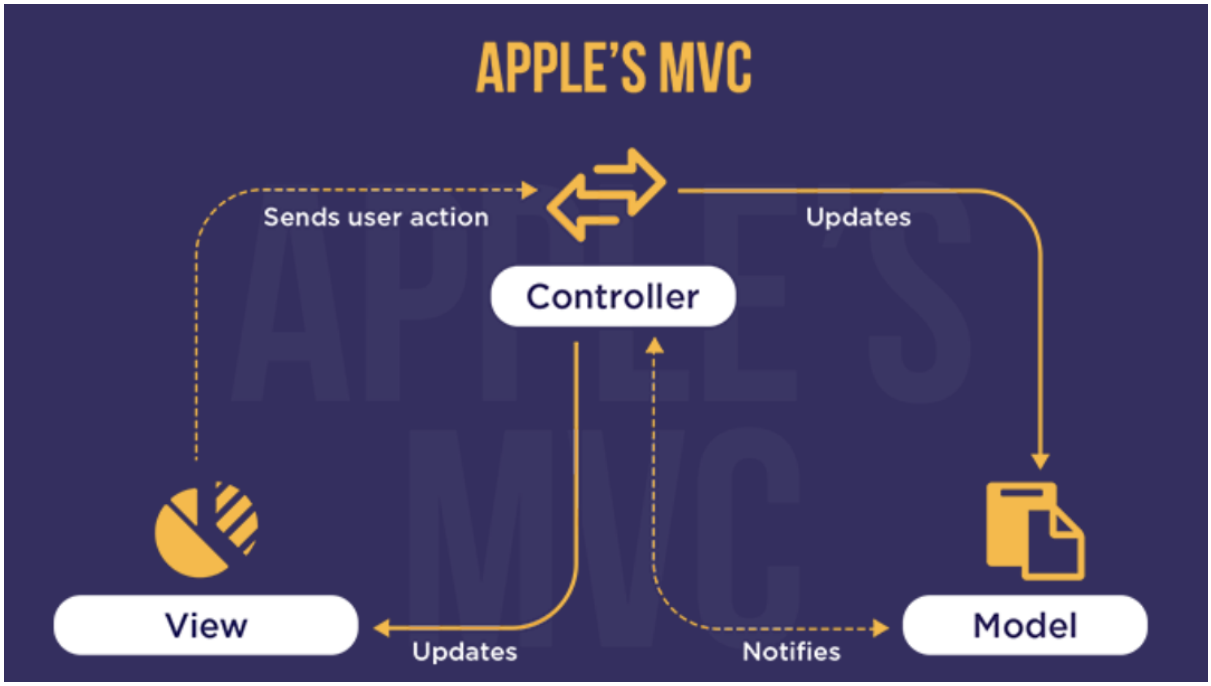


Рисунок 2.1 - Шаблон MVC

Переваги перед класичним шаблоном проєктування MVC. У цьому шаблоні проектування немає прямого зв'язку між моделлю та поданням. Крім того, контролер містить у собі логіку процесу подання. Такий поділ обов'язків більше підходить для розроблення додатків.

Недолік шаблону проектування Apple MVC.

Оскільки контролер так залучений у життєвий цикл подання, важко визначити, що вони відокремлені один від одного.

2) MVVM. Шаблон MVVM містить у собі три компоненти: Model, View і ViewModel. Тут ViewModel виступає в ролі посередника. ViewModel ініціює зміни в моделі, а також оновлює себе на основі оновленої моделі. Крім того, він включає прив'язку даних і дій користувача, як у шаблоні MVP Supervising Controller, але між поданням і моделлю подання, а не між поданням і моделлю [29]. Це пов'язує View для оновлення відповідно на основі посилання на ViewModel, як показано нижче на рисунку 2.2:

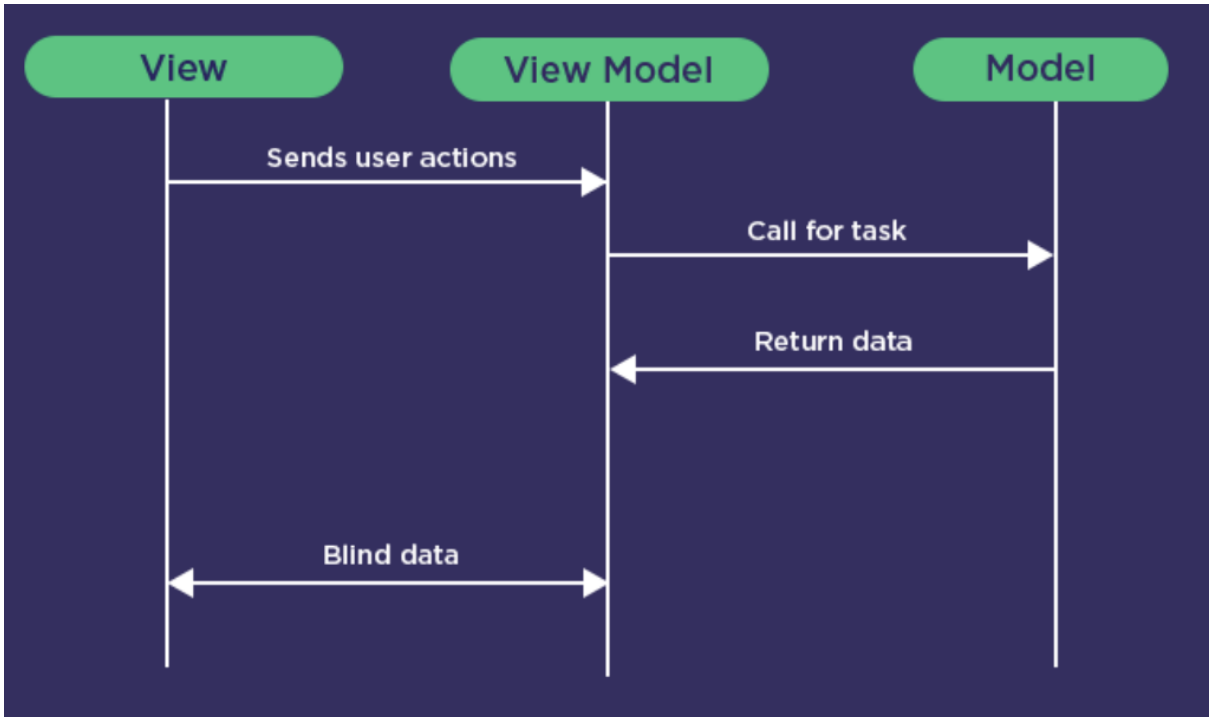


Рисунок 2.2 - Шаблон MVVM.

Переваги перед шаблоном проєктування MVC.
Односторонній зв'язок між View і ViewModel скорочує кількість рядків коду, необхідних для синхронізації View і ViewModel. Висока здатність до тестування та розширення.

3) VIPER. Цей шаблон має ідею поділу обов'язків за допомогою п'яти рівнів, перелічених нижче:

1. View — клас, який показує інтерфейс програми користувачеві, а також отримує відповідь.
2. Interactor — містить бізнес-логіку програми.
3. Presenter — включає бізнес-логіку, пов'язану з користувацьким інтерфейсом.
4. Entity — містить прості об'єкти даних.
5. Router – Обробляє обмін між модулями Viper. Приклад роботи та зв’язків між модулями у VIPER зображено на рисунку 2.3 [9].

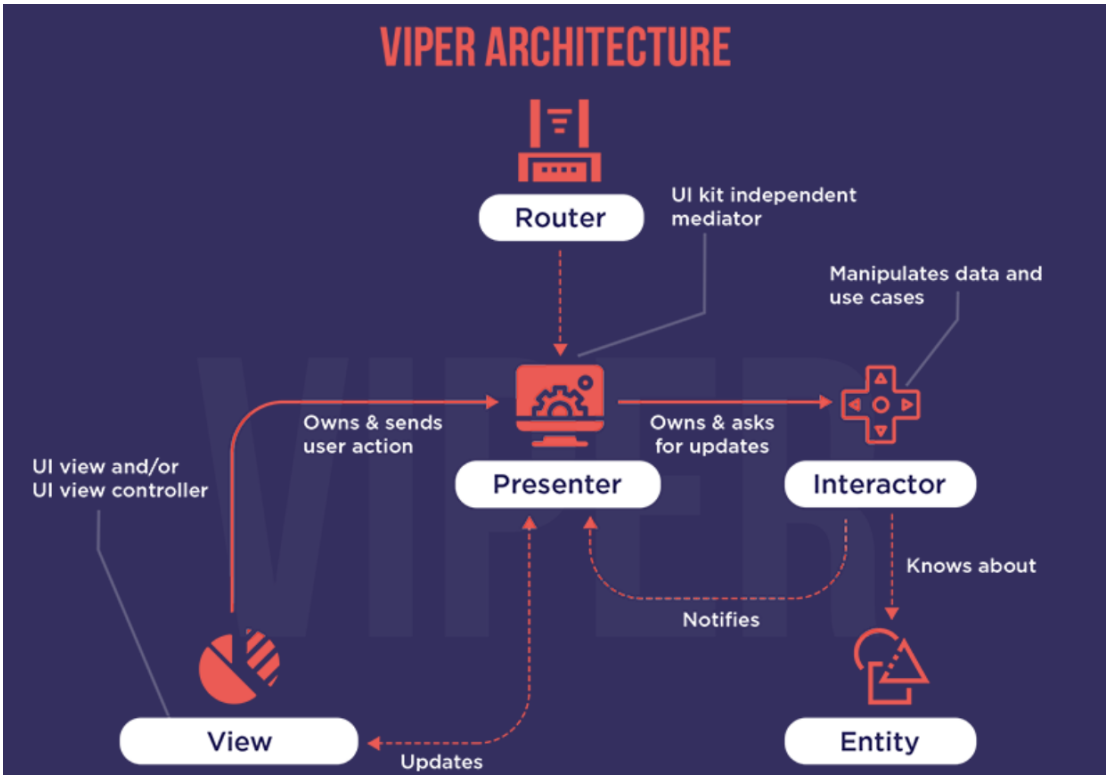
6. 

Рисунок 2.3 - Шаблон VIPER.

Переваги перед патернами MV(x).

Розв'язує проблеми збірки та навігації з попередніми архітектурами завдяки ідеї поділу. Завдяки чіткій архітектурі та слабкому зв'язку кожного модуля один з одним це знижує накладні витрати на зміну та виправлення помилок. Його модульний підхід забезпечує правильне середовище для модульного тестування.

2.4.1 Обґрунтування вибору архітектури

Враховуючи вищенаведені властивості та задачі які постають перед архітектурними шаблонами, було обрано шаблон проєктування MVVM, так як він в повній мірі вирішує поставлені в роботі задачі без надлишкових залежностей між модулями, зайвого коду чи абстракцій.

2.5 Обґрунтування вибору платформи для поширення додатку

Для поширення додатку та можливості завантажити на мобільний пристрій було вирішено обрати Apple App Store. Для цього було налаштовано наступні обов'язкові поля в системі для відображення додатку на платформі:
- Категорія "Business".

- Основна мова англійська.

- Додаткові мови корейська, іврит, японська.

- Вік від 12 років.

- Платформа iOS.

- Додано правила користування та політика конфіденційності.

- Додано інформацію, яка попереджає користувача про дані, які збирає додаток. Наприклад ідентифікатор пристрою, діагностику, аналітику, дані про роботу додатку та телефону, контент користувача, локацію [41], [45]. Приклад представлення користувачу, які дані збираються під час користування додатком, зображено на рисунку 2.4.

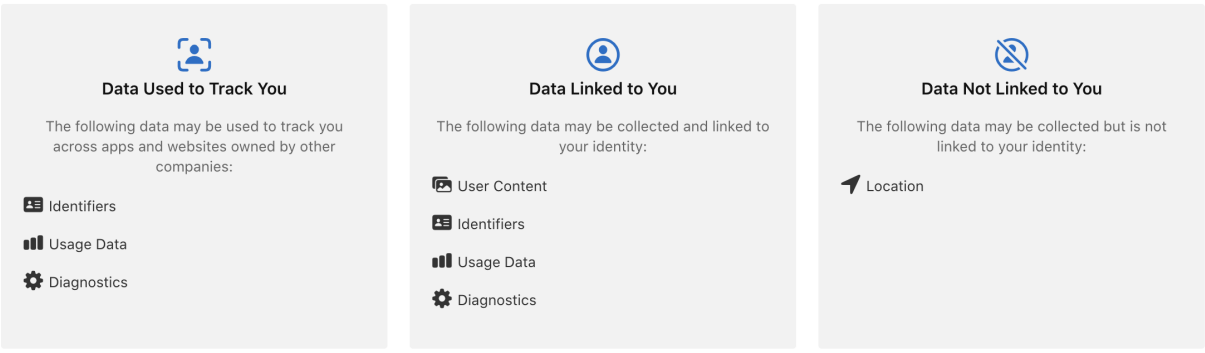


Рисунок 2.4 - Представлення користувачу типів даних, які збирає додаток.

2.6 Обґрунтування вибору мови програмування

Для реалізацій функцій додатку було запропоновано дві мови програмування, такі як Swift та Objective C. В таблиці 2.1 наведено порівняльну характеристику та головні відмінності Swift та Objective C.

Таблиця 2.1 - Порівняння характеристик Swift та Obective-C

Властивості	Swift	Objective-C
Дизайн	Розроблений для розробки операційних систем Apple.	Розроблено як об'єктно-орієнтований разом із функцією обміну повідомленнями Smalltalk.
Спадкування	Не допускає множинного успадкування	Не допускає множинного успадкування
Ліцензія	Це проект із відкритим вихідним кодом під ліцензією Apache	Перебуває під ліцензією GPL (General Public License)
Тип	Статичний і типізований	Динамічна типізація
Логічні оператори	Використовує істинні та хибні значення	C++ використовує YES, NO і BOOL
Шаблони та бібліотеки	Підтримує кілька бібліотек разом з Objective C	В Objective C відсутні бібліотеки шаблонів

Також більш детальна інформація про пункти з таблиці 2.1, наведено нижче.

- Спадкування.

Swift переглядає загальноприйняті умови успадкування. Таким чином, більше не потрібно ставити кому в кінці рядка або писати круглі дужки, щоб оточити умовні вирази всередині операторів if/else. Ще велика зміна в тому, що виклики методу не розташовуються всередині один одного. Метод і функції викликаються в Swift, використовуючи стандартний розділений комами список параметрів у круглих дужках. Результатом є чиста, виразна мова, зі спрощеним синтаксисом і граматикою.

- Swift легше підтримувати ніж Objective-C.

Спадщина утримує Objective-C позаду: мова не може розвиватися без розвитку C. C вимагає підтримки 2 кодових файлів для поліпшення часу інсталяції та ефективності створення додатка - вимога, яка переноситься на Objective-C. Swift скасовує вимогу двох-файлів. Xcode і компілятор LLVM може з'ясувати залежність і виконати покрокові зміни автоматично. У результаті, повторюване завдання розділення змісту (файл заголовка) від тіла (файл реалізації) стає справою минулого. Swift поєднує в собі заголовок Objective-C (.h) і файли реалізації (.m) в одному файлі коду (.swift).

- Безпечність.

Одним із моментів в Objective-C можна вважати спосіб обробки покажчиків, особливо nil (NULL). В Objective-C нічого не трапиться, якщо викликати метод зі змінною покажчика nil (неініціалізованим). Вираз або рядок коду стає нездійсненними і може здатися, що вираз не впаде, але насправді буде повно помилок. Опціональні типи дають можливість існування в Swift коді nil опціонального значення, що свідчить про можливість створення помилки компілятора при написанні поганого коду. Це створює короткий цикл зворотного зв'язку і дає змогу кодувати більш впевнено. Проблеми можуть бути усунені вже при написаному коді, що значно зменшує кількість часу і грошей, які витрачаються на виправлення помилок.

- Управління пам'яттю.

Swift уніфікований так, як ніколи не був Objective-C . Підтримка автоматичного підрахунку посилань (ARC) є повною по процедурним та об'єктно-орієнтованим шляхам коду. В Objective-C, ARC підтримується всередині Cocoa API та об'єктно-орієнтованого коду, але він не доступний для C коду та API, наприклад Core Graphics. Це означає, що розробник бере на себе управління пам'яттю під час роботи з Core Graphics APIs, та інших старіших API, доступних на iOS. Витоки пам'яті, які може мати Objective-C, неможливі в Swift. Тому що ARC обробляє все управління пам'яттю під час компіляції, і витрати, які пішли б на управління пам'яттю, тепер можна сфокусувати на core app logic і нових можливостях. Це відбувається тому, що ARC у Swift працює і на процесуальному, і на об'єктно-орієнтованому коді [21], [26].

Враховуючи вище перелічені порівняльні характеристики мов програмування, було обрано мову Swift.

2.7 Обґрунтування вибору технологій для алгоритмів штучного інтелекту.

Для реалізації логіки опрацювання фото з використанням нейронної мережі, було розроблено тестове завдання класифікації зображень і випробувано з використанням стандартних інструментів [47], [48].

Core ML Tools за замовчуванням створює модель Core ML з багатовимірним масивом, як типом для входу і виходу. Приклад використання моделі з масивом для вхідних даних, зображено на рисунку 2.5.

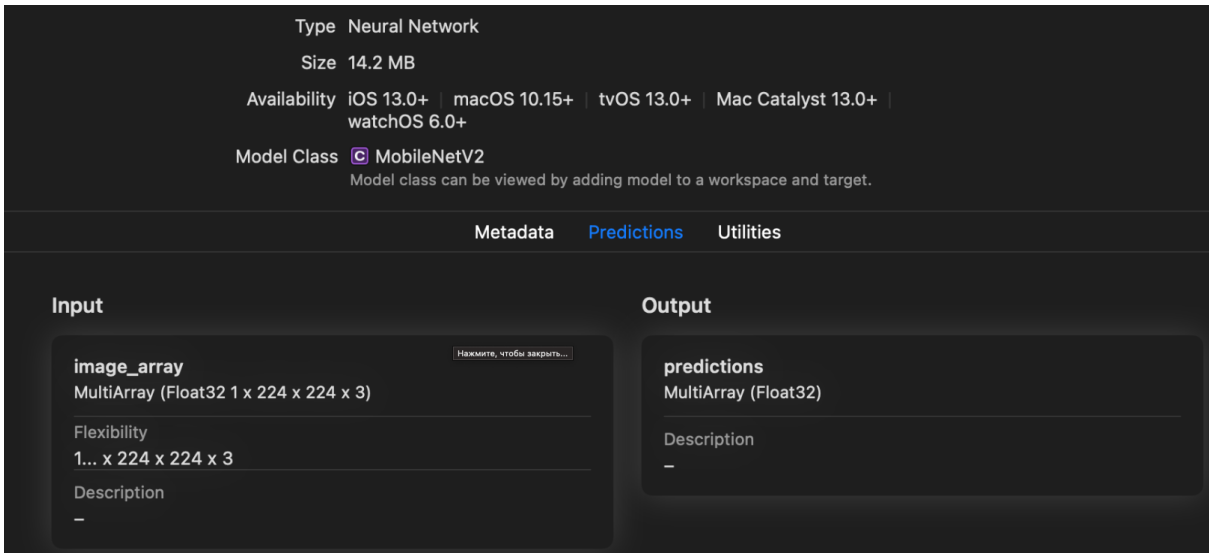


Рисунок 2.5 - Модель із вхідними даними у вигляді масиву.

Проте для зображень можливо вказати тип входу і виходу як ImageType. Такий тип вхідних даних, є ефективніший за рахунок відсутності операції копіювання елементів, якщо на вході очікується тільки зображення. Приклад моделі вхідних даних типу зображення представлено на рисунку 2.6.

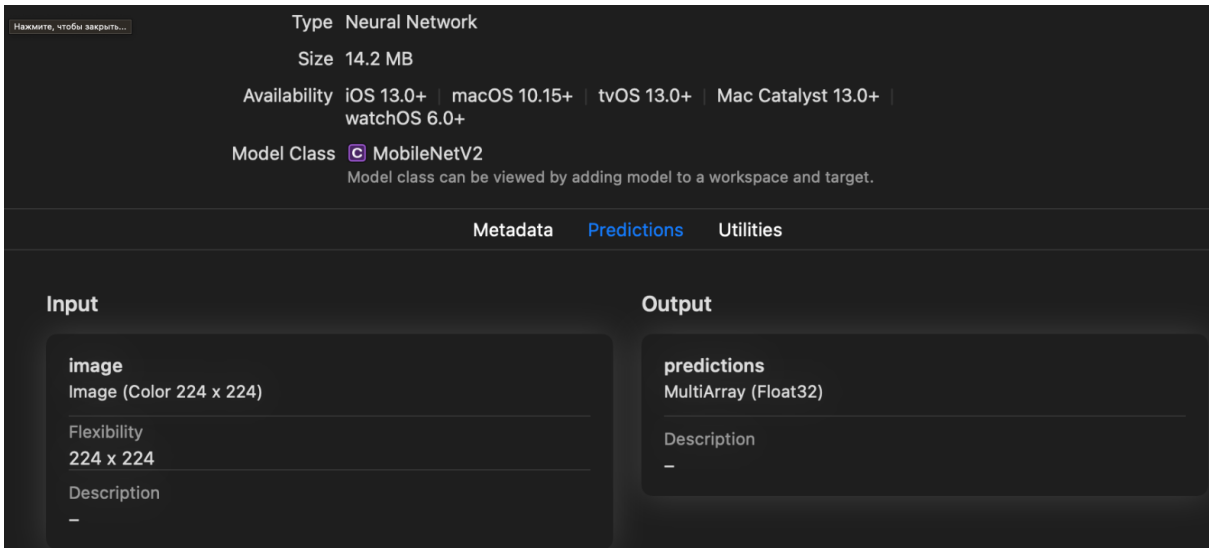


Рисунок 2.6 Модель із вхідними даними у вигляді зображення.

В CoreML моделі підтримують 8-бітові зображення в градаціях сірого і 32-бітові кольорові зображення з 8 бітами на компонент. Також підтримують int 32, double і float 32 як скалярні типи. Починаючи з iOS 16 і macOS 13, можливо

використовувати зображення OneComponent16Half Grayscale і плаваючі 16 мультимасивів для вхідних і вихідних даних.

Моделі TF відрізняються тим, що керують введенням зображень. Необхідно вивчити модель, щоб визначити, чи потрібна попередня обробка для перетвореної моделі.

Для тестування моделей було задано вхідним типом зображення в форматі RGB. Приклад налаштування RGB зображень представлено на рисунку 2.7.

```
Y[0, :, :] = channelScale * X[0, :, :] + blueBias
Y[1, :, :] = channelScale * X[1, :, :] + greenBias
Y[2, :, :] = channelScale * X[2, :, :] + redBias
```

Рисунок 2.7 - Налаштування RGB зображення для моделі.

Налаштування моделі, яка приймає вхідне зображення і виводить масив об'єктів відповідно до зазначених типів об'єктів представлено на рисунку 2.8.

```
message VisionFeaturePrint {

    // Specific vision feature print types

    // Scene extracts features useful for identifying contents of natural images
    // in both indoor and outdoor environments
    message Scene {
        enum SceneVersion {
            SCENE_VERSION_INVALID = 0;
            // VERSION_1 is available on iOS,tvOS 12.0+, macOS 10.14+
            // It uses a 299x299 input image and yields a 2048 float feature vector
            SCENE_VERSION_1 = 1;
        }

        SceneVersion version = 1;
    }

    // Object extracts features useful for identifying and localizing
    // objects in natural images
    message Object {
        enum ObjectVersion {
            OBJECT_VERSION_INVALID = 0;
            // VERSION_1 is available on iOS,tvOS 14.0+, macOS 11.0+
            // It uses a 299x299 input image and yields two multiarray
            // features: one at high resolution of shape (288, 35, 35)
            // the other at low resolution of shape (768, 17, 17)
            OBJECT_VERSION_1 = 1;
        }

        ObjectVersion version = 1;

        repeated string output = 100;
    }

    // Vision feature print type
    oneof VisionFeaturePrintType {
        Scene scene = 20;
        Object object = 21;
    }

}
```

Рисунок 2.8 - Налаштування моделі, яка приймає вхідне зображення і виводить масив об'єктів.

Нижче наведено опис ключових характеристик моделей та результати тестування.

- CoreML. Є основою для фреймворків та функціональних можливостей для конкретних доменів. Core ML підтримує Vision для аналізу зображень, Natural Language для обробки тексту, Speech для перетворення аудіо в текст і Sound Analysis для ідентифікації звуків в аудіо. При тестуванні фреймворку, використовувались стандартні інструменти XCode для збору ефективності роботи. Приклад ефективності роботи CoreML та навантаження на девайс зображено на рисунку 2.9. Фреймворк побудований на трьох основних технологіях:
- ЦП (центральний процесор): працює з процедурами, що інтенсивно використовують пам'ять, як обробка природної мови (NLP).
- GPU (графічний процесор): підходить для ресурсномістких процесів, як робота із зображеннями та виявлення об'єктів.
- ANE (Apple Neural Engine): платформа була розроблена для прискорення роботи нейронних мереж.

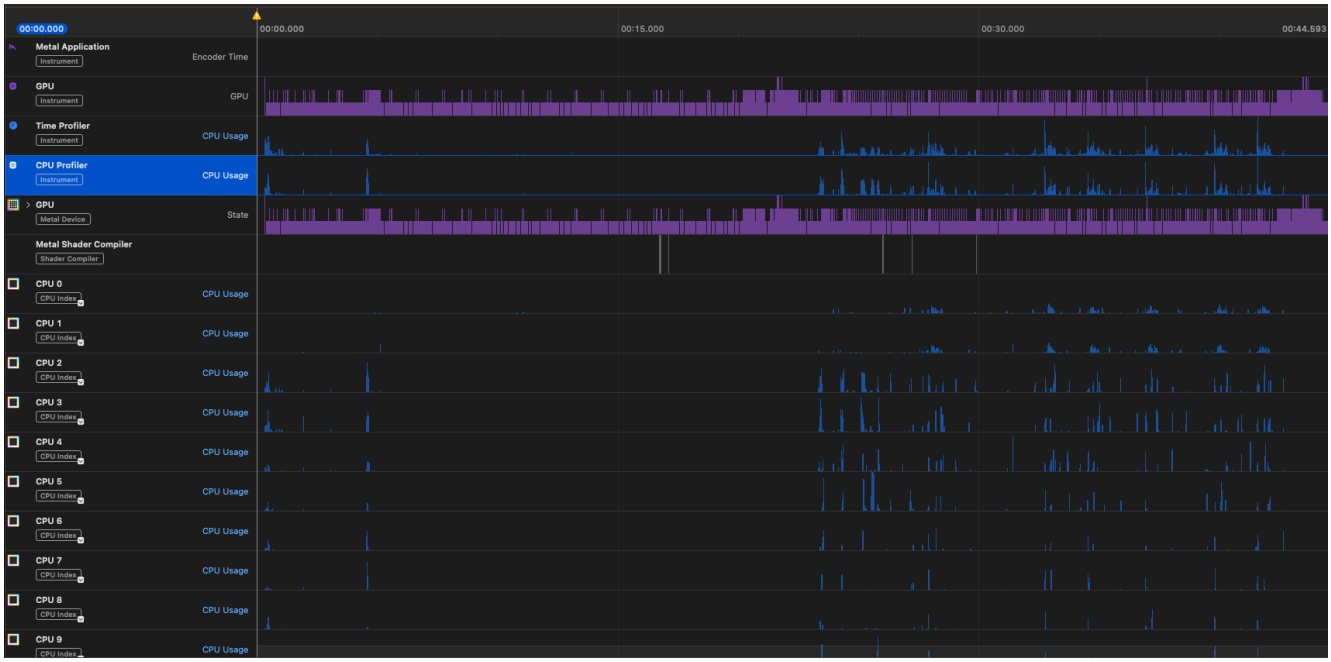


Рисунок 2.9 - Навантаження CoreML на пристрій в процесі класифікації зображень.

Одна з головних переваг Core ML полягає в тому, що він дає змогу перемикатися між CPU, GPU і ANE під час його роботи. Це оптимізує

продуктивність на пристрої і означає, що вам не потрібно вирішувати, який із них запускати для кожної моделі.

- TFLearn. Це модульна і прозора бібліотека глибокого навчання, побудована на основі Tensorflow, яка надає API більш високого рівня до TensorFlow, щоб полегшити і прискорити експерименти, залишаючись при цьому повністю прозорою і сумісною з ним. Переваги: продуктивно-швидка система, великий перелік можливостей. Навантаження TSLearn на пристрій в процесі класифікації зображень зображено на рисунку 2.10.

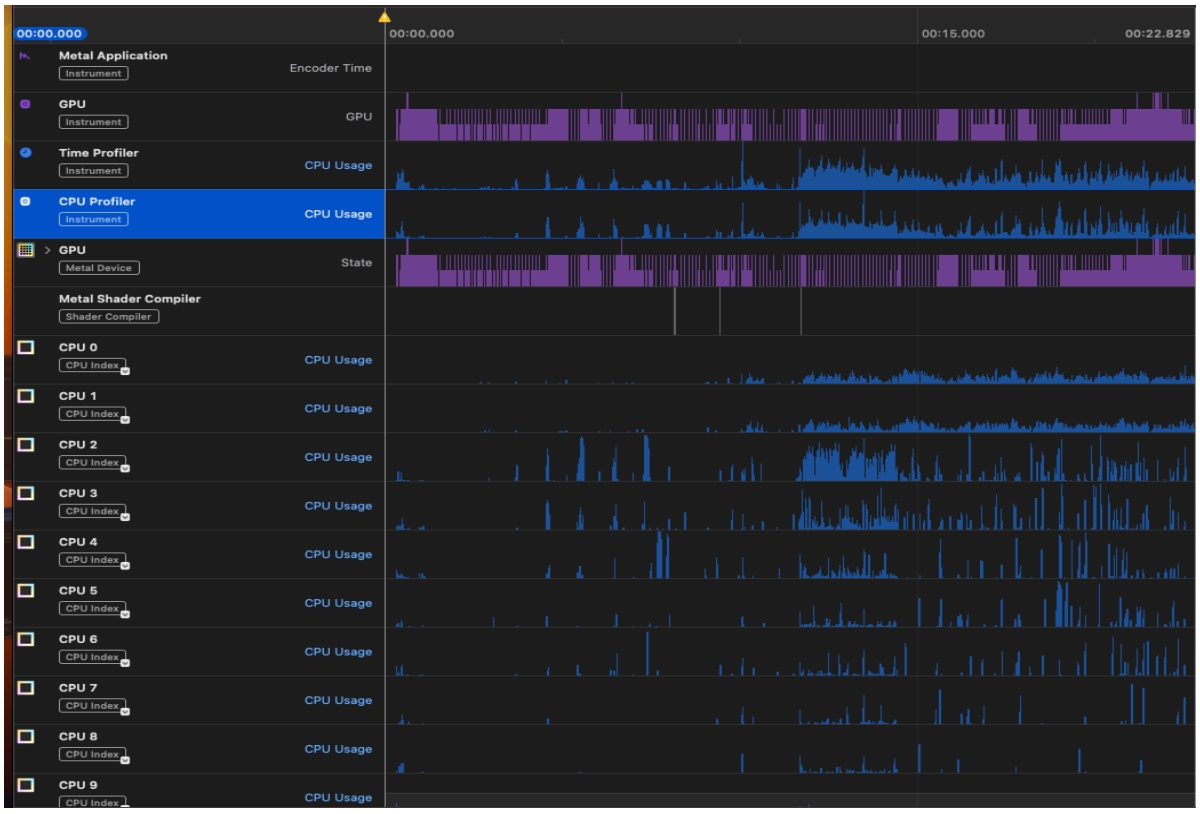


Рисунок 2.10 - Навантаження TSLearn на пристрій в процесі класифікації зображень.

З рисунків 2.9 та 2.10 можна спостерігати, що CoreML з 0.0 до 15 секунд роботи системи, навантажує GPU на 8-15%, а використані процеси CPU на 0-20% , в той час як я TSLEarn навантажує GPU на 40-60% та процеси CPU 5-43%.

Після порівняння результатів тестування CoreML з TFLearn для класифікації зображень і роботи з нейронною мережею, було обрано CoreML. Це нативний фреймворк від Apple, який постійно покращується, оновлюється, має чітку та

прозору документацію, простий у використанні, широкий спектр можливостей та має велику спільноту розробників для обговорення питань чи проблем.

2.7 Обґрунтування вибору локального сховища

Існує два основні способи зберігання даних у застосунку iOS: CoreData і Realm.

Порівняльні характеристики CoreData та Realm представлено в таблиці 2.1.

Таблиця 2.1 - Характеристики CoreData та Realm.

CoreData	SQLite	Realm
Core Data - це основа для управління об'єктним графом. Об'єктний граф - це сукупність взаємопов'язаних об'єктів.	SQLite - легкий движок баз даних, який є неймовірно продуктивним, тому підходить для вбудованих систем, таких як мобільні пристрої	Realm - це кросплатформенна мобільна база даних, яка працює швидше і ефективніше, ніж SQLite і Core Data
Основні дані зберігають вміст об'єкта в детальному вигляді	SQLite орієнтований в основному на традиційне зберігання вмісту таблиць	Realm зберігає об'єкти в оптимальному форматі, що відображається в пам'яті, використовуючи технологію зберігання в стовпчиках для швидкого пошуку.
CoreData не є базою даних	SQLite - це база даних відносин	Realm не є базою даних, але Вона дозволяє оголошувати зв'язки між об'єктами за допомогою графа об'єктів

- Переваги використання Realm над CoreData. Немає складної схеми. Не потрібно визначати схему для даних перед їх збереженням. Дані автоматично зберігаються у вигляді об'єктів, що означає, що не потрібно керувати окремими рядками і стовпчиками структурованих даних у додатку. Це спрощує управління даними порівняно з CoreData. У Realm можна зберігати об'єкти практично будь-якого типу: від простих типів, як-от рядки, числа і дати, до складніших типів, як-от масиви, списки і навіть вкладені об'єкти. Це дає змогу легко моделювати відносини між різними сутностями, не турбуючись про попереднє створення явної схеми.
- Переваги використання CoreData над Realm. Apple Foundation Data Framework за замовчуванням дає змогу розробникам просто налаштувати базу даних і відразу зануритися в програмування застосунку. Це чудовий варіант, якщо застосунок не має бути дуже складним, а бази даних не надто великими. Однак, якщо вийти за його межі (скажімо, за допомогою SQLite), це ускладниться, оскільки SQLite зберігає свої дані не так, як передбачала Apple. Крім того, якщо виконувати будь-які складні запити або керувати синхронізацією бази даних на декількох пристроях (наприклад, iCloud), то це стає ще більш громіздким через обмеження, пов'язані з CoreData.

На основі дослідження представлених варіантів для локального сховища було обрано CoreData для великих об’ємів даних та для незначних об’ємів фреймворк UserDefaults. Обидва фреймворки є нативні для платформи iOS розроблені компанією Apple, мають широкий спектр можливостей, за рахунок нативності, мають високий коефіцієнт швидкодії виконання коду та постійно оновлюються і підтримуються.

2.8 Обґрунтування вибору відображення архітектурних планів

Щоб фотограф мав змогу орієнтуватись на будівельному майданчику, мав представлення про обсяги своєї роботи, не переходив у зону відповідальності іншого фотографа та бачив місця в яких необхідно зробити фото, було вирішено використовувати архітектурні плани будівель. Для вирішення цієї задачі відображення об’єктів у додатку, без втрати швидкості у відклику додатку і сильного навантаження на CPU та GPU, було запропоновано використати плани у вигляді графічних креслень. План являє собою графічні креслення поверхів з апартаментами в зменшених масштабах. Такий підхід дозволяє створювати плани під будь-яку архітектуру будівель і не обмежуватись у розмірах чи кількості даних. Для взаємодії та відображення планів, креслення форматуються у формат svg. Перевагою такого підходу є:

- Можливість компресії даних.
- Можливість змінювати контент планів в додатку з використанням їх мета дати.
- Можливість взаємодії користувачів із планом. Збільшити, зменшити, змістити, повернути.

Також такий підхід із графічними планами допомагає вирішити задачу відображення точок збирання даних на будівельному майданчику. Приклад відображення плану в додатку, зображено на рисунку 2.11.

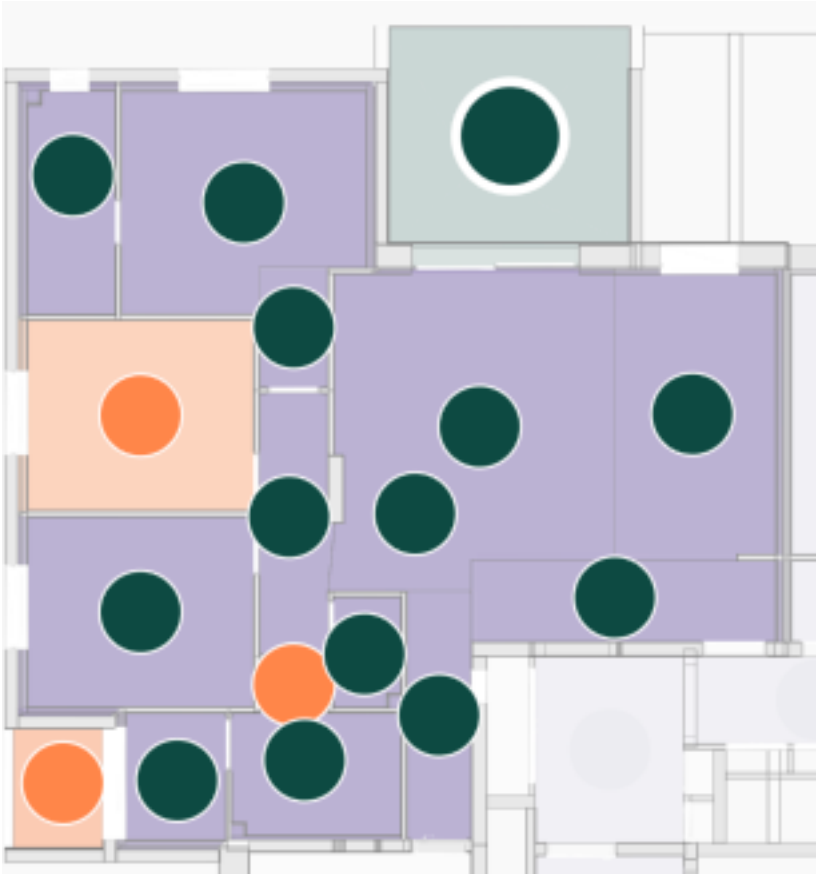


Рисунок 2.11- Відображення архітектурного плану для одного поверху.

2.9 Висновок

В даному розділі було досліджено, проаналізовано та обґрунтовано вибір платформи для реалізації програмного забезпечення, операційної системи, середовища розробки, мови програмування, архітектурних підходів, методів поширення додатку, технологій для алгоритмів штучного інтелекту, тип та фреймворк для зберігання даних в локальному сховищі.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДОДАТКУ

Виходячи із поставленої мети роботи, система, що розробляється повинна забезпечити:

- Збір інформації у вигляді фото.
- Робота з камерою 360 градусів.
- Локальне збереження інформації.
- Робота із сервером.
- Можливість обробки даних.
- Зручний інтерфейс.

3.1 Проектування та архітектура

Як було описано в другому розділі, додаток має бути спланований відповідно до архітектурного шаблону MVVM + Clean Architecture. Для реалізації цієї задачі, всі файли проекту структуровано, згруповано відповідно до їхнього призначення. Основа структури проекту зображена на рисунку 3.1

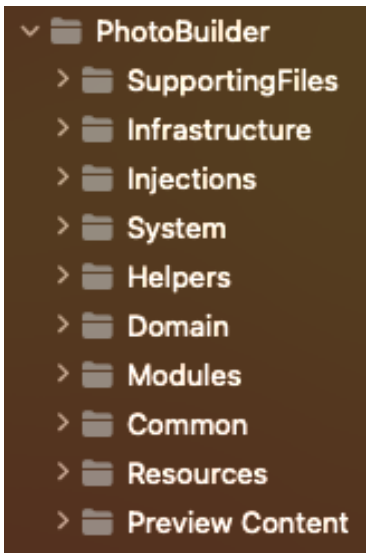


Рисунок 3.1 - Основна структури проекту.

3.1.1 Допоміжні налаштування

В SupportingFiles розміщені файли, які відповідають за налаштування проекту при запуску. Налаштовуються файли для різних середовищ проекту, роботи з хмарним сервісом AWS, зовнішнім сервісом Firebase та внутрішньою системою Appe Store Connect. Структура архітектурної одиниці SupportingFiles зображено на рисунку 3.2.

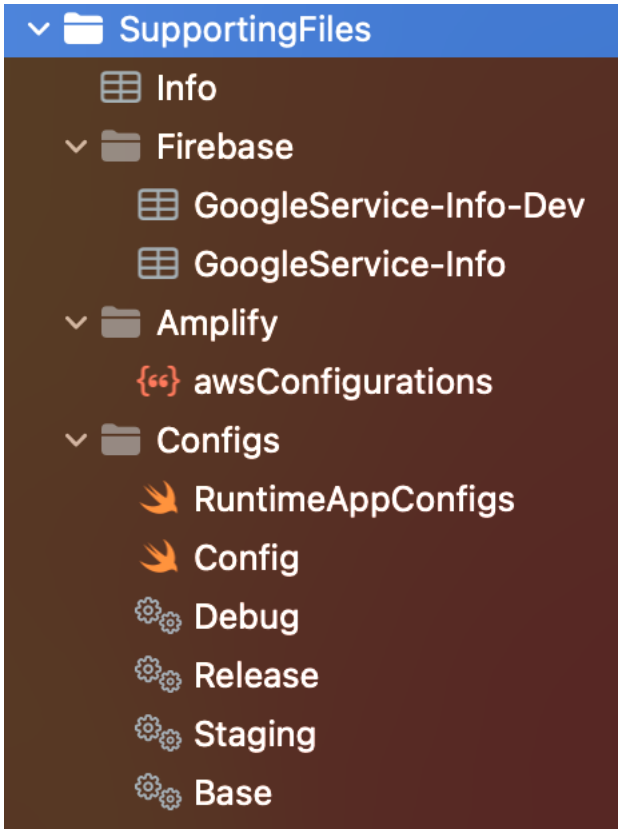


Рисунок 3.2 - Структура SupportingFiles.

Проект має декілька середовищ роботи, від яких залежать налаштування самого додатку.

Середовища робота:

- Debug.
- Release.
- Stage.

Від середовища можуть залежати найрізноманітніші конфігурації додатку. Наприклад кінцева точка для серверу, посилання на якісь ресурси, хмарні системи, сервіси та інше.

Середовище Debug використовується саме для розробки. На цьому середовищі вказані конфігурації на ресурси, системи, технології з якими можна експерементувати, налаштовувати роботу зовнішніх сервісів, бібліотек, серверу без впливу на робочу версію додатку, яка знаходиться на платформі для поширення. Ці налаштування будуть діяти тільки для цього середовища. Stage використовується для тестування. Це середовище має наближену конфігурацію до Release аби тестувальники могли змогу протестувати та імітувати реальну роботу додатку, яку очікує користувач. Release це налаштування з якими додаток завантажується на платформу для поширення. В цьому середовищі не рекомендовано робити якісь зміни на пряму. Новий функціонал, виправлення помилок в коді, дизайні чи інше спочатку має відбуватись на Debug, потім тестуватись на Stage і тільки тоді інтегрується в середовище Release. Помилки на Release зазвичай критичні, тому що це заважає користування додатком. Для роботи середовищ створені відповідні файли Debug, Stage, Release з конфігураціями. Приклад параметрів конфігурацій Debug середовища зображено на рисунку 3.3

```
// Configuration settings file format documentation can be found at:
// https://help.apple.com/xcode/#/dev745c5c974

BASE_BUNDLE_IDENTIFIER
PRODUCT_BUNDLE_IDENTIFIER
APP_NAME
APP_DISPLAY_NAME
BUNDLE_DISPLAY_NAME
ONLY_ACTIVE_ARCH[config
BASE_URL
FIREBASE_PLIST
```

Рисунок 3.3 - Параметри конфігурацій Debug.

В наслідок аналізу задачі розподілення роботи між фотографами на будівельному майданчику, був розроблений сервіс авторизації з використанням хмарного сервісу AWS, який був інтегрований з допомогою Pods. Для

налаштування роботи AWS створено файл awsConfigurations з відповідними вимогами сервісу. На рисунку 3.4 наведений конфігураційний файл у якому налаштовані параметри роботи сервісу AWS з додатком.

```
"auth": {
    "plugins": {
        "awsCognitoAuthPlugin": {
            "UserAgent": "aws-amplify-cli/0.1.0",
            "Version": "1.0",
            "IdentityManager": {
              "Default": {}
            },
            "CredentialsProvider": {
                "CognitoIdentity": {
                  "Default": {
                    "PoolId": "us-east-1",
                    "Region": "us-east-1"
                  }
                }
              },
            "CognitoUserPool": {
              "Default": {
                "PoolId": "us-east-1",
                "AppClientId": "104*******",
                "Region": "us-east-1"
              }
```


Рисунок 3.4 - Конфігурація awsConfigurations.

Внаслідок аналізу задачі стабільної роботи додатку, було використано сервіс Firebase. Для конфігурації сервісу Firebase був створений файл GoogleService-Info, в якому добавлені конфігурації для правильної роботи та зв’язку додатку з сервісом. Після чого налаштування файлу були розподілені на два допоміжних файли для Debug з використанням GoogleService-Info-Dev і для Relase GoogleService-Info. На рисунку 3.4 наведені параметри конфігурацій GoogleService-Info.

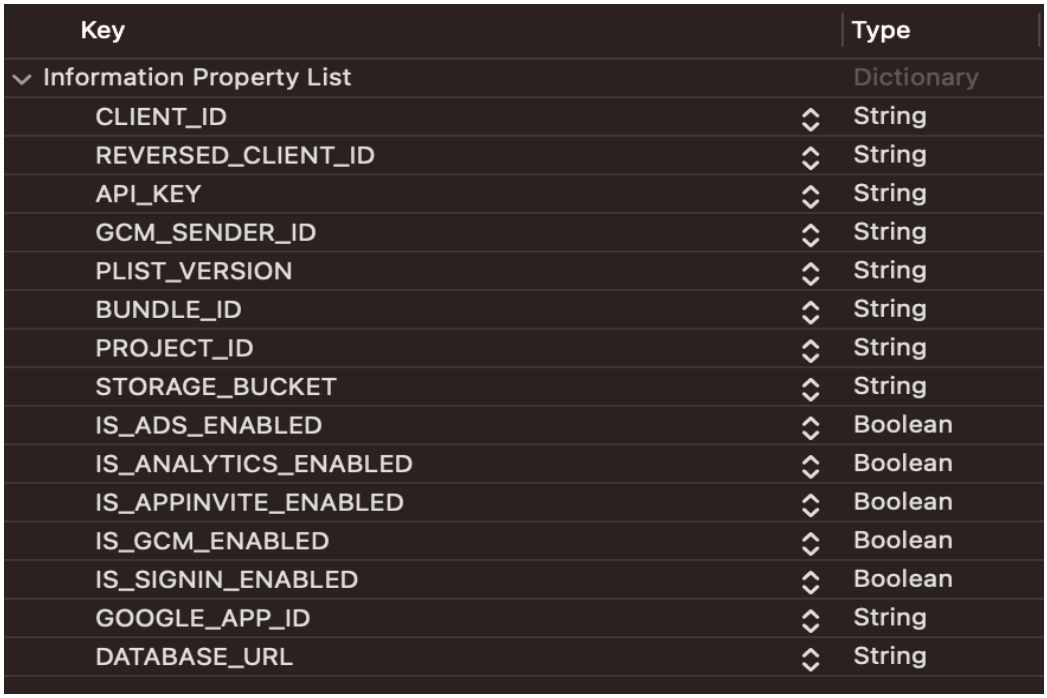


Рисунок 3.4 - Параметри конфігурацій GoogleService-Info.

3.1.2 Інфраструктура

В Infrastructure містяться файли, які відповідають за логіку конструювання процесу обробки даних, базових налаштувань для користувача та сповіщень. Приклад структури зображено на рисунку 3.5.

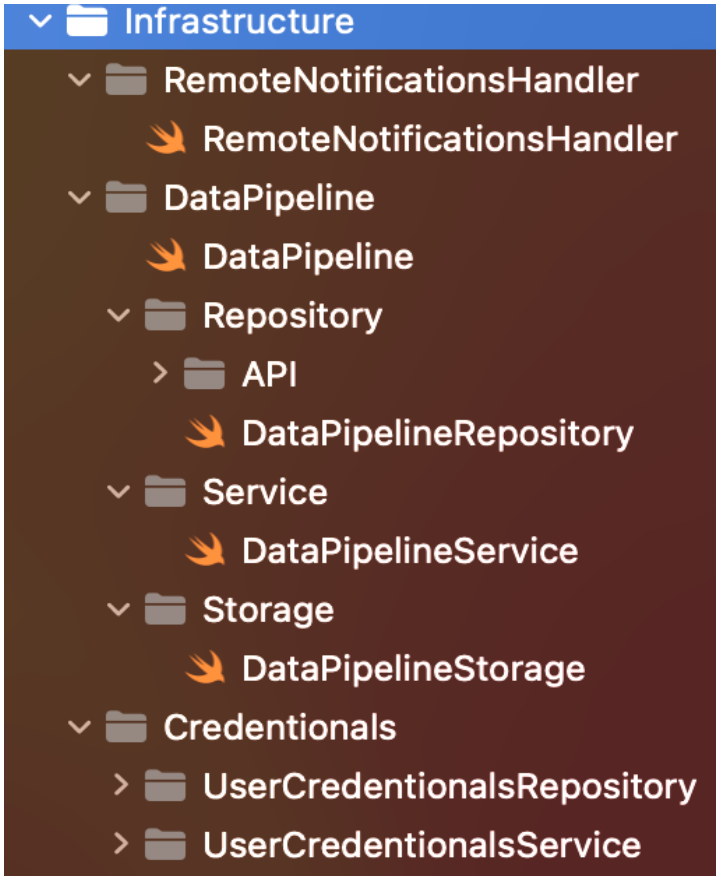


Рисунок 3.5 - Структура Infrastructure.

В папці RemoteNotificationsHandler знаходиться логіка, яка відповідає за оброблення повідомлень, які надходять до додатку із зовнішніх сервісів.

DataPipeline являє собою модуль, в якому знаходиться логіка викликів певних класів та їх методів, щоб опрацювати дані, які надходять з камери в додаток. Цей модуль поділяється на такі шари як Domain, Data. До Domain відноситься Service, а до Data, відноситься Repository та Storage.

Credentionals являє собою сервіс для простих маніпуляцій з даними про користувача. Основна задача це зберегти в зашифрованому вигляді ім’я, електронну пошту користувача з допомогою фреймворку Keychain та дістати або ж видалити у потрібний момент.

3.1.3 Ін'єкції

Inections являє собою набір файлів, які ініціалізують всі сутності в проекті, класи та їх залежності з використанням Container. Цей спосіб ініціалізації дозволяє вирішити проблему залежностей між сутностями. В додатку є три головні шари ініціалізації: Storages, Helpers, Services. Приклад структури Injections зображено на рисунку 3.6.

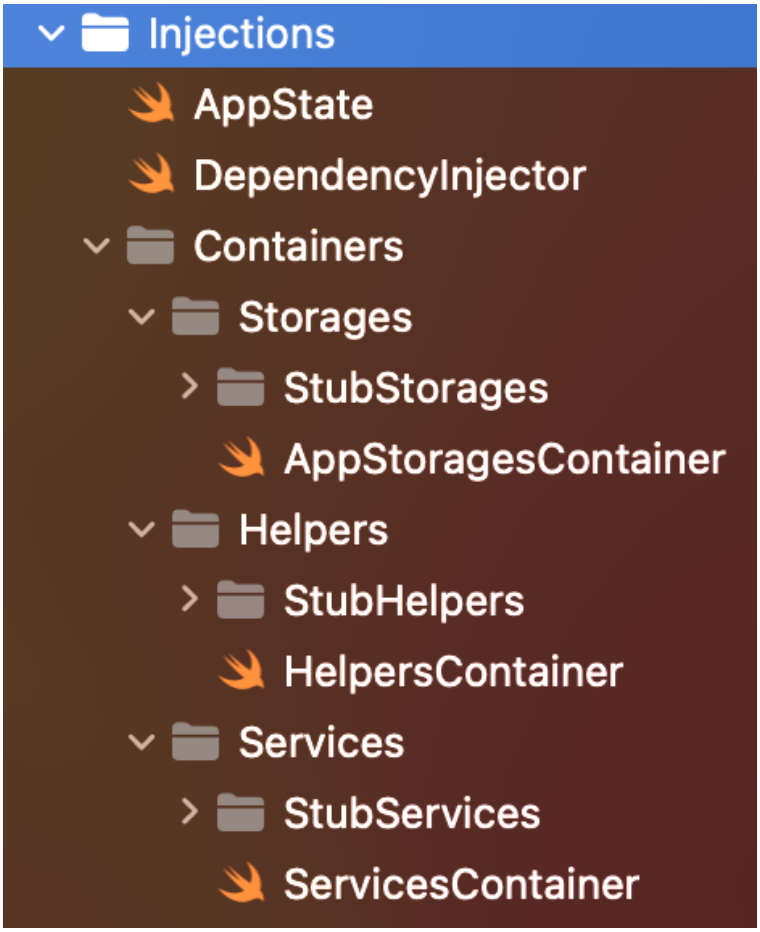


Рисунок 3.6 - Структура Injections.

AppState - це файл, який відповідає за стани додатку, реагує на їх зміну.

DependencyInjector - файл, який відповідає за залежності між сутностями та їх конструкцію.

Container містить три шари сутностей, які відповідають за різні аспекти додатку.

- Storages - це сутності, які дістають, зберігають, видаляють, оновлюють інформацію в локальні базі даних чи іншому локальному сховищі.

- Helpers - це сутності, які дають змогу налаштувати зв’язок якогось модулю з іншим модулем, сервісом чи додатковими інструментами.

- Services - це сутності, які відповідають за бізнес логіку в модулях та передачею даних між шарами Presentation та Data.

3.1.4 Система

System являє собою головний модуль додатку, який виконує запуск програми та ініціалізує всі сутності, сервіси, бібліотеки. Приклад структури System зображено на рисунку 3.7.

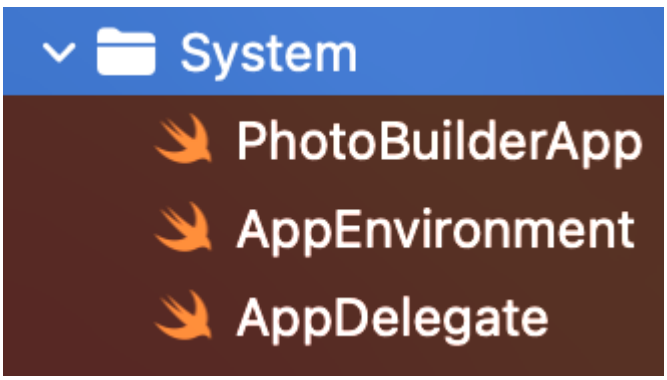


Рисунок 3.7 - Структура System.

PhotoBuilderApp - це головний файл проекту, який відповідає за запуск програми, іншими слова це точка входу та зв’язку програми із системою. В цьому

файлі налаштовано зовнішні сервіси, вказані API ключі до них та ініціалізується DIContainer.

AppEnvironment - це файл в якому конфігуруються зв’язки модулів, ініціалізуються всі шари та їх залежності.

AppDelegate - це допоміжний файл системи, який містить в собі методи для обробки стану додатку чи сповіщень. Створений щоб підтримувати старіші версії iOS.

3.1.5 Допоміжні сутності

Helpers являє собою модуль, який містить допоміжні сутності для головних модулів додатку. Приклад структури Helpers зображено на рисунку 3.8.

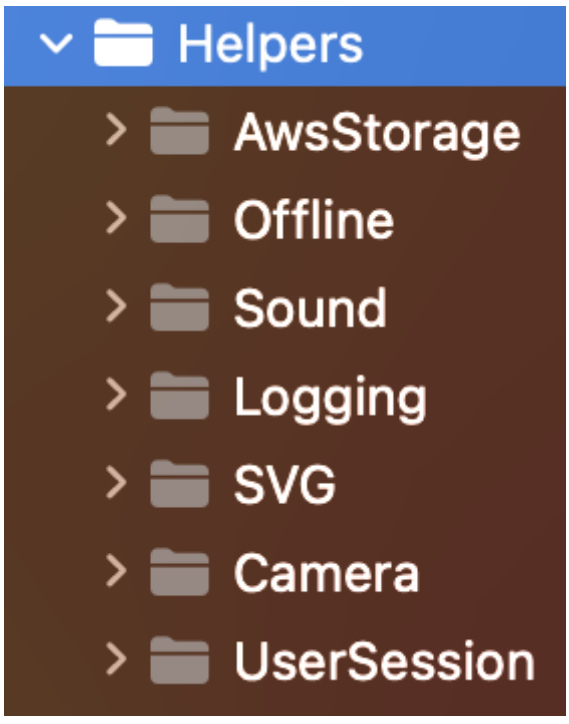


Рисунок 3.8 - Структура Helpers.

AwsStorage - являє собою сутність, яка дає змогу контактувати із сервісом AWS.

Offline - являє собою сутність, яка виконує логіку по збереженні деяких даних до локальної бази даних.

Sound - являє собою сутність, яка містить в собі логіку звуків у додатку.

Logging - сутність, яка містить логіку та функції для відображення прогресу чи стани процесів у консолі. Використовується в якості допоміжного інструменту при розробці.

SVG - сутність, яка відповідає за конфігурацію та налаштування даних у форматі svg для подальшої роботи.

Camera - сутність, яка відповідає за логіку роботи, зв’язку додатку і камери 360.

UserSession - сутність, яка містить допоміжну логіку для сесію користувача в додатку.

3.1.6 Сутності

Domain - це модуль, який відповідає за сутності та об’єкти баз даних в додатку. Приклад структури Domain зображено на рисунку 3.9.

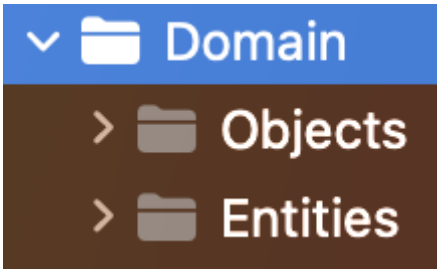


Рисунок 3.9 - Структура Domain.

3.1.7 Головні модулі

Modules - це збірка головних модулів додатку їх шарів та зв’язків. Приклад структури Modules зображено на рисунку 3.10

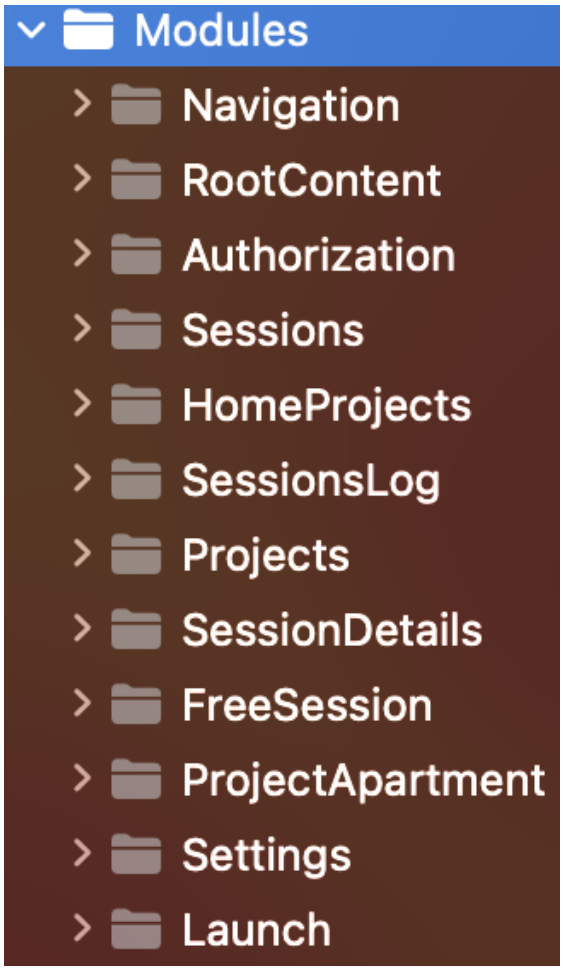


Рисунок 3.10 - Структура Modules.

Navigation містить файли, які відповідають за навігацію між екранами. Приклад структури Navigation зображено на рисунку 3.11.

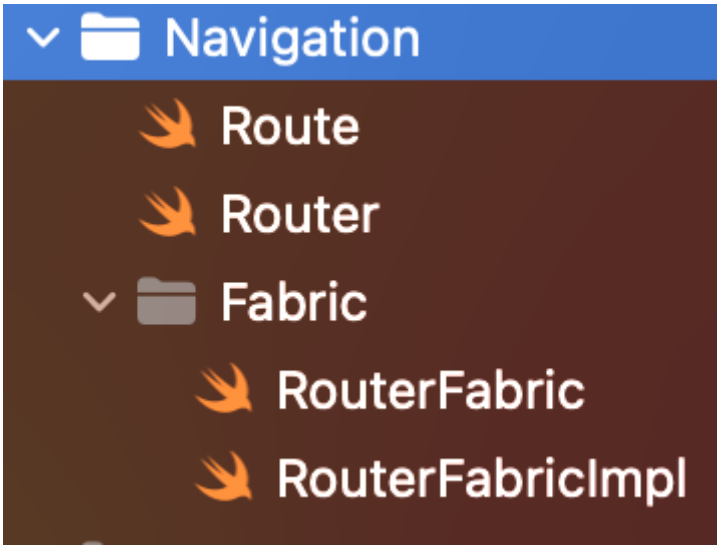


Рисунок 3.12 - Структура Navigation.

- Route являє собою маршрути модулів.
- Router містить логіку маршрутів між модулями.
- Fabric являє собою конструктор залежностей для модуля.

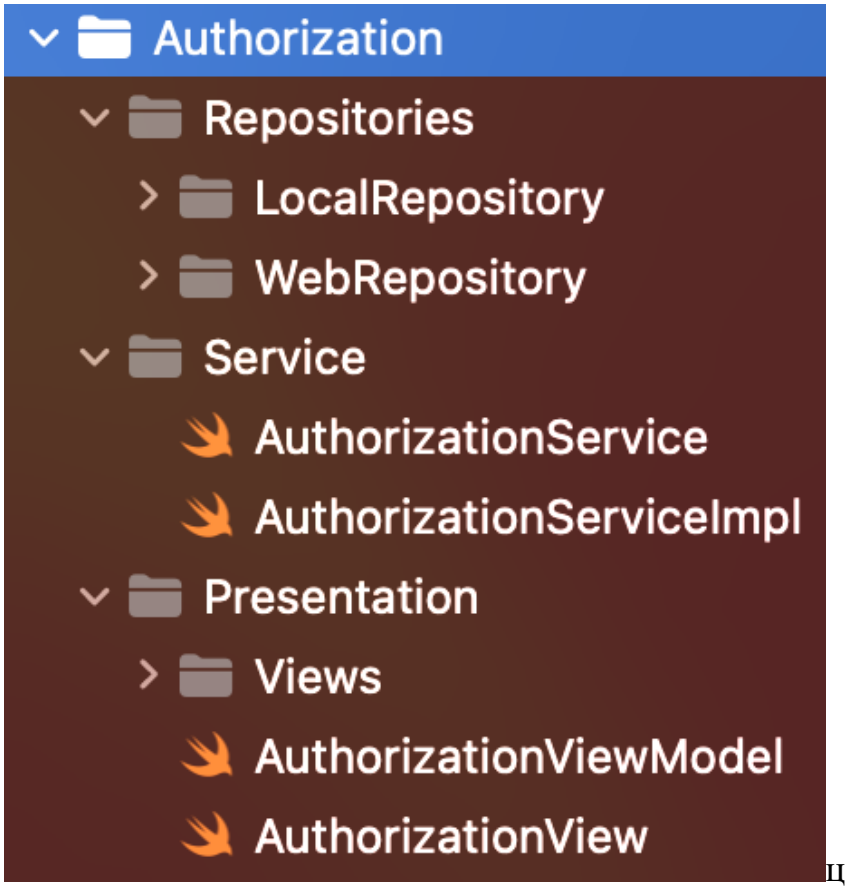
Основні модулі додатку:

- RootContent. Модуль, в кому вирішується, з якого екрану буде функціонувати додаток.
- Authorization. Модуль для авторизації в додатку з обліковим записом користувача.
- Sessions. Модуль для роботи із задачами, які має користувач.
- HomeProjects. Модуль для роботи із проектами користувача.
- SessionsLog. Модуль для роботи із завершеними задачами користувача.
- Projects. Те ж саме що і HomeProjects, але відрізняється дизайном та структурою. Для роботи з проектами.
- SessionDetails. Модуль для роботи із детальною інформацією задачі користувача.
- FreeSessions. Модуль для роботи із створеними користувачем задачами.
- Settings. Модуль для роботи із налаштуваннями додатку. Налаштування проводить користувач.
- Launch. Модуль для роботи із стартовим екраном додатку.

Всі модулі поділяються на три шари. Repository, Service, Presentation.

- Repository - містить логіку роботи модулю з даними. Ціль цього шару забезпечити модуль потрібною інформацією, яку він бере з серверу або ж з локальної бази даних чи іншого локального сховища.
- Service - містить бізнес логіку модулю та передає дані до Presentation з Repository.
- Presentation - містить в собі налаштування дизайну та логіку для роботи з даними, які потрібно представити користувачу в потрібному вигляді.

Приклад одного із модулів наведено на рисунку 3.13.



ц

Рисунок 3.13 - Приклад модулю авторизації та його шарів.

3.1.8 - Поширені модулі

Common - являє собою модуль, в якому містяться розширення до типів даних, класів, елементи дизайну, які використовуються 2 чи більше разів в додатку. Щоб уникнути дублювання коду і була можливість використати якийсь елемент, метод багато разів в залежності від потреби. Приклад структури Common зображено на рисунку 3.14.

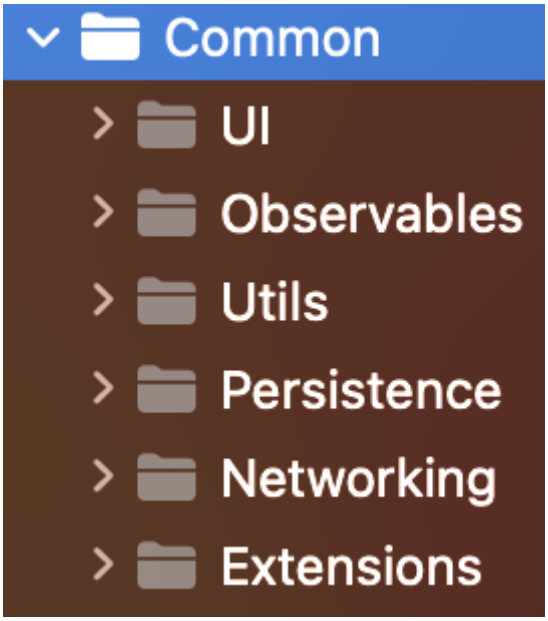


Рисунок 3.14 - Структура Common.

UI - являє собою модуль з елементами графічного дизайну, які використовуються на багатьох екранах.

Observables - модуль з сутностями, які реагують на певні фактори чи дії користувача.

Utils - модуль в якому налаштовані інтегровані бібліотеки та їх методи, які використовуються часто користувачем.

Persistence - модуль, який відповідає за локальну базу даних.

Networking - модуль, який відповідає за зв’язок із серверною частиною.

Extensions - модуль у якому знаходяться розширення до різних сутностей.

3.1.9 Ресурси

Модуль Resources відповідає контент, який представлений у додатку.

До Resources відноситься:

- Звуки

- Зображення

- Локалізація

- Шрифти

- Кольори

Приклад структури Resources зображено на рисунку 3.15.

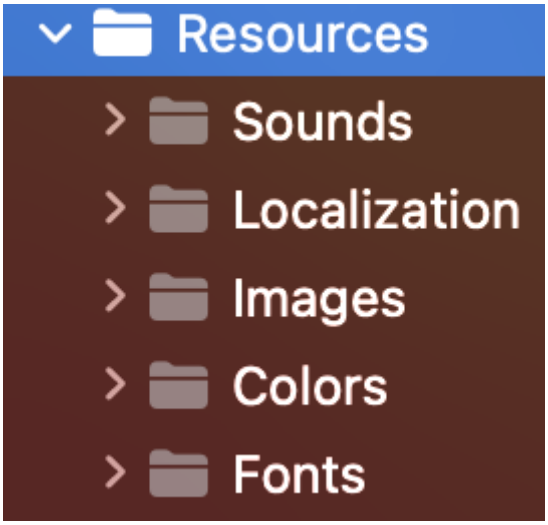


Рисунок 3.15 - Структура Resources.

3.2 Створення структур даних

Важливим стало питання правильних структур даних аби уникнути плутанини та змішування сутностей і їх властивостей. Для вирішення цієї проблеми були створені, такі сутності:

- User. Структура, яка містить дані про користувача.
- Project. Структура, яка відповідає за поверхневі властивості, які має проект.
- ProjectDetails. Структура, яка містить детальну інформацію щодо проекту.
- Session. Структура, яка відповідає за поверхневу інформацію створеної задачі для фотографа.
- SessionDetails. Структура, яка містить детальну інформацію про задачу для фотографа.
- Building, Floor, Apartment. Сутності, які відповідають за будівальні об’єкти та їх властивості.
- ScoutPhoto - Структура, яка відповідає за властивості зробленої фотографії.

Ці сутності використовуються майже у всіх модулях, тому правильно додані властивості впливають на кількість коду і надають змогу уникнути дублювання сутностей.

3.3 Інтегрування AI

В наслідок аналізу поставленої мети до швидкодії та роботи додатку з великою кількістю даних, було інтегровано нейронну мережу. Як було зазначено у другому розділі, для роботи з AI було обрано фреймворк CoreML від Apple для навчання моделі опрацьовувати дані.

Перевага цього підходу обробки даних, полягає в тому, що модель навчається і швидкість опрацювання покращується з кожною зробленою фотографією.

Після інтеграції до проекту, було налаштовано наступні конфігурації для правильної роботи моделі.

- Model Type. Тип моделі був вказаний як NeuralNetwork.

- Size. Поле яке налаштовується автоматично відповідно до обсягу моделі.
- Document Type. Вказано як Core ML Model.
- Availability. Налаштування які відповідають за доступність моделі на різних ОС. Так як, додаток працює на iOS, то параметром було вказано систему iOS версії не нижче 13.

Приклад налаштування моделі зображено на рисунку 3.16.

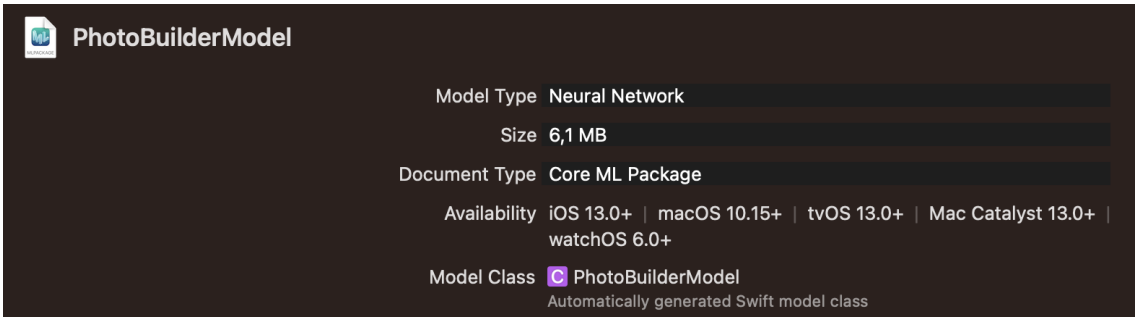


Рисунок 3.16 - Параметри моделі.

На вході модель приймає елемент Image з параметром Color(512 - 256). На виході отримуємо масив даних у вигляді масиву типу Float32. Приклад даних входу і виходу моделі наведено на рисунку 3.17.

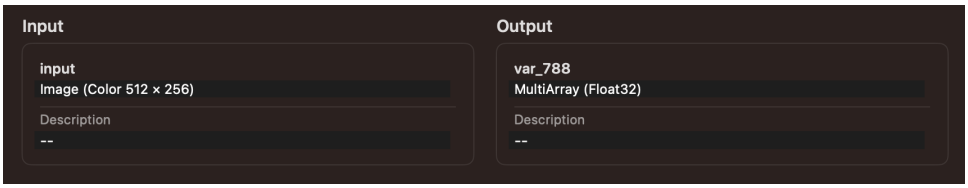


Рисунок 3.17 - Вхідні та вихідні дані моделі.

Також було налаштовано додаткову мета дату для моделі у вигляді параметрів. Ці параметри можна сприймати як результат моделі на опрацьовані фото. Якщо фотографія пройшла перевірку, користувач може працювати з цим фото далі, в іншому разі він отримує повідомлення, що фотографія не пройшла алгоритм опрацювання за одним із параметрів.

Параметри мета дати:

- AbsoluteObstruction. Означає, що на фото взагалі не можливо виділити об’єкт через перешкоду.
- Checked. Означає, що з фото в нормі і можна працювати далі.
- ElementsObstructionVision. Означає, що на фото якась перешкода, і не можливо знайти об’єкт на фото. Зазвичай це можуть бути машини, зайві інструменти, колони та інше.
- Foggy. Означає що якість на фото туманна.
- LensSlightlyCovered. Означає, що головний об’єктив камери прикритий.
- MisalignedLens. Означає, що об’єктив камери не стоїть не рівно.
- NonConstructionSite. Означає, що фото зроблене не на будівельному майданчику.
- OutOfFocus. Означає, що фотографія розмита.
- PersonObstructionVision. Означає, що людина потрапила на фото.
- Unknown. Невідома помилка, у випадках, якщо модель не може опрацювати фото.

Приклад налаштувань мета дати та додаткових конфігурацій, зображено на рисунку 3.18.

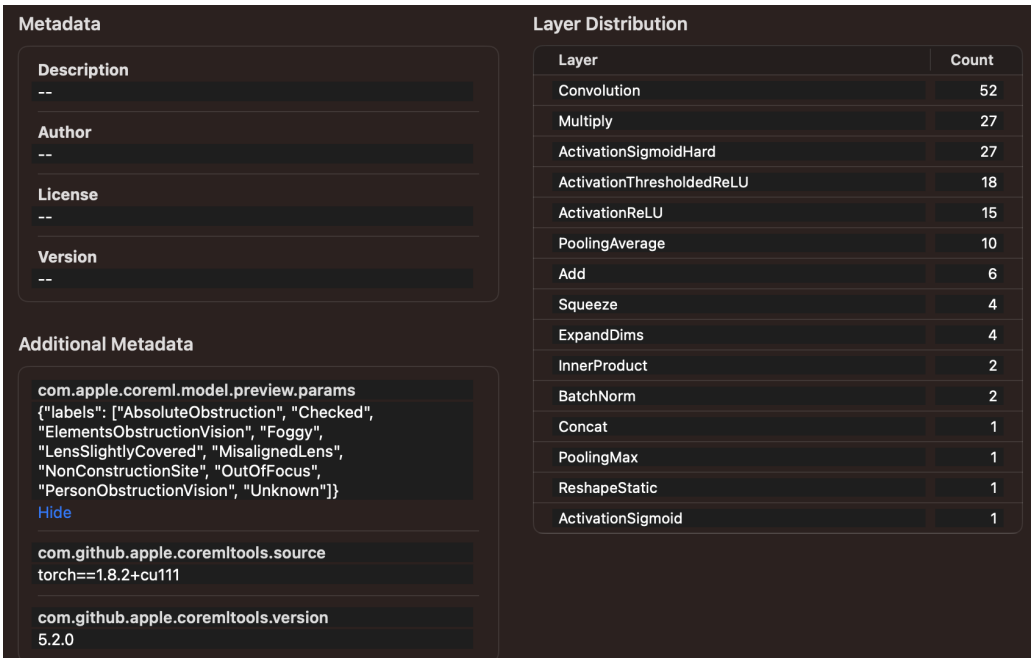


Рисунок 3.18 - Налаштування додаткових конфігурацій моделі.

3.3.1 Розробка програмного коду для опрацювання даних моделлю

При використанні CoreML, фреймворк сам генерує swift файл з властивостями та інтерфейсами для полегшення роботи та масивного коду.

PhotoBuilderModel це клас, який відповідає за моделлю нейронної мережі.

Клас містить наступні властивості:

- model: MLModel. MLModel інкапсулює методи прогнозування, конфігурацію та опис моделі. У більшості випадків використовується Core ML без прямого доступу до класу MLModel. Замість цього використовується зручний для клас-обгортка, який Xcode автоматично генерує при додаванні моделі.
- UrlOfModelInThisBundle: URL. Змінна яка вкзує на посилання моделі.

Інтерфейси:

- load

Метод створює екземпляр PhotoBuilderModel асинхронно з додатковою конфігурацією.
Завантаження моделі може зайняти деякий час, якщо вміст моделі не доступний негайно (наприклад, зашифрована модель). Використовується цей, коли виклик знаходиться в основному потоці.

Параметри:

- MLModelConfiguration: бажана конфігурація моделі.
- CompletionHandler: обробник завершення, який буде викликаний при успішному або неуспішному завершенні завантаження моделі.
- Prediction.

Метод прогнозування фото.

Параметри:

- Ввести у вигляді кольорового (kCVPixelFormatType_32BGRA) буферу зображення, розміром 512 пікселів завширшки та 256 пікселів заввишки.

- Повертає: об'єкт NSError, що описує проблему.
- Повертає: результат передбачення як PhotoBuilderModelOutput.

Структура ModelHelper була створена для зв’язку з моделлю нейронної мережі та додатку.

Щоб отримати доступ до інтерфейсів моделі, було інтегровано бібліотеку CoreML. Для роботи з фото обрано об’єкт UIImage. UIImage являє собою клас, який наслідується від NSObject з бібліотеки UIKit, відповідно було інтегровано бібліотеку UIKit до ModelHelper.

Для взаємодії з інтерфейсами моделі, було налаштовано наступні властивості. - modelConfiguration типу MLModelConfiguration, та встановлено параметр computeUnits як cpuOnly. Цей параметр вказує моделі, що потрібно використовувати для прогнозування.

- IgnoredErrors типу [ModelErrors]. Ця змінна використовується для процесу розробки, аби можна було робити фото для тесту не на будівельному майданчику.

- model типу PhotoBuilderModel. Це екземпляр моделі, який налаштовується з допомогою modelConfiguration.

ModelHelper містить наступні інтерфейси :

- runModel(on image: UIImage) -> ModelResult.
  На вході цей метод приймає фото, яке було зроблено камерою і отримано через мережу та викликає метод checkImage. На виході очікуєть дані типу ModelResult для розуміння чи фото пройшло алгоритм опрацювання.
- checkImage(_ image: CVPixelBuffer) -> ModelResult.

На вході метод приймає фотографію у вигляді CVPixelBuffer та на виході ModelResult. Цей метод працює з вихідними даними моделі та вказує результат опрацювання викликаючи model.prediction(input: image).

- isArraySizeIncorrect(_ output: [Double]) -> Bool.

На вході метод приймає масив чисел типу Double, масив являє собі вихідні значення моделі, як було зображено на рисунку 3.17.

- IsTestPassed(_ output: [Double]) -> Bool.

На вході метод приймає масив чисел типу Double ти на виході Bool. В методі проходить перевірка чи властивість моделі check є позитивною.

3.4 Створення інтерфейсу

Після аналізу у другому розділі задачі створення простого та зручного користувацького інтерфейсу, було інтегровано фреймворк SwiftUI, та весь дизайн було розроблено кодом і для перегляду результату механізм Canvas. На рисунку 3.19 наведено стартовий екран авторизації.

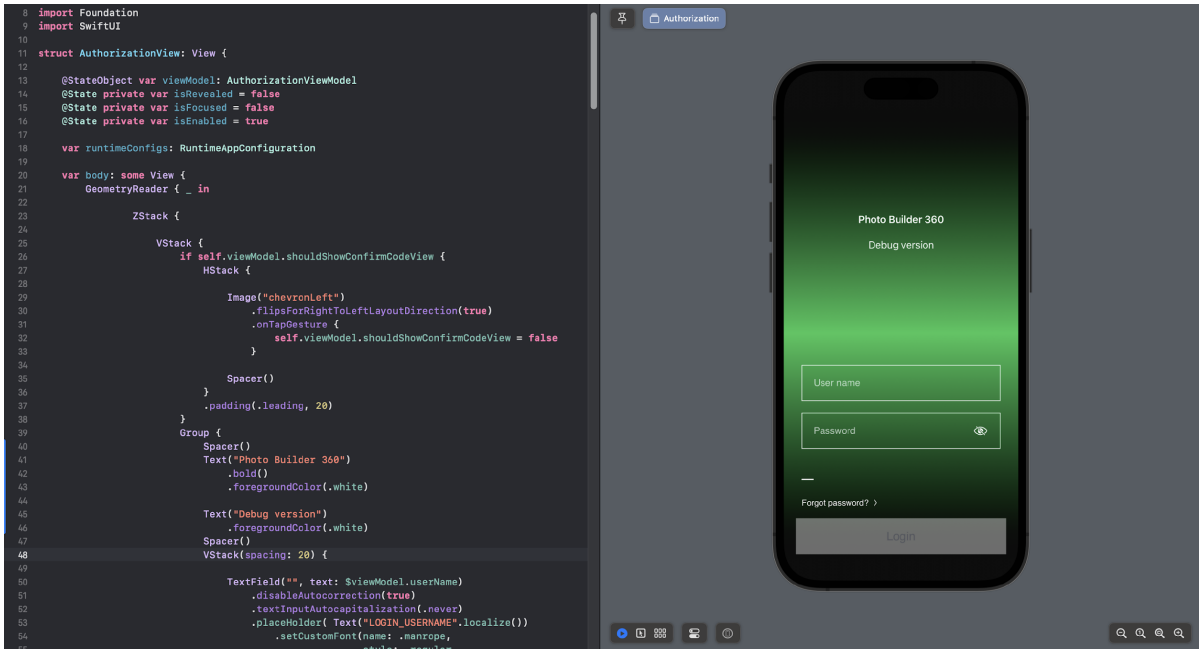


Рисунок 3.19 - Дизайн екрану авторизації.

Оновлений механізм Canvas в Xcode дає можливість переглянути дизайн одразу майже на всіх доступний моделях iPhone. На рисунку 3.20 представлено дизайн на різних моделях телефону.

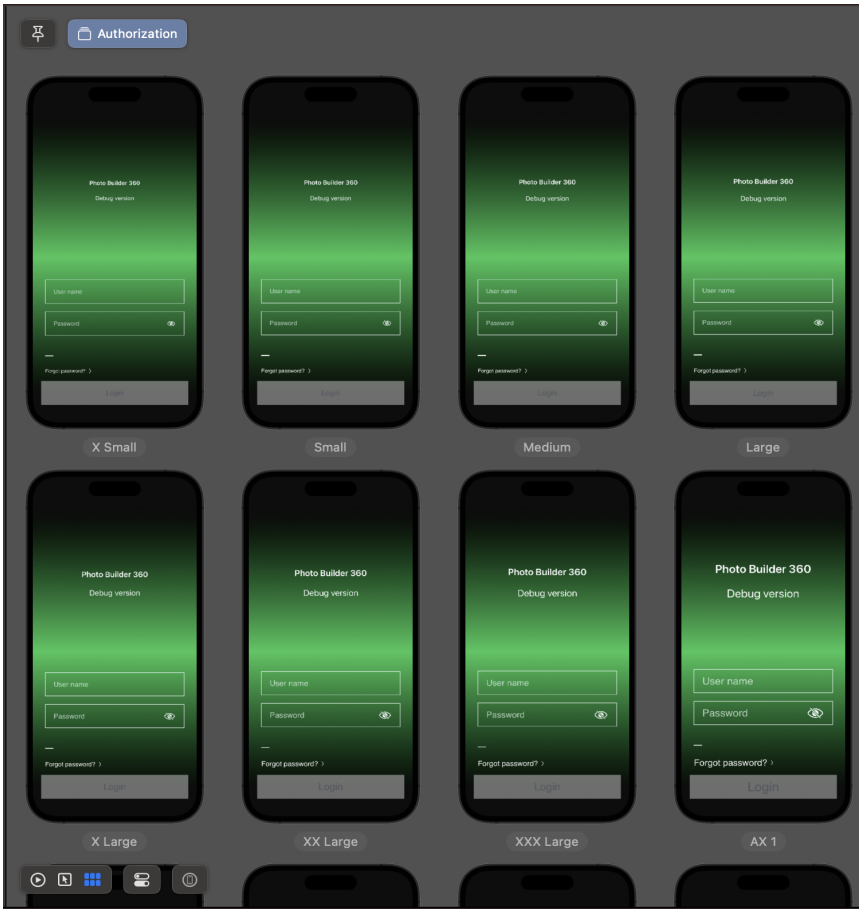


Рисунок 3.20 - Перевірка інтерфейсу на різних моделях.

3.4.1 Розробка програмного коду для дизайну

Для налаштування дизайну в коді використовується структура з відповідною назвою, яка наслідується від View. View - це тип, який представляє частину користувальницького інтерфейсу додатку і надає модифікатори, які ви використовуються для налаштування представлень.

Усі структури повинні містити змінну body типу some View. Для відображення дизайну в Canvas створюється структура типу PreviewProvider.

За допомогою SwiftUI всі екрани мають дуже схожу структуру, що дозволяє зменшити кількість коду та масивних файлів. Базова структура дизайну зображена на рисунку 3.21.

```swift
import Foundation
import SwiftUI

struct BaseView: View {

    var body: some View {
        NavigationView {
            GeometryReader { _ in
                EmptyView()
            } //: GeometryReader
        } //: NavigationView
    } //: Body
}

#if DEBUG
struct Base_Previews: PreviewProvider {

    static var previews: some View {
        return BaseView()
    }

}
#endif
```

Рисунок 3.21 - Базові налаштування екрану.

З допомогою вище зображених базових налаштувань вже можна створити новий екран в додатку.

На всіх екранах використовуються наступні властивості:

- Body. Головний контейнер, для представлення налаштованих дочірніх контейнерів на екрані.
- NavigationView. Представлення стеку екранів, що представляє видимий шлях в ієрархії навігації.
- GeometryReader. Контейнер, що визначає його вміст як функцію власного розміру та координатного простору.
- PreviewProvider. Попередній перегляд для будь-яких провайдерів, відкритих у редакторі вихідного коду. Xcode генерує попередній перегляд, використовуючи поточне призначення запуску як підказку для пристрою для

відображення. Наприклад, Xcode показує наступний попередній перегляд, якщо ви вибрали ціль iOS для запуску на симуляторі iPhone 12 Pro Max:

3.5 Розробка мережевого шару

Після аналізу задачі роботи додатку з сервером та вибору REST API для рішення цієї задачі, було вирішено використати та інтегрувати додаткові бібліотеки такі як:

- Alomofire.
- Combine.
- AlomofireNetworkActivityLogger.

Усі бібліотеки інтегровані в проект з допомогою Pods та інтегровані в код з допомогою команди import. Модуль Networking був встановлений як залежність в модулях, яким потрібні дані із сервера або ж щось надіслати до серверу. Для викликів API використовуються два основних посилання які залежать від середовища Debug або ж Release. Щоб зробити виклик API по конкретному заздалегідь спланованому маршрутом, який називається endpoint потрібно вказати параметри запиту та використати базове посилання. Приклад посилання за яким відбувається запит до серверу представлено в формулі 3.1

(HTTPS://BASE_URL) + API_ROUTER_CONST + ENDPOINT + PARAMS. (3.1)

BASE_URL - це базове точка доступу до сервера.

API_ROUTER_CONST - це константа, яка вказує який саме тип серверу потрібно використати.

ENDPOINT - вказує на якийсь конкретний запит.

PARAMS - параметри, які можуть вказувати на тип даних, фільтрацію чи подібне.

Приклад запиту авторизації на сервер зображено на схемі 3.1.

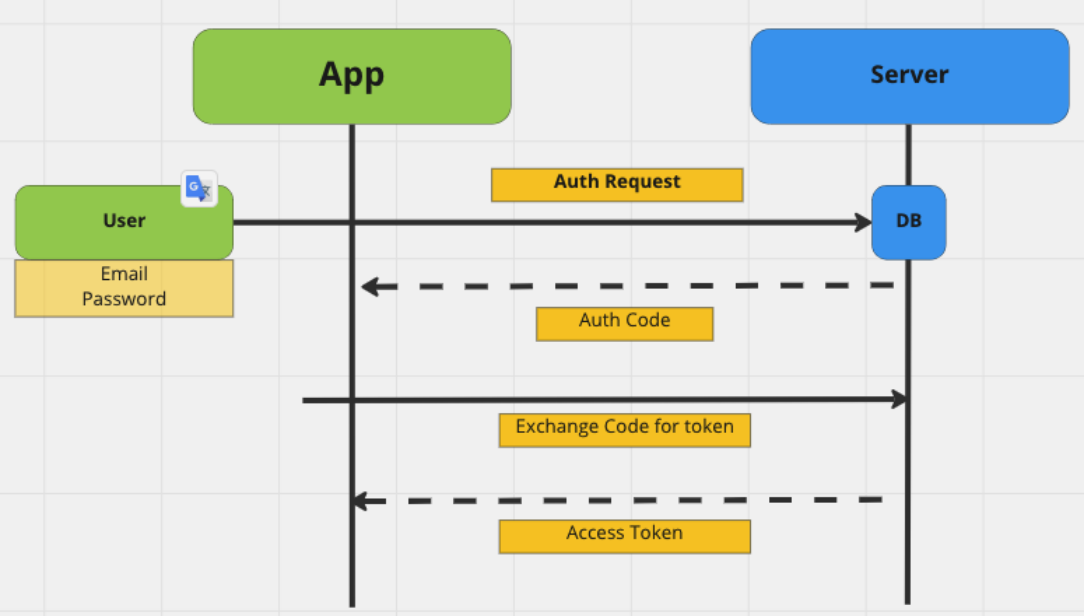


Схема 3.1 - Запит авторизації на сервер.

3.5.1 Розробка програмного коду Networking

Для роботи з сервером було розроблено дві сутності: NetworkClient та RequstInfo

NetworkClient - це клас, який наслідується від протоколу NetworkProvider і реалізує його методи.

Цей клас містить реалізацію наступних методів:

1) RequestUpload().

Метод робить виклик API з допомогою endpoint та Session екземпляру. За допомогою цього методу, контролюється прогрес завантаження фото на сервер.

Параметри входу:

- RequsetInfoConvertible. Структура, яка містить властивості запиту та параметри.
- Completion. Зворотній виклик функції.

Параметри виходу :

- Data.

- Error.

2) Request().

Метод створений для запитів до серверу без зворотнього виклику.

Параметри входу:

- RequsetInfoConvertible.

Параметри виходу:

- Data.
- Error.

Кожний із методів отримує відповідь серверу у вигляді Data. З використанням функції tryMap тип Data можна розгорнути у вигляді результатів:

- Success(let data). Позитивна відповідь від сервера, там повернення відповідних даних конкретного типу, який очікувався від запиту.
- Failure(let error). Негативна відповідь, яка повертає помилку з вказаним кодом та причиною.

RequestInfo - це структура, яка містить властивості для запиту.

Властивості RequestInfo:

- Url: URL
- Endpoint: String
- Parameters: [String: Any]
- Encoding: JSONEncoding
- Headers: HTTPHeaders
- Interceptor: RequestInterceptor
- RequestModifier: Session.RequestModifier.
- Method: HTTPMethod
- Data: Data
- ShouldContainAuthToken: Bool

Всі ці властивості необхідні для коректної роботи запиту.

3.5.2 Розробка програмного коду для мережевої роботи з камерою 360

Фотографії в додатку виконуються з використанням зовнішньої камери 360, з модулем WiFi. Для таких запитів, потрібно налаштувати конфігурацію для наступник параметрів.

- NSExeptionAllowInsecureHTTPLoads – Yes
- NSIncludesSubdomains – Yes

Ці параметри дозволяють надсилати HTTP запити до камери, та отримувати фотографію.

Основна логіка роботи з камерою належить сутності CameraHelper та CameraManager.

Протокол CameraHelper містить такі інтерфейси:

- Connect(). Метод, який встановлює зв’язок з камерою та на виході отримує статус підключення камери.
- GetTakePictureState(). Метод отримує статус зробленої фотографії з камери.
- TakePicture(). Метож, який робить запит до камери з комндою зробити фотографію, та на виході повертає фотографію у вигляді UIImage.
- GetInfo(). Метод, який робить запит до камери та повертає на виході інформацію про камеру, таку як модель, версія, статус підключення, положення і тд.

В процесі фотографування, статус камери змінюється відносно до процесу. Статуси камери перераховані в Enum:

- InProgress. Статус в процесі зйомки.
- Done. Статус завершення з готовим фото.
- Failed. Статус повідомляє про помилку.

3.6 Розробка відображення архітектурних планів.

Внаслідок аналізу задачі відображення інформації фотографу в додатку було вирішено, що плани в додаток надходитимуть з хмарного сервісу AWS. Додаток за запитом getFloorPlans отримує таку інформацію як:

- Назва проєкту
- Назва будівлі
- Номер поверху
- Посилання на план.

Після того, як посилання на план отримано, відбувається завантаження плану у вигляді файлів формату svg, після чого збереження планів в локальну базу даних.

За відображення архітектурних планів відповідає модуль Apartment.

Цей модуль містить три шари передачі даних:

- Repository. Відповідає за збереження чи завантаження планів із локальної бази даних.
- Service. Відповідає за передачу даних із Repository до Presentation.
- Presentation. Відповідає за відображення планів на екрані.

Оскільки програмі потрібно опрацьовувати натиски, свайпи, збільшення чи зменшення плану, то використовується об’єкт WKWebView. Об'єкт WKWebView - це власне подання платформи, яке ви використовується для безперешкодного включення веб-вмісту в інтерфейс додатка. Веб-представлення підтримує повноцінний веб-браузер і представляє вміст HTML, CSS та JavaScript поряд з власними представленнями додатку. Для опрацювання якихось дій з svg файлом, використовуються сутності, які працюють з javaScript.

Модуль SVGHelper готує файл svg до відображення і проводить маніпуляції у файлі з використанням javaScript. Після того, як файл налаштовано, в метадату файлу, вноситься додатковий контент у вигляді точок. Точки на планах позначають місця, де фотографу потрібно поставити камеру та зробити фото.

3.6.1 Розробка програмного коду для роботи з планами будівель.

Для відображення апартаментів, в шарі Presentation модуля Apartment, ApartmentView та ApartmentViewModel.

ApartmentView має дочірнє WebView, що і відображає план. У ApartmentViewModel знаходиться логіка, яка відповідає за зв’язок з допоміжними модулями та логікою екрану ApartmentView.

Коли відбувається натиск на план чи будь-яка інша дія , WebView надсилає повідомлення до програми, яке опрацьовується в класі SVGWebView, який наслідується від WKScriptMessageHandler.

SVGWebView містить наступні розширені методи:

1. GetDocumentDirectory().для збереження самого файлу в директорії.
2. UserContentController().

   Параметри входу:

- WKUserContentController. Об'єкт для управління взаємодією між кодом JavaScript і веб-представленням, а також для фільтрації вмісту у веб-представленні.
- WKScriptMessage. Об'єкт, який інкапсулює повідомлення, відправлене кодом JavaScript з веб-сторінки.

Коли отримано повідомлення з WebView, воно опрацьовується на SWGWebView для оновлення вмісту веб-представлення і маніпуляцій з файлом. Наприклад після натискання на точку, програма отримує повідомлення з WebView і оновлює вміст, аби план збільшився в розмірах.

3. LoadSVG(). Метод відповідає за завантаження вмісту на WebView за посиланням.

Параметри на вході:

- Url

- [Point]. Point – це сутність, яка містить інформацію про розташування точки на плані.

4. SetupUI(). Метод налаштовує всі елементи UI на екрані.
5. SelectPoint(). Реагує на натиск точки.
6. ColorPoint(). Змінює колір точки відповідно до логіки.

Функції 4-6 вказують об’єкту webView з допомогою команди evaluateJavaScript, що потрібно зробити з вмістом webView. EvaluateJavaScript - обчислює заданий рядок JavaScript.

Параметри:

- JavaScriptString. Рядок JavaScript для обробки.

- CompletionHandler .Блок-обробник, який виконується після завершення оцінки скрипта. Метод викликає блок незалежно від того, чи завершилася обробка скрипта успішно чи ні. Блок не має значення, що повертається і приймає наступні параметри:
- Object. Результат виконання скрипту, або нуль, якщо виникла помилка.
- Error. Нуль при успішному завершенні, або object error з інформацією про проблему.

3.7 Інтегрування локальної бази даних

Як було зазначено у другому розділі, для вирішення задачі роботи додатку без мережі, було вирішено обрати CoreData.

З допомогою інструментів CoreData було створено моделі даних, які зберігаються в додатку.

Прикладами таких моделей є:

- UserObject
- SessionObject
- ProjectObject
- PlanObject

- PhotoObject

Приклад інтерфейсу для створення моделі даних та її властивостей зображено на рисунку 3.22.

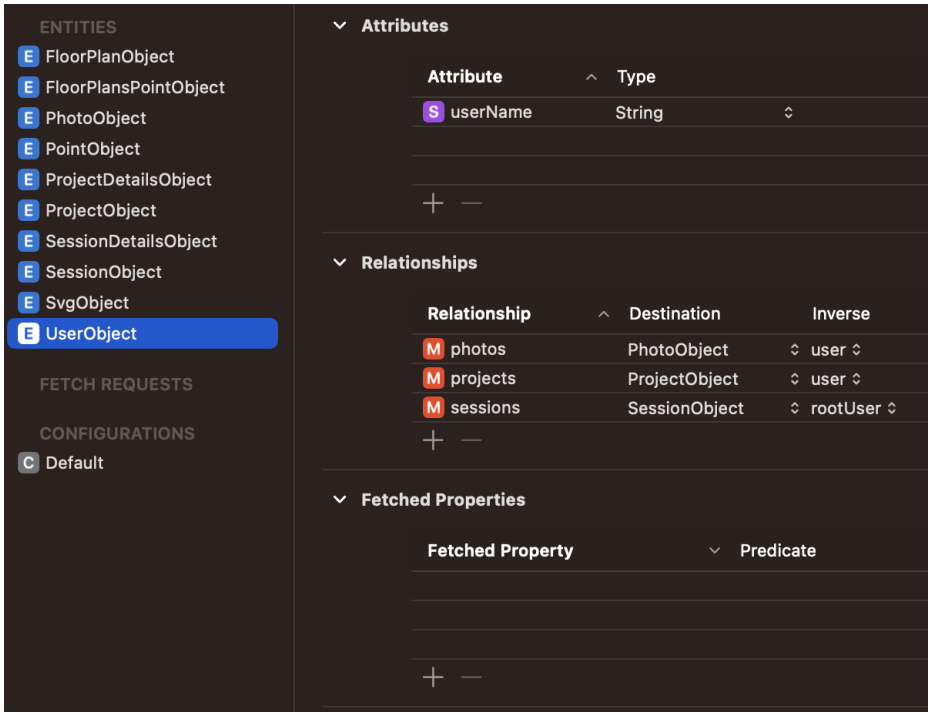


Рисунок 3.22 - Моделі даних в CoreData.

3.7.1 Розробка програмного коду для локальної роботи

За налаштування CoreData відповідає модуль Persistence, а саме клас CoreDataStack.

CoreDataStack має такі властивості:

- IsStoredLoaded: Bool. Інформує про стан бази даних та готовність для використання.
- BgQueue: DispatchQueue. Черга для операцій налаштування бази даних на глобально потоці.
- Container: NSPersistentContainer. Контейнер, який інкапсулює стек основних даних у додатку.
- ViewContext: NSManagementObjectContext. Об'єктний простір для маніпулювання та відстеження змін керованих об'єктів.

При ініціалізації CoreDataStack відбувається налаштування локального сховища з параметрами:

- Directory: FileManager.SearchPathDictionary.

- DomainMask: FileManager.SearchPathDomainMask.

- Version. Версія локального сховища для автоматичного мігрування.

Властивість Container ініціалізує NSPersistentContainer за назвою моделі сховища на посиланням в каталозі директорій. Після налаштування CoreDataStack локальне сховище стає готове до використання.У кожному модулі в шарі Repository, який працює з локальними даними в додатку є залежність від NSPersistentContainer.

Після ініціалізації Container в модулі, Repository може використовувати наступні інтерфейси:

- Save()
- Update()
- Remove()
- Fetch()

Це методи дженеріки, які працюють на пряму з Container і керують моделями даних в CoreData. За допомогою цих методів, модуль може дістати потрібні дані або зробити іншу операцію просто викликавши container в репозиторії та потрібний метод.

3.8 Висновок

У даному розділі було розроблено архітектуру та модулі додатку. Розроблено і описано функціонал головних та додаткових модулів, їх зв’язок та призначення. Реалізовано методи, необхідні для функціонування згідно технічних вимог проєкту. Інтегровано локальне сховище та додаткові зовнішні сервіси для роботи з мережею, фотографіями, дизайном, збирання аналітики, логування роботи додатку та обробки даних.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вибір способу тестування програми

Для тестування розробленої програми розглядалось два шляхи тестування:

- Автоматизоване тестування
- Ручне тестування

Автоматизоване тестування мобільних додатків – це підхід, який передбачає використання інструмента для пошуку помилок і оцінки продуктивності продукта. Для початку QA-спеціаліст вибирає набір інструментів, складає план тестування, виконує план і збирає отримані дані після тестування.

Переваги автоматизованого тестування :

- Запуск тестів в будь який час. Такий вид тестування підходить, коли необхідно проводити тести 24 години на добу

- Повторне використання. Розробник може повторити тестування, як тільки буде змінено програмний код.

- Підвищена швидкість виправлення помилок на ранніх етапах розробки. Наприклад помилки, які пов’язані з пам’яттю простіше знайти після декількох одночасних тестів.

Недоліки автоматизованого тестування :

- Витрати на обслуговування. Тести складно оновлювати і потребують багато часу.

- Обмеження, які має інструмент для тестування.

- Втрата ефективності при тестуванні дрібного функціоналу.

Ручне тестування мобільних додатків - це базовий підхід для тестування програм, який покладається тільки на людину. Під час ручного тестування, передбачається, що спеціаліст виконає тест самостійно від початку до кінця без використання автоматичних інструментів.

Переваги ручного тестування:

- Точність. При ручному тестуванні, спеціаліст може точніше зрозуміти, яким буде кінцевий результат функціонала для користувача.

- Краще підходить до складних процедур тестування. Наприклад, тестуванні ігор. Тестування такого продукту потребує від користувача натисків, свайпів, нахилу девайсу.

- Надає змогу отримати краще розуміння проблеми та вибору рішення на концептуальному рівні.

4.2 Вимоги до розробленого програмного забезпечення

Головні етапи тестування містять наступні пункти:

1) Адаптація дизайну на різних моделях iPhone.
- Відповідність графічних елементів на різних девайсах.
- Перевірка активних елементів на дії користувача.
- Швидкість відгуку графічного інтерфейсу.
- Перевірка на роботу спеціальних жестів.
2) Ресурси системи.
- Контроль над втратами пам’яті.
- Перевірка на вплив програми на батарею пристрою.
- Навантаження на CPU.
3) Контроль підтримки версій iOS.
- Перевірка роботи програми на мінімальній версії iOS 16 та на найновіші iOS 16.1
4) Реакція програми на роботу із зовнішніми сервісами.
- Робота з сервісами зовнішніх бібліотек.
- Робота з API.
- Робота з WiFi.
5) Перевірка апаратних допоміжних пристроїв.

- Перевірка роботи підключення до камери.
- Перевірка на стабільність підключення програми до камери.
- Перевірка фотосесії.

4.3 Розробка тест кейсів

Згідно до вимог продукту було проведено тестування програмного забезпечення. Для забезпечення якості програми необхідно створити сценарії тест кейсів для тестування головних можливостей додатку.

Усі тести та моніторинг ресурсів, графіків проводився в середовищі Xcode з використанням стандартних інструментів Profile. Приклад вибору інструментів з Xcode зображено на рисунку 4.1.

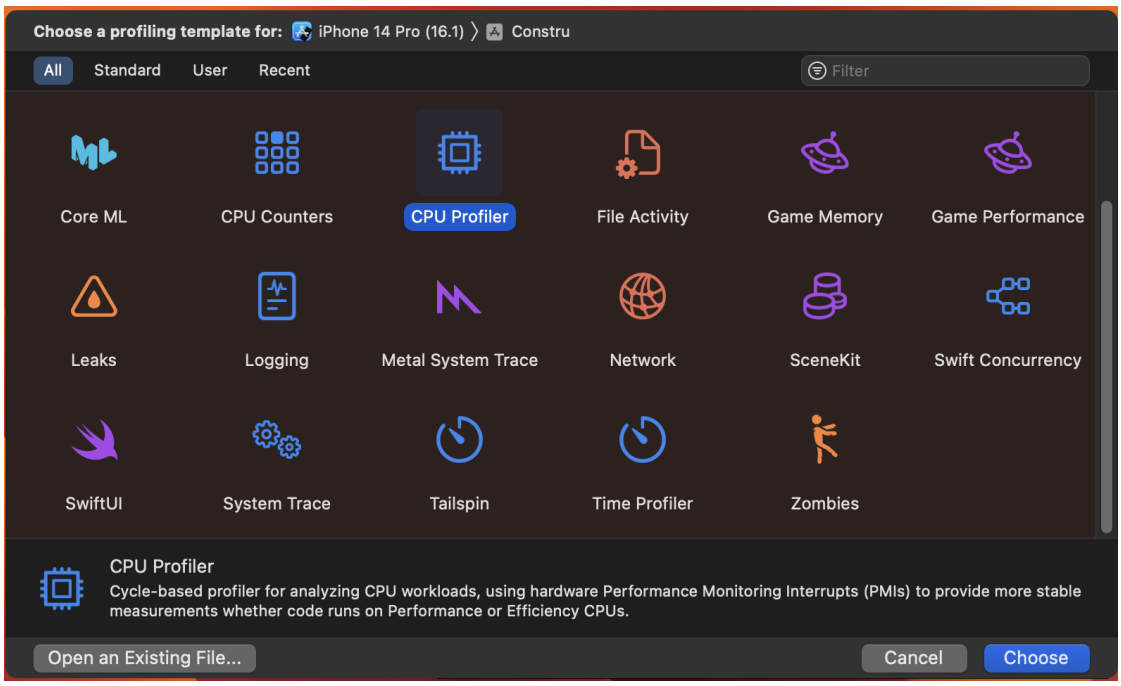


Рисунок 4.1 - Інструменти для тестування з Xcode.

Основні тест кейси для перевірки роботи програми представлені у таблиці 4.1.

Таблиці 4.1 - Сценарії для перевірки функціоналу

Назва тесту	Крок	Очікування
Перевірка роботи програми на версіях iOS не нижче 16.	Встановити програму на пристрій із iOS 16 та вище, запустити додаток та завантажити план будівлі.	Додаток працює, після скачування плану можна його відкрити. Додаток працює стабільно, збоїв немає.
Перевірка додатку на аварійні завершення роботи.	Відкрити програму, завантажити план, зробити не менше 200 фото і повернутись в меню.	Додаток працює стабільно без збоїв.
Перевірка графічного інтерфейсу на різних моделях девайсу.	Відкрити програму на iPhone SE та iPhone 13 Pro max.	Дизайн відображається згідно розроблених макетів.
Перевірка підключення камери.	Запустити додаток, змінити мережу WiFi на камеру, зробити фото, змінити точку доступу WiFi.	Додаток працює стабільно під час зміни точок доступу в мережі WiFi.
Перевірка навантаження на пам’ять девайсу.	Запустити додаток, завантажити план, підключитись до камери, відкрити план будівлі, зробити 2000 фото.	Додаток працює стабільно, збоїв немає.

Продовження таблиці 4.1.

Перевірка правильності роботи алгоритмів нейронної мережі.	Відкрити план будівлі, зробити фотографію в темному приміщенні, зробити фотографію під неправильним кутом об’єктива, зробити фотографію з людиною на фото.	Додаток працю стабільно. Нейронна мережа попереджає про помилки: 1) Фото занадто темне 2) Об’єктив розміщений не правильно 3) На фото люди.
Перевірка навантаження на CPU	Запустити програму, відкрити архітектурний план, зробити 10 фото.	Додаток працює стабільно без збоїв.

- Перевірка навантаження на пам’ять - пристрою відповідає поставленим вимогам. Але при перевірці об’єктів при в пам’яті були знайдені деякі об’єкти, які не видалялись із пам’яті та займали надлишкові мегабайти. Проблему було вирішено з використанням механізму ARC та перетворень сильних посилань на слабкі. Приклад надлишкових посилань на об’єкти в пам’яті зображено на додатку А.

- Перевірка навантаження на CPU відповідає поставленим вимогам. Після 10 зроблених фотографій, та відправці їх на сервер, зайнятість CPU сягає 3-20%. Приклад використаних ресурсів зображено на рисунках 4.2, 4.3, 4.4

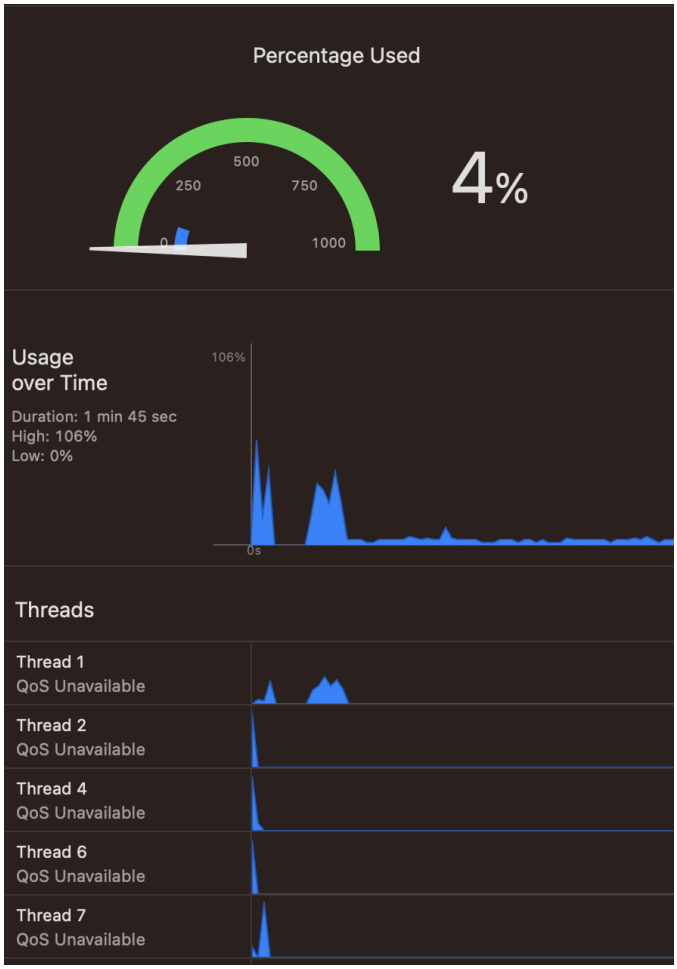


Рисунок 4.2 - Навантаження на CPU.

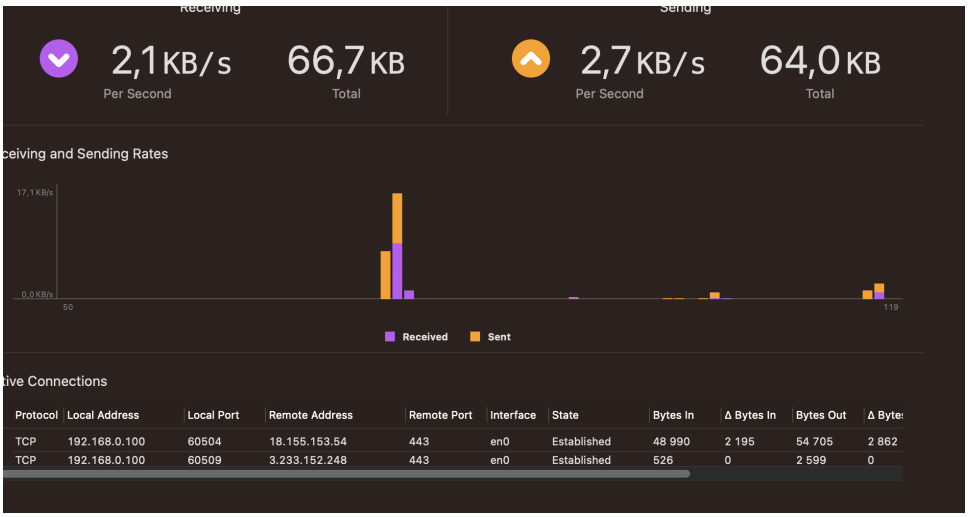


Рисунок 4.3 - Навантаження на мережу.

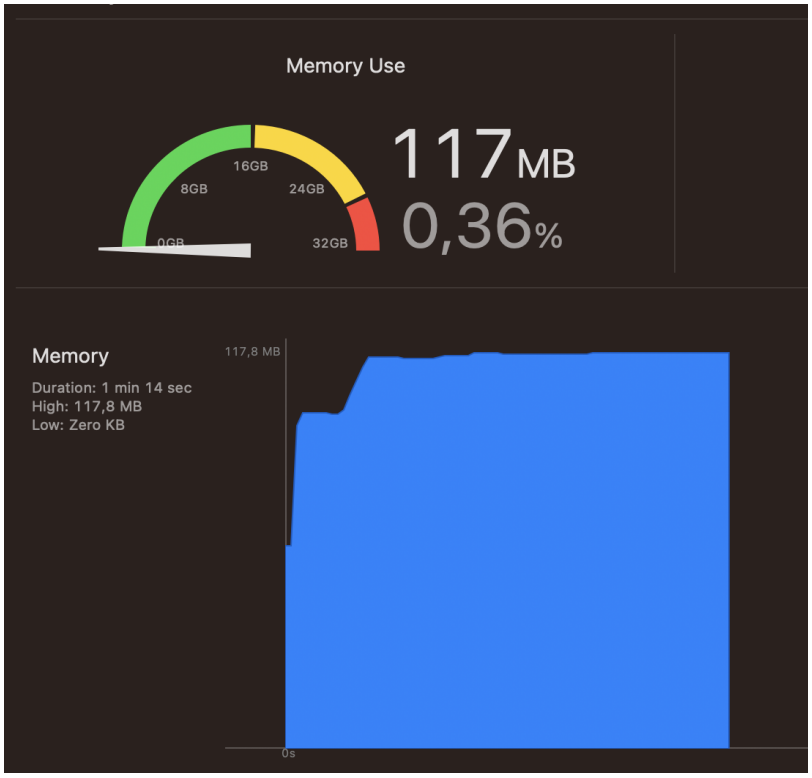


Рисунок 4.4 - Навантаження на пам’ять девайсу.

4.4 Висновок

У даному розділі було обґрунтовано вибір способу тестування додатку. Складено план та описано процес тестування програмного забезпечення. З наведених вище рисунків та аналізу роботи додатку під час тестування, було отримано такі результати: Слабке навантаження на пам’ять телефона під час обробки інформації. Під час процесу обробки фотографій, параметр змінював значення з 85 до 117 мегабайт. Це означає, що після обробленої фотографії, пам’ять, яка була відведена на фотографію звільняється і використовується для іншої. Слабке навантаження на CPU та високий показник швидкодії додатку. З рисунку 4.2 можна спостерігати, що середній показник навантаження на CPU знаходиться в зеленій зоні та сягає 4-10%. З графіків також можна побачити розподілення процесів по різним потокам, що забезпечує паралельність виконання багатьох процесів, прискорюючи роботу додатку. Стабільна робота додатку. Успішність виконання всіх тест кейсів, свідчить про стабільну роботу додатку без збоїв.

5. ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит результатів розробленого мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту

Як було зазначено раніше, попри стрімкий розвиток технологій, які використовуються на будівництвах, все ще актуальні такі проблеми, як висока конкуренція, низька рентабельність, недостатня якість робіт, збільшений ризик перевищення термінів будування об’єктів тощо.

Одним із способів вирішення цих проблем є використання апаратно будівельного програмного забезпечення, а саме мобільних додатків, які можуть покращувати та полегшувати той чи інший процес на будівництві. Разом з тим, фахівці відмічають певний розрив між існуванням мобільних будівельних додатків та їх використанням. Тому актуальним питанням є не тільки створення нового програмного забезпечення, але й повне використання існуючого.

У зв’язку з цим, у виконаній нами магістерській роботі було розроблено ідеологію, алгоритми та зроблено програмно-апаратну реалізацію мобільного додатка для швидкого збирання, зберігання інформації про параметри стану будівельного об’єкту, обробки цих даних з використанням штучного інтелекту та надсилання цих даних до серверної частини.

Для цього було досліджено і проаналізовано сучасні системи та популярні підходи розробки програмного забезпечення на платформі IOS та методи збирання і обробки інформації з використанням нативних інструментів платформи та штучного інтелекту.

Для встановлення комерційного потенціалу розробленого нами мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту було запрошено 3-х експертів – кандидатів технічних наук, доцентів Кулика Я.А., Гармаша В.В. та Маслія Р.В.

Встановлення комерційного потенціалу розробленого нами мобільного додатка було здійснено за критеріями, наведеними в таблиці 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання технічного рівня та комерційного потенціалу будь-якої розробки і їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 5.1

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів	
Ринкові перспективи						
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів		Експлуатаційні витрати значно нижчі, ніж в аналогів
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок		Великий ринок з позитивною динамікою

Продовження таблиці 5.1

7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження таблиці 5.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Запрошені експерти оцінили розроблений нами мобільний додаток для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту таким чином (див. таблицю 5.2):

Таблиця 5.2 – Результати технологічного аудиту розробленого мобільного додатка (за шкалою оцінювання 0-1-2-3-4)

Критерії	Прізвище, ініціали експертів		
	Кулик А.Я.	Гармаш В.В.	Маслій Р.В.
	Бали, що їх виставили експерти:		
1	4	3	3
2	3	3	4
3	3	3	4
4	4	4	4
5	4	3	4
6	3	3	3
7	3	3	3
8	4	3	3
9	4	3	3
10	3	3	4
11	4	4	4
12	3	3	4
Сума балів	$СБ_1 = 42$	$СБ_2 = 38$	$СБ_3 = 43$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{42+38+43}{3} = \frac{123}{3} = 41,00$		

Встановлення комерційного потенціалу розробленого нами мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з

використанням штучного інтелекту будемо здійснювати на основі рекомендацій, наведених в таблиці 5.3.

Таблиця 5.3 – Рівні комерційного потенціалу будь-якої наукової розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Оскільки середньоарифметична сума балів, що їх виставили експерти, складає 41,00 балів, то це свідчить, що розроблений нами мобільний додаток для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту має рівень комерційного потенціалу, який вважається «високим».

Це пояснюється тим, що нами запропоновано програмно-апаратну систему для швидкого та безперервного збирання інформації про параметри стану будівельного об’єкту та обробки цих даних і представлення у вигляді фотографій для порівняння їх в контексті з співставленим до планів проекту.

5.2 Розрахунок витрат на розроблення мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту

При розробленні мобільного додатка були зроблені певні витрати. Зокрема:

А). Основна заробітна плата $З_о$ розробників, яка визначається за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \text{ грн,} \qquad (5.1)$$

де М – місячний посадовий оклад розробника, грн; приймемо, що

М = (6700…23000) грн/місяць;

$Т_р$ – число робочих днів в місяці; приймемо $Т_р$ = 20 день;

t – число днів роботи розробників.

Зроблені розрахунки зведемо до таблиці 5.4:

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на оплату праці, грн
1. Науковий керівник магістерської роботи	20500	1025	20 годин	≈ 3417
2. Магістрант-студент-виконавець	2000 (беремо 6700)	335	84	≈ 28140
3. Консультант з економічної частини	19000	950	1,5 години	≈ 238 (при 6-годинному робочому дні)
Загалом				$З_о$ = 31795 грн

Б). Додаткова заробітна плата $З_д$ розробників розраховується як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_д = \alpha \cdot З_о = (0{,}1...0{,}12) \cdot З_о . \qquad (5.2)$$

Приймемо, що α = 0,11. Тоді для нашого випадку отримаємо:

$$З_д = 0{,}11 \times 31798 = 3497{,}45 \approx 3498 \text{ грн.}$$

В). Нарахування на заробітну плату $НЗП_{зп}$ розробників (дослідників) розраховуються за формулою:

$$НЗП_{зп} = (З_о + З_д) \cdot \frac{\beta}{100}, \qquad (5.3)$$

де β – ставка обов'язкового єдиного внеску на державне соціальне страхування, %. β = 22%. Тоді:

$$НЗН_{зп} = (31795 + 3498) \times 0{,}22 = 7764{,}46 \approx 7765 \text{ грн.}$$

Г). Амортизація основних засобів А, які використовувались під час виконання цієї роботи:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \text{ грн}, \qquad (5.4)$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що $Н_а$ = (2,5...25)%;

Т – термін використання основних засобів, місяці.

Зроблені розрахунки зведено в таблицю 5.5.

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
1. Комп'ютерна техніка, обладнання тощо	47000	25	3,1 (при 85% використанні	2580,10 ≈ 2581
2. Приміщення університету, кафедри	17000	3,5	3,1 при 90% використанні	138,34 ≈ 139
Всього				А = 2720 грн

Д). Витрати на матеріали М розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \text{ грн.,} \qquad (5.5)$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг; $Ц_i$ – вартість матеріалу *і*-го јнайменування; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); $В_i$ – маса відходів матеріалу *і*-го найменування; $Ц_в$ – ціна відходів матеріалу *і*-го найменування; n – кількість видів матеріалів.

Е). Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ грн,} \qquad (5.6)$$

де $Н_і$ – кількість комплектуючих *і*-го виду, шт.; $Ц_і$ – ціна комплектуючих *і*-го виду; $К_і$ – коефіцієнт транспортних витрат, $К_і$ = (1,1…1,15); n – кількість видів комплектуючих.

Під час виконання роботи загальні витрати на матеріали та комплектуючі склали приблизно 900 грн.

Ж). Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д}, \qquad (5.7)$$

де В – вартість 1 кВт-год. електроенергії, в 2022 р. В ≈ 3,0 грн/кВт;

П – установлена потужність обладнання, кВт; П = 1,0 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 214 годин;

$К_п$ – коефіцієнт використання потужності; $К_п < 1 = 0,77$.

$К_д$ – коефіцієнт корисної дії, $К_д = 0,64$.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д} = \frac{3 \cdot 1,0 \cdot 214 \cdot 0,77}{0,64} = 772,40 \approx 773$$ грн.

И). Інші витрати $В_{інш}$ можна прийняти як (50…300)% від основної заробітної плати розробників, тобто:

$$В_{інш} = (0,5....3) \times З_о. \qquad (5.8)$$

Для нашого випадку отримаємо:

$$В_{інш} = 1,1 \times 31795 = 34974,50 \approx 34975 \text{ грн.}$$

К). Сума всіх попередніх статей витрат складає витрати на виконання нашої роботи (безпосередньо розробником-магістрантом) – В.

$$В = 31795 + 3498 + 7765 + 2720 + 900 + 773 + 34975 = 82426 \text{ грн.}$$

Л). Загальні витрати на розроблення та можливе впровадження розробленого нами мобільного додатка $В_{заг}$ розраховуються за формулою:

$$В_{заг} = \frac{В}{\beta}, \qquad (5.9)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання цієї роботи. Можна прийняти, що, $\beta \approx 0{,}88$ [1], оскільки робота потребує незначного доопрацювання.

Тоді: $В_{заг} = \frac{82426}{0{,}88} = 93665{,}91$ грн або приблизно 94 тисячі грн.

Тобто прогнозовані загальні витрати на розробку та можливе впровадження (комерціалізацію) розробленого нами мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту становлять приблизно 94 тисячі грн.

5.3 Розрахунок економічного ефекту від можливої комерціалізації нашої розробки

Економічний ефект від впровадження та можливої комерціалізації розробленого мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту пояснюється його значно кращими функціональними можливостями. Тому нашу розробку можна реалізовувати на ринку дещо дорожче, ніж аналогічні за функціями розробки.

У 2022 році подібний за функціями мобільний додаток разом з додатковими мобільними пристроями, масштабу проєкту тощо вартував на ринку (залежно від масштабу будівельного проєкту) до \$4000 × 12 × 40 = 1900 тисяч грн (при курсі національної валюти 1 \$ = 40 грн). Для подальших розрахунків приймемо за основу середню ціну розробки в 1000 тисяч грн). Тоді розроблений нами мобільний додаток можна буде реалізовувати на ринку в середньому приблизно за 1200 тисяч грн або на 200 тисяч грн дорожче.

Аналіз ринку також показав, що потенційна кількість замовників такого мобільного додатка становить 6-8 осіб на рік. Для розрахунків приймемо 7 клієнтів. Разом з тим, проведений аналіз показав, що кількість таких клієнтів буде зростати, тобто можна очікувати зростання попиту на нашу розробку принаймні протягом 3-х років після її впровадження.

Тобто, якщо наша розробка буде впроваджена з 1 січня 2024 року (оскільки потребує що доопрацювання), то її результати будуть виявлятися протягом 2024-го, 2025-го та 2026-го років.

Прогноз зростання попиту на нашу розробку складає по роках:

а) 2024 р. – приблизно +2 шт. до базового року;

б) 2025 р. – +3 шт. до базового року;

в) 2026 р. – +4 шт. до базового року.

Можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації, тобто виведення нашої розробки на ринок, становитиме:

$$\Delta П_i = \sum_{1}^{n} (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}), \qquad (5.10)$$

де $\Delta Ц_o$ – покращення основного якісного показника від впровадження результатів нашої розробки у цьому році. Для нашого випадку це є збільшення ціни реалізації нашої розробки $\Delta Ц_o$ = 1200 - 1000 = + 200 тисяч грн;

N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 7 шт.;

ΔN – покращення основного кількісного показника від впровадження результатів розробки. Таке покращення становитиме по роках, відповідно: у 2024 році – + 2 шт., у 2025 році +3 шт., та у 2026 році + 4 шт. (відносно базового 2022 року);

$Ц_o$ – основний якісний показник (тобто ціна), який визначає обсяг діяльності у році після впровадження результатів розробки, грн; $Ц_o$= 1200 тисяч грн;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для нашого випадку n = 3;

λ – коефіцієнт, який враховує сплату податку на додану вартість; $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати ρ = (0,2...0,5); візьмемо ρ = 0,5;

υ – ставка податку на прибуток. У 2022-23 роках υ = 18%. У 2024 році також очікуємо на 18%.

Тоді можливе зростання чистого прибутку Δ П$_1$ для потенційного інвестора протягом першого року від можливого впровадження нашої розробки (2024 р.) складе:

$$\Delta П_1 = [200 \cdot 7 + 1200 \cdot 2] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 1298$$ тис. грн.

Можливе зростання чистого прибутку Δ П$_2$ для потенційного інвестора від можливого впровадження нашої розробки протягом другого (2025 р.) року складе:

$$\Delta П_2 = [200 \cdot 7 + 1200 \cdot 3] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 1708$$ тис. грн.

Можливе зростання чистого прибутку Δ П$_3$ для потенційного інвестора від можливого впровадження нашої розробки протягом третього (2026 р.) року складе:

$$\Delta П_3 = [200 \cdot 7 + 1200 \cdot 4] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) \approx 2182$$ тис. грн.

Приведена вартість зростання всіх чистих прибутків від можливого впровадження нашої розробки становитиме:

$$ПП = \sum_{1}^{\tau} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої роботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Для нашого випадку т = 3 роки;

τ – ставка дисконтування. Приймемо τ = 0,10 (10%);

t – період часу від моменту початку розроблення мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту до моменту отримання можливих чистих прибутків потенційним інвестором.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації нашої розробки, складе:

$$ПП = \frac{1298}{(1+0,1)^2} + \frac{1708}{(1+0,1)^3} + \frac{2182}{(1+0,1)^4} \approx 1073 + 1283 + 1490 = 3846 \text{ тисяч грн.}$$

Теперішня вартість інвестицій PV, що повинні бути вкладені для реалізації нашої розробки: $PV = (1,0…5) \times В_{заг}$.

Для нашого випадку $PV = (1,0…5) \times 94 = 5 \times 94 = 470$ тисяч грн.

Розраховуємо абсолютний ефект від можливих вкладених інвестицій $Е_{абс}$.

$$Е_{абс} = ПП – PV, \qquad (5.12)$$

де ПП – приведена вартість збільшення всіх чистих прибутків для інвестора від можливого впровадження нашої розробки, грн;

PV – теперішня вартість інвестицій PV = 470 тисяч грн.

Абсолютний ефект від можливого впровадження нашої розробки (при прогнозованому ринку збуту) за три роки складе:

$$Е_{абс} = 3846 – 470 = 3376 \text{ тисяч грн.}$$

Оскільки $Е_{абс} > 0$, то комерціалізація нашої розробки може бути доцільною.

Далі розрахуємо внутрішню дохідність $Е_в$ вкладених інвестицій:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1,$$

(5.13)

де $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс}$ = 3376 тис. грн;

PV –теперішня вартість початкових інвестицій PV = 470 тис. грн;

$T_ж$ – життєвий цикл розробки, роки.

$T_ж$ = 4 років (2023-й, 2024-й, 2025-й, 2026-й роки)

Для нашого випадку отримаємо:

$$E_в = \sqrt[4]{1 + \frac{3376}{470}} - 1 = \sqrt[4]{1 + 7,1829} - 1 = \sqrt[4]{8,1829} - 1 = 1,69 - 1 = 0,69 = 69\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією нашої розробки.

Мінімальна дохідність або мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau_{мін} = d + f, \quad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d = (0,10...0,12);

f – показник, що характеризує ризикованість вкладень для будівництва;

f = (0,1...0,50). Приймемо f = 0,40.

Для нашого випадку отримаємо:

$$\tau_{мін} = 0,12 + 0,40 = 0,52 \text{ або } \tau_{мін} = 52\%.$$

Оскільки величина $E_в$ = 69% > $\tau_{мін}$ = 52%, то потенційний інвестор у принципі може бути зацікавлений у фінансуванні та комерціалізації нашої розробки.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту.

Термін окупності $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}. \qquad (5.15)$$

Для нашого випадку термін окупності $T_{ок}$ коштів становитиме:

$$T_{ок} = \frac{1}{0,69} = 1,45$$ років < 3 років,

що свідчить про потенційну доцільність комерціалізації розробленого мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту.

Далі проведено моделювання залежності величини внутрішньої дохідності вкладених потенційних інвестицій від рівня інфляції в країні. Як відомо, на наступні роки, на жаль, прогнозується високий рівень інфляції (в межах 30% і вище), що обумовлюється військовою агресію росії проти України.

Прийнявши рівень інфляції у 30% отримаємо:

$$ПП = \frac{1298}{(1+0,3)^2} + \frac{1708}{(1+0,3)^3} + \frac{2182}{(1+0,3^4} \approx 768 + 777 + 764 = 2309 \text{ тисяч грн.}$$

Тоді абсолютний ефект від можливого впровадження нашої розробки за три роки складе:

$$E_{абс} = 2309 – 470 = 1839 \text{ тисяч грн.}$$

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1,$$

де $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс}$ = 1839 тисяч грн;

PV –теперішня вартість початкових інвестицій PV = 470 тисяч грн.

Для нашого випадку отримаємо:

$$E_в = \sqrt[4]{1+\frac{1839}{470}} - 1 = \sqrt[4]{1+3,9127} - 1 = \sqrt[4]{4,9127} - 1 = 1,488 - 1 = 0,488 = 48,8\%.$$

Зроблені розрахунки у вигляді графіків наведено на рис. 5.1.

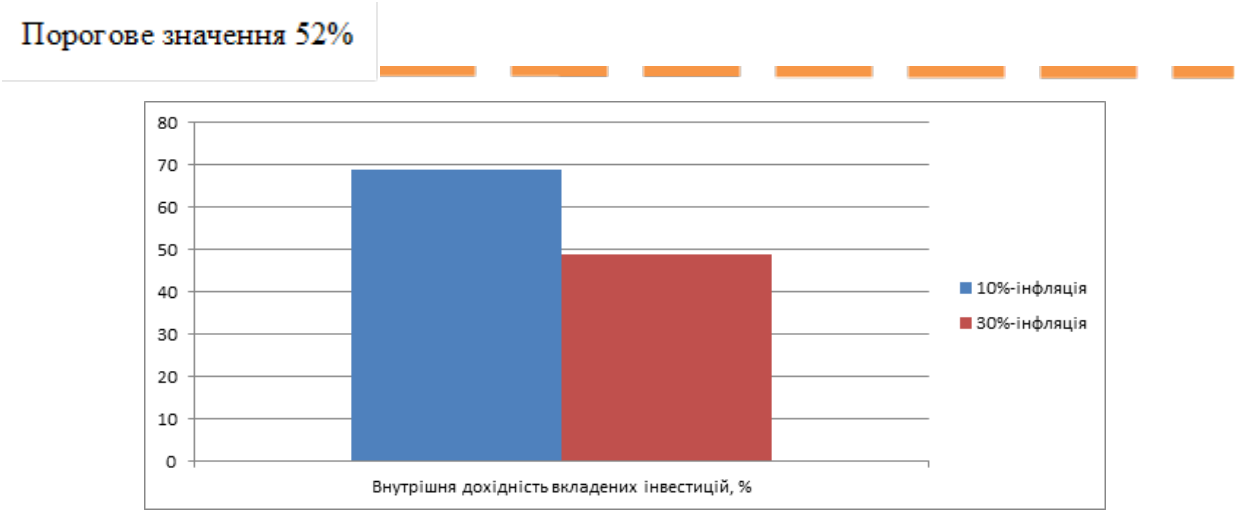


Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій від рівня інфляції в країні.

Аналіз графіка на рис 5.1 показує, що при рівні інфляції в 10% величина внутрішньої дохідності інвестицій становить $E_в$ = 69%, що значно більше порогового значення τмін = 52% (для будівництва) і тому комерціалізація нашої розробки може бути доцільною.

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю:

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1. Витрати на розробку	Не більше 100 тис. грн	94 тис. грн.	Досягнуто

2. Абсолютний ефект від впровадження розробки (в майбутній вартості грошей), тисяч грн	Не менше 3000 тисяч грн (за три роки)	3376 тисяч грн	Виконано
3. Внутрішня дохідність інвестицій, %	не менше 52%	69%	Досягнуто
4. Термін окупності інвестицій, роки	до 3-ти років	1,45 років	Виконано

Таким чином, основні техніко-економічні показники розробленого нами мобільного додатка для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту, визначені у технічному завданні, виконані.

ВИСНОВКИ

При розробці даної роботи було проаналізовано основні аспекти діяльності та процеси на будівництві.

Проведений аналіз сучасного стану предметної області. Визначено основні проблеми, а також шляхи їх вирішення, а саме впровадження мобільного додатку на основі будівельних технологічних стандартів, моделей процесів, норм і нормативів.

Було автоматизовано та покращено такі процеси на будівництві, а саме:

- Планування робіт.
- Швидкість збирання та подання даних забудовнику і замовнику.
- Збереження коштів від надлишкових витрат.
- Зменшення ризиків виходу за межі графіку робіт.

За рахунок автоматизації контролю процесів на будівельному майданчику, покращення якості робіт, пришвидшення процесу зі збору даних та їх аналізу на основі зроблених фотографій.

Було проаналізовано і порівняно між собою аналоги додатку, що розробляється.

Був проведений аналіз різних підходів до реалізації рішень поставлених задач. В результаті аналізу було визначено, що нативний додаток є більш оптимальним рішенням.

Проаналізовані сучасні підходи до розробки мобільних додатків, а також технології, що використовуються в сучасній розробці мобільних застосунків.

Було проаналізовано та досліджено сучасні архітектурні підходходи для створення мобільних додатків. В результаті дослідження було визначено, що оптимальним рішенням є шаблон MVVM.

Досліджено підходи та засоби роботи додатку з мережею, а саме REST API та GraphQL. В результаті чого було обрано оптимальним рішенням REST API з використанням зовнішніх сервісів Alamofire та AlamofireNetworkLogger.

Проаналізовано та досліджено сучасні підходи до збереження інформації в додатку локально, а саме фреймворки та бази даних CoreData, SQL, Realm. В результаті чого було обрано CoreData та спроектовано таблицю бази даних.

Для рішення задач розробки користувальницького інтерфейсу досліджувались нативні фреймворки UIKit та SWIFTUI. Виходячи з отриманих результатів в процесі аналізу та дослідження, було обрано SWIFTUI для розробки дизайну.

Також для вирішення задач з обробки великої кількості даних було проаналізовано фреймворки CoreML та TS для інтегрування штучного інтелекту з використанням нейронної мережі. В результаті проведених порівняльних тестів, аналізу та досліджень, було обрано CoreML, як шлях вирішення задачі.

Після розробки було проведено тестування всіх можливостей додатку, стабільності роботи, швидкодії та навантаження програми на пам’ять і CPU.

Розроблений застосунок відповідає сучасним тенденціям програмного забезпечення для будівельного бізнесу і сприяє вирішенню таких проблем як: збільшення лімітів часу на будівництво, зайві затрати на матеріали, якість робіт та надходження інформації про стан робіт та об’єктів в паперовому вигляді із затримкою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Сердюк Г.В. Мобільний додаток для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту / Г.В. Середюк., Є.А. Паламарчук // XV міжнародна науково-практична конференція “ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА АВТОМАТИЗАЦІЯ – 2022”(жовтень 2022) Одеського національного технічного університету, - Одеса: ОНТУ, 20-21 жовтня 2022 р.
2) AppMagic – [Електронний ресурс] – Режим доступу: https://appmagic.rocks/top-charts/apps
3) iOS App Development [Електронний ресурс] – Режим доступу: https://aws.amazon.com/mobile/mobile-apps.
4) Constriction Marketing Association – [Електронний ресурс] – Режим доступу: https://blog.constructionmarketingassociation.org/benefits-of-construction-mobile-apps/
5) mindInventory – [Електронний ресурс] – Режим доступу: https://www.mindinventory.com/blog/how-mobile-apps-benefit-construction-businesses/
6) ProCrew Schedule – [Електронний ресурс] – Режим доступу: https://procrewschedule.com/tips-and-benefits-of-using-mobile-apps-in-the-construction-site/
7) Розробка програмного забезпечення – [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Розробка_програмного_забезпечення
8) Flowfinity – [Електронний ресурс] – Режим доступу: https://www.flowfinity.com/blog/7-ways-mobile-apps-are-transforming-the-construction-industry.aspx
9) Sentry – [Електронний ресурс] – Режим доступу: https://sentry.io/for/mobile/?utm_source=google&utm_medium/
10) Accelerance – [Електронний ресурс] – Режим Доступу: https://www.accelerance.com/blog/why-is-mobile-app-development-so-important-today

11) SmartInnovations – [Електронний ресурс] – Режим доступу: https://www.smartsight.in/technology/why-mobile-applications-are-important-for-businesses/
12) techahead – [Електронний ресурс] – Режим доступу: https://www.techaheadcorp.com/blog/importance-of-a-mobile-app-for-the-construction-business-and-its-productivity/
13) Mission20Zero – [Електронний ресурс] – Режим доступу: https://m20zero.com/how-mobile-apps-can-help-optimize-your-business-prospects/
14) Postindustria – [Електронний ресурс] – Режим доступу: https://postindustria.com/apple-core-ml-leveraging-the-power-of-machine-learning-for-mobile/
15) Mission20Zero – [Електронний ресурс] – Режим доступу: https://m20zero.com/how-mobile-apps-can-help-optimize-your-business-prospects/
16) Infostretch – [Електронний ресурс] – Режим доступу: https://www.infostretch.com/blog/a-primer-to-mobile-operating-systems/
17) Clarion Technologies – [Електронний ресурс] – Режим доступу: https://www.clariontech.com/blog/top-mobile-app-development- frameworks-in-2019
18) Офіційний сайт Google Play Market – [Електронний ресурс] – Режим доступу: https://play.google.com/store/apps/details?id=com.fivesysdev.mathy&hl= en_US&gl=US
19) Офіційний сайт Apple App Store – [Електронний ресурс] – Режим доступу: https://www.apple.com/ua/app-store/
20) Jeff McWherter. Professional Mobile Application Development / Scott Gowell, Jeff McWherter, 2012р. – 432 с.
21) Chris Adamson. iOS 10 SDK Development: Creating iPhone and iPad Apps with Swift / Chris Adamson, Janie Clayton, 2017р – 264 с.
22) Jagannath S., Debasish D., Lov K., Aneesh K. Application Dev // 18th ICDCIT 2022 p – 165c.
23) Tamai S. Technological and business Fundamentals for mobile app dev // Springer Nature Switzerland AG 2022 p – 181 c.

24) InfoSystem – [Електронний ресурс] – Режим доступу: https://www.hyperlinkinfosystem.com/blog/the-relevance-of-mobile-app-development-to-the-modern-world

25) Goce Armenski, Marjan Gusev. Architecture of Modern e-learning Systems // The 6th International Conference for Informatics and Information Technology, 2008, P. 38-42. – Режим доступу: http://ciit.finki.ukim.mk/data/papers/6CiiT/6CiiT-09.pdf

26) Wikipedia Swift programming. Режим доступу: Wiki swift - Дата доступу: 20.05.2021

27) Blinco K., Mason J., McLean N., Wilson S. (2004), Trends and issues in e-learning infrastructure development, A White Paper for alt-i-lab 2004 Prepared on behalf of DEST (Australia) and JISC-CETIS (UK).

28) Yiyi S., Practical Application Development // Apress / 2019 – 293 с.

29) Robert C. Martin. Clean Architecture // Prentice Hall / 2017 р. – 432 с.

30) Teacher Training and Professional Development: Concepts, Methodologies, Tools and Aplication / Management Association, Information Resources, P. 915.

31) Баженов В.А., Лізунов Інформатика. Комп"ютерна техніка. Комп"ютерні технології: підручник для студ. вищ. навч. закл. / В.А. Баженов та ін.; [наук. ред. : Г.А. Шинкаренко, О.В. Шишов] ; ЛНУ ім. І. Франка ; Київський нац. унт будівництва і архітектури ; Нац. техн. ун-т України "Київський політ. ін-т" - Київ: Каравела, 2011. - 592 с.

32) Griffiths D. Head First Android Development: A Brain-Friendly Guide 1st Edition / D. Griffiths. — O'Reilly Media, 2015 — 734 p.

33) Murphy M. The Busy Coder's Guide to Advanced Android Development M. Murphy. — CommonsWare, LLC, 2011 — 630 p.

34) Darwin F. Android Cookbook: Problems and Solutions for Android Developers 2nd Edition / F. Darwin. — O'Reilly Media, 2017 — 772 p. Schildt H. Java: A Beginner's Guide, Eighth Edition, 8th Edition / H.Schildt. — McGraw-Hill Education, 2018 — 720 p.

35) Quickly Creating, Designing and Utilizing Mobile Apps for Your Business, 2nd Edition / J.McCalister. — CreateSpace Independent Publishing Platform, 2014 — 170 p.

36) Lee V. Mobile Applications: Architecture, Design, and Development / V.Lee. — Prentice Hall, 2004 — 368 p.

37) Iversen J. Learning Mobile App Development: A Hands-on Guide to Building Apps with iOS and Android, 1st Edition / J.Iversen, M.Eierman. — Addison-Wesley Professional, 2013 — 464 p.

38) Vasic M. Mastering Android Development with Kotlin: Deep dive McCalister J. Mobile Apps Made Simple: The Ultimate Guide to into the world of Android to create robust applications with Kotlin / M.Vasic. — Packt Publishing, 2017 — 378 p.

39) Leiva A. Kotlin for Android Developers: Learn Kotlin the easy way while developing an Android App, 1st Edition / A.Leiva. — CreateSpace Independent Publishing Platform, 2016 — 240 p.

40) KamathU.Deep Learning for NLP and Speech Recognition/ U.Kamath, W.Whitaker. — Springer, 2018 — 621 p.

41) Grenadiuk A. Mobile App Marketing And Monetization / A.Grenadiuk.— Semantic Valley LLC, 2014 — 151 p.

42) Berney P. Mobile Marketing: Lessons from Global Brand Leaders on How to Make a Success of the Mobile Channel / P.Berney. — Kogan Page, 2019 — 224 p.

43) Hoss O. App Store Optimization: A Step-by-Step Guide to Boosting your App's Organic Downloads / O.Hoss. — CreateSpace Independent Publishing Platform, 2019 — 309 p.

44) Greene R. App Marketing: Top Mobile App Monetization and Promotion Strategies / R.Greene. — Tru Nobilis Pubilishing, 2017 —104p.

45) Google Play: number of available apps 2009-2019[Електронний ресурс]: [Website] Точка доступу : https://www.statista.com/statistics/266210/number-of-available- applications-inthe-google-play-store.

46) George K. The smart grid development platform // Arthec house / Page, 2017 – 217 с.

47) Mats G., Jonathon M. / Practical Artificial Intelligence with Swift // Secret Lab? 2020 – 565 с.

48) Elaine C, Darren J. A Developer's Guide to Building AI Applications // O'Reilly Media, Incorporated, 2020 / 2020 – 50 с.

49) Matt R. Arun P., Mobile Artificial Intelligence Projects: Develop seven projects on your smartphone using artificial intelligence and deep learning techniques // Packt Publishing Ltd / 2019 – 312 c.

50) Laurence M. AI and Machine Learning for On-Device Development // O'Reilly Media, Inc / 2021 – 328 c.

ДОДАТКИ

Додаток А
(обов’язковий)
ВНТУ

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., професор
О.В. Бісікало
“____”_____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

“Мобільний додаток для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту**”**

08-02.МКР.000.10.000 ТЗ

Керівник роботи:
к.т.н., проф. каф. АІІТ Паламарчук Є.А

. “___” _____ 2022 р.

Виконавець:

ст. гр. 1АКІТ-21м Середюк Г.В.
“___” _____ 2022 р.

Вінниця 2022

1. Назва та галузь застосування

Мобільний додаток для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту. Розроблена система змогу покращити ефективність робіт на будівництві майже в усіх його аспектах, підвищити рентабельність та спростити процес контролю за будівництвом.

2. Підстава для розробки

Попри стрімкий розвиток будівельної галузі та технологій, які використовуються на будівництвах, все ще актуальні такі проблеми, як дефіцит кваліфікованої робочої сили, висока конкуренція, низька рентабельність, якість робіт та збільшений ризик перевищення термінів будування. Раніше наведені проблеми, відтворюються в таких аспектах та ситуаціях, як перевищення лімітів всіх термінів та бюджету, дані із будівництва замовник отримує із запізненням у вигляді великої кількості звітів у паперовому вигляді, якість роботи перевіряється, тільки якщо є змога потрапити на будівельний майданчик, після закінчення будівництва, замовник може витратити надлишкові кошти на матеріали, зарплату для працівників.

Саме тому виникає необхідність розробки апаратно-програмного забезпечення для автоматизації процесів на будівельних майданчиках та швидкого доступу до актуальної інформації об’єктів на базі отриманих даних. Підставою для розробки є наказ ВНТУ №__ від _____

3. Мета та призначення розробки

Головною метою є побудова ідеології, алгоритму та програмно-апаратної реалізації системи для швидкого збирання, зберігання інформації про параметри стану будівельного об’єкту, обробки даних з використанням штучного інтелекту та надсилання цих даних до серверної частини для покращення процесів різного виду робіт на будівельному майданчику.

4. Технічні дані

4.1. Вид програмного забезпечення – мобільний додаток;
4.2. Підтримка ОС – iOS;
4.3. Підтримка версії ОС – 16.0+;
4.4. Розмір додатку – 48 MB;
4.5. Штучний інтелект – нейронна мережа(CoreML);
4.6. Мова програмування - Swift,
4.7. Фреймворк для локального сховища - CoreData;
4.8. Мови графічного інтерфейсу – англійська, іврит, корейська, японська.

5 Джерела розробки

5.1. Lee V. Mobile Applications: Architecture, Design, and Development / V.Lee. — Prentice Hall, 2004 — 368 p.
5.2. Iversen J. Learning Mobile App Development: A Hands-on Guide to Building Apps with iOS and Android, 1st Edition / J.Iversen, M.Eierman. — Addison-Wesley Professional, 2013 — 464 p.
5.3. Mats G., Jonathon M. / Practical Artificial Intelligence with Swift // Secret Lab 2020 – 565 с.
5.4. Robert C. Martin. Clean Architecture // Prentice Hall / 2017 p. – 432 с.
5.5. Iversen J. Learning Mobile App Development: A Hands-on Guide to Building Apps with iOS and Android, 1st Edition / J.Iversen, M.Eierman. — Addison-Wesley Professional, 2013 — 464 p.

6. Стадії та етами розробки

6.1 Розділ 1 “ Дослідження предметної області ” має бути виконаний до 3.10.2022.
6.2 Розділ 2 “ Обґрунтування методів реалізації ” має бути виконаний до 21.10.2022.
6.3 Розділ 3 “ Розробка програмного забезпечення додатку ” має бути виконаний до 12.11.2022.
6.4 Розділ 4 “ Тестування розробленого програмного забезпечення ” має бути виконаний до 27.11.2022.

6.5 Розділ 5 “ Економічний розділ ” має бути виконаний до 04.12.2022.
6.5 Оформлення пояснювальної записки має бути виконаний до 10.12.2022.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «28»__11_ 2022 р.
7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «16»__12_ 2022 р.
7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «23»__12_ 2022р.

Додаток Г (обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МОБІЛЬНИЙ ДОДАТОК ДЛЯ РОБОТИ З АРХІТЕКТУРНИМИ ПЛАНАМИ БУДІВЕЛЬ І ОБРОБКОЮ ДАНИХ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Студент групи 1АКІТ-21м

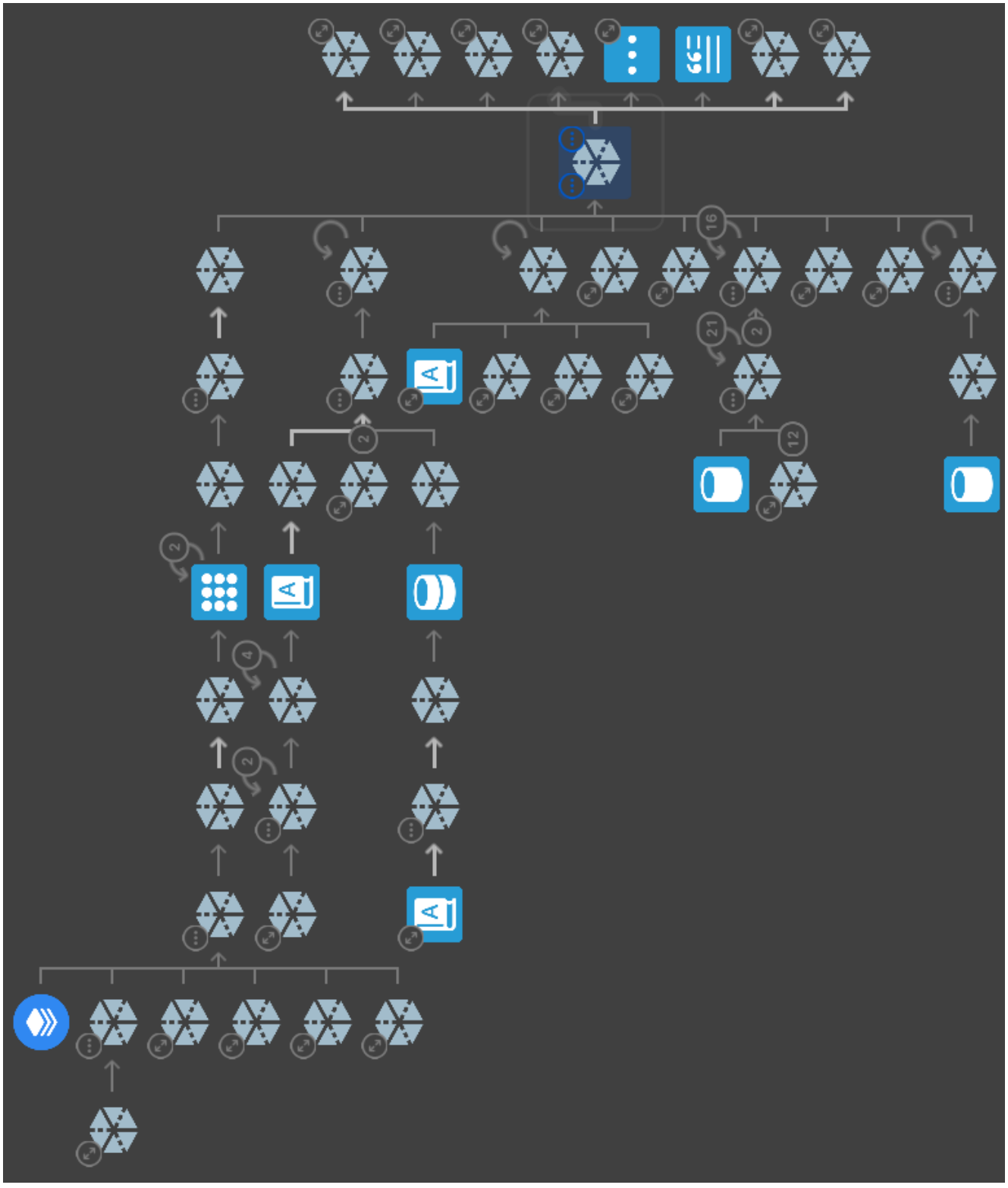
Гліб СЕРЕДЮК
Підпис. *Ім’я ПРІЗВИЩЕ*

Керівник к.т.н., професор, кафедри АІІТ

Євген ПАЛАМАРЧУК
Підпис. *Ім’я ПРІЗВИЩЕ*

Опонент к.т.н., доцент, кафедри КН

Олег Колесницький
Підпис. *Ім’я ПРІЗВИЩЕ*



Граф сутностей та об’єктів при взаємодії з планом.

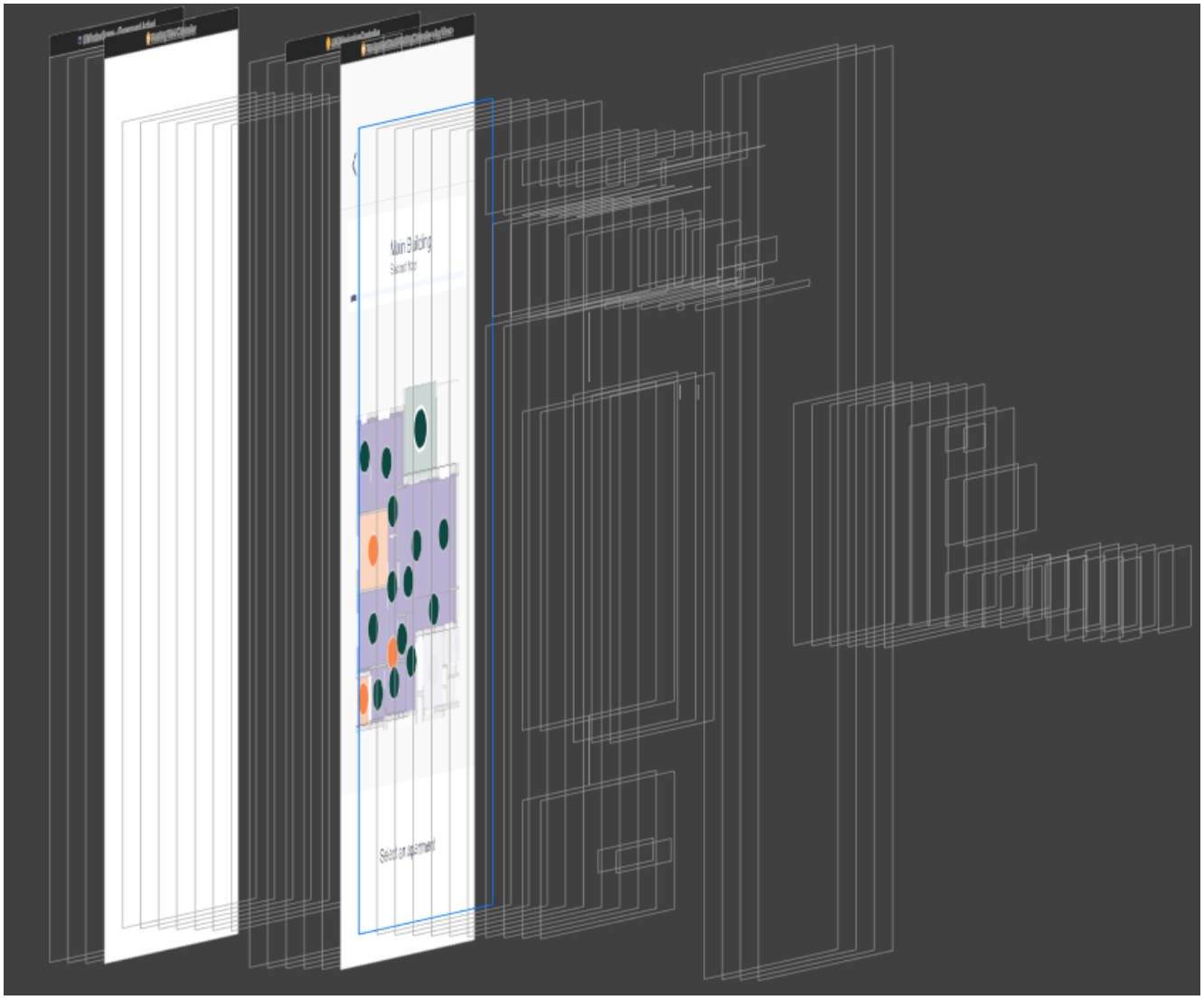
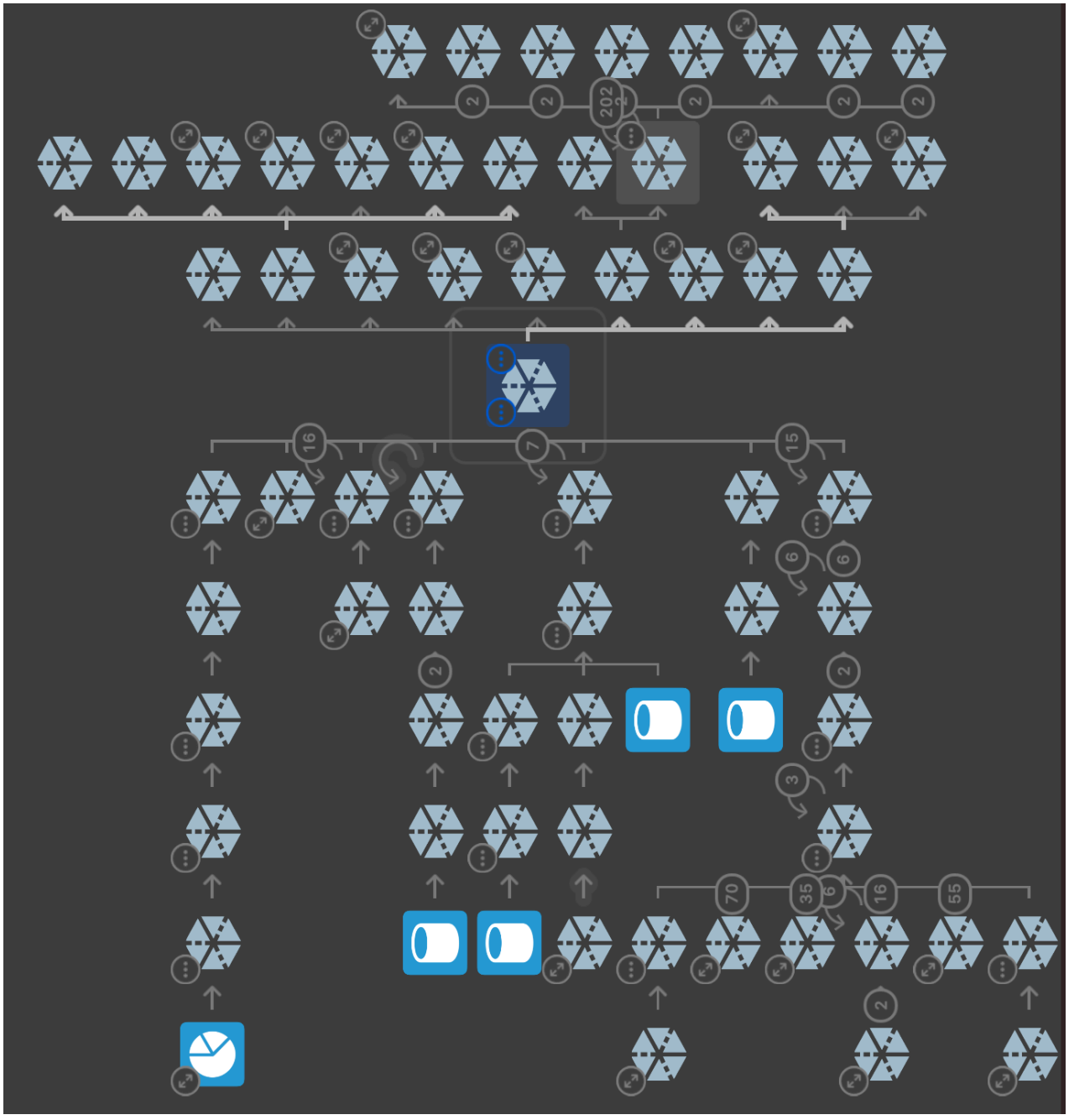


Схема графічних елементів при відображенні плану



Граф об’єктів та їх посилань в пам’яті під час операцій моделі нейронної мережі.

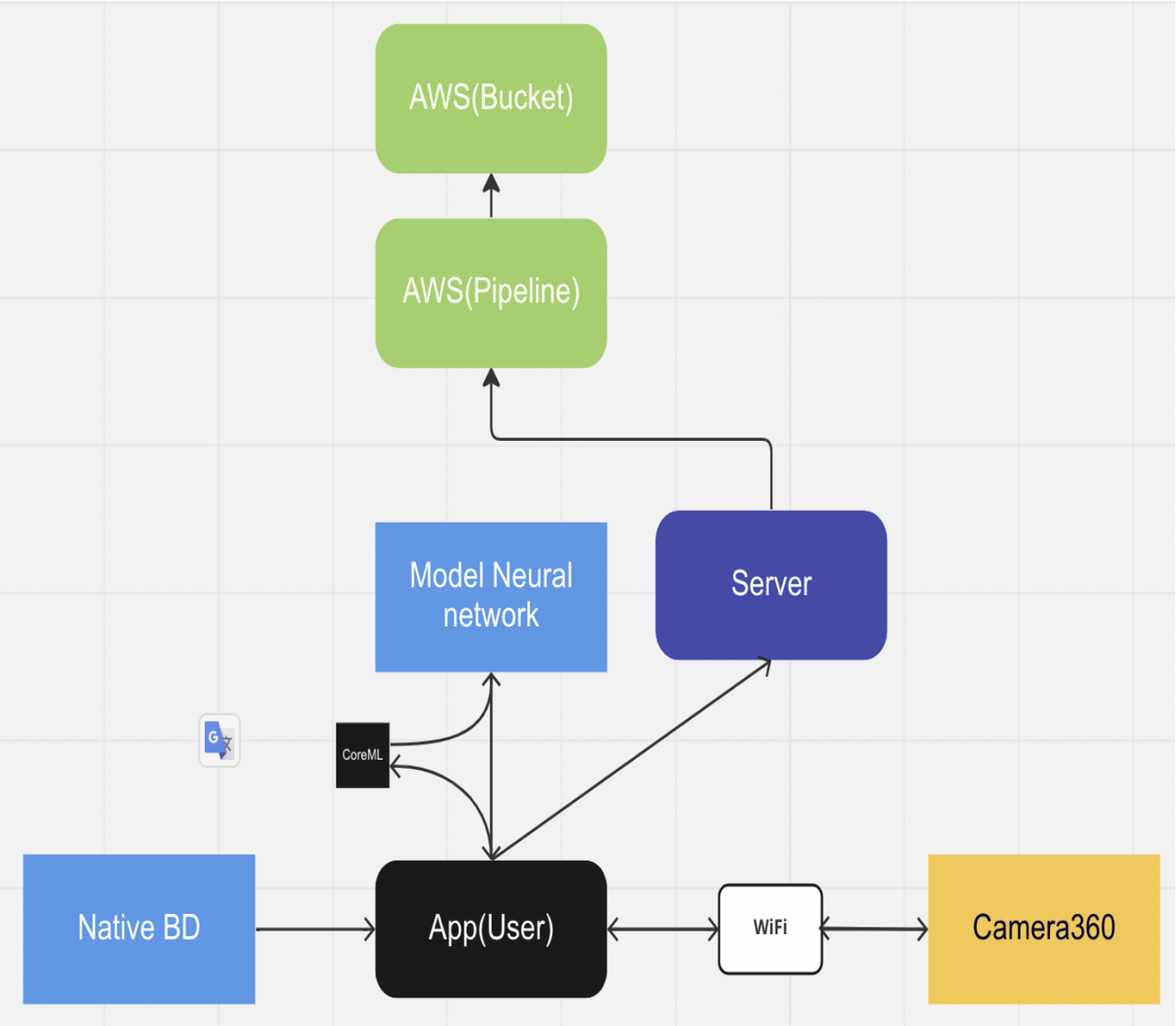


Схема взаємодії додатку з камерою 360 град.

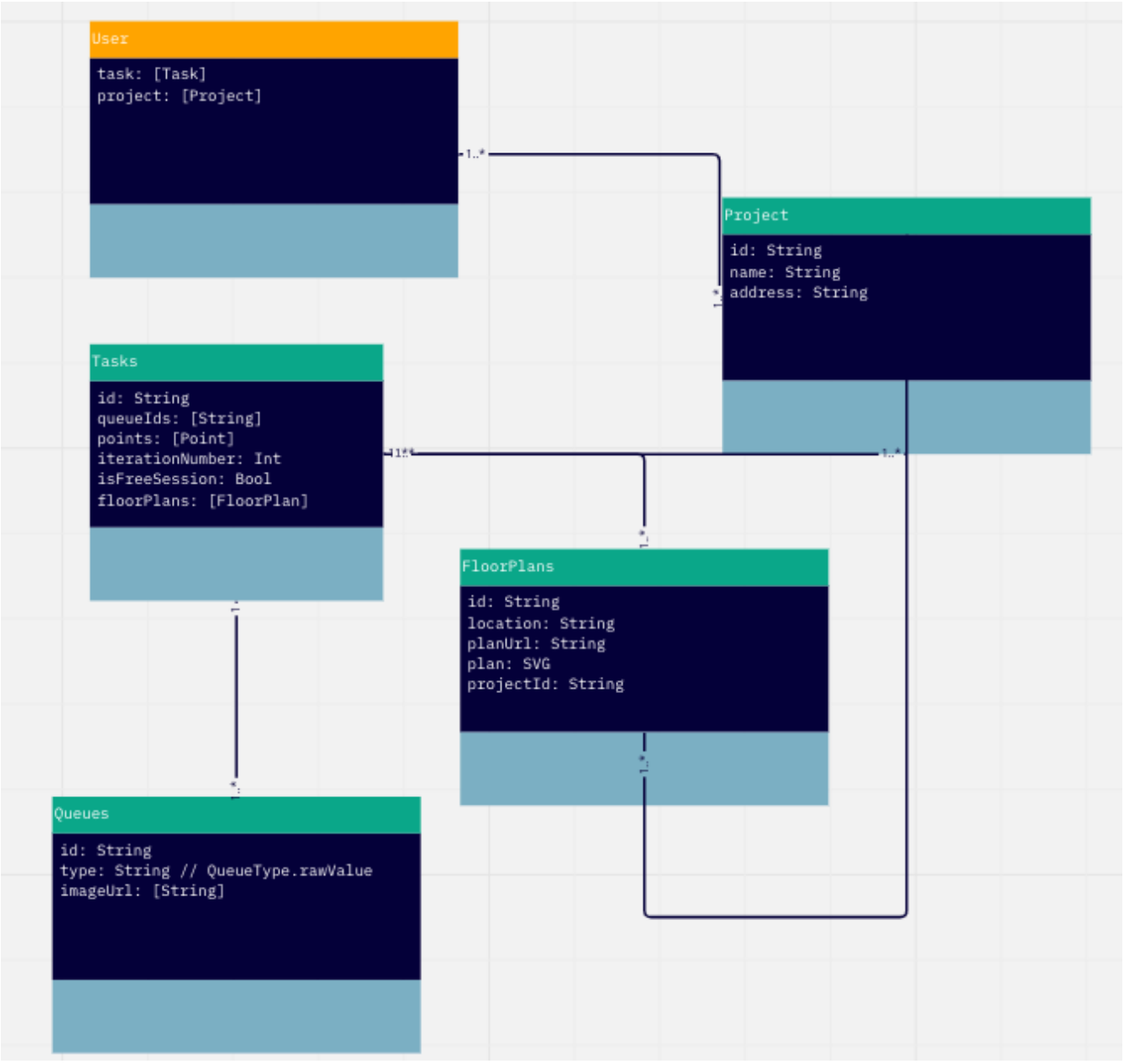
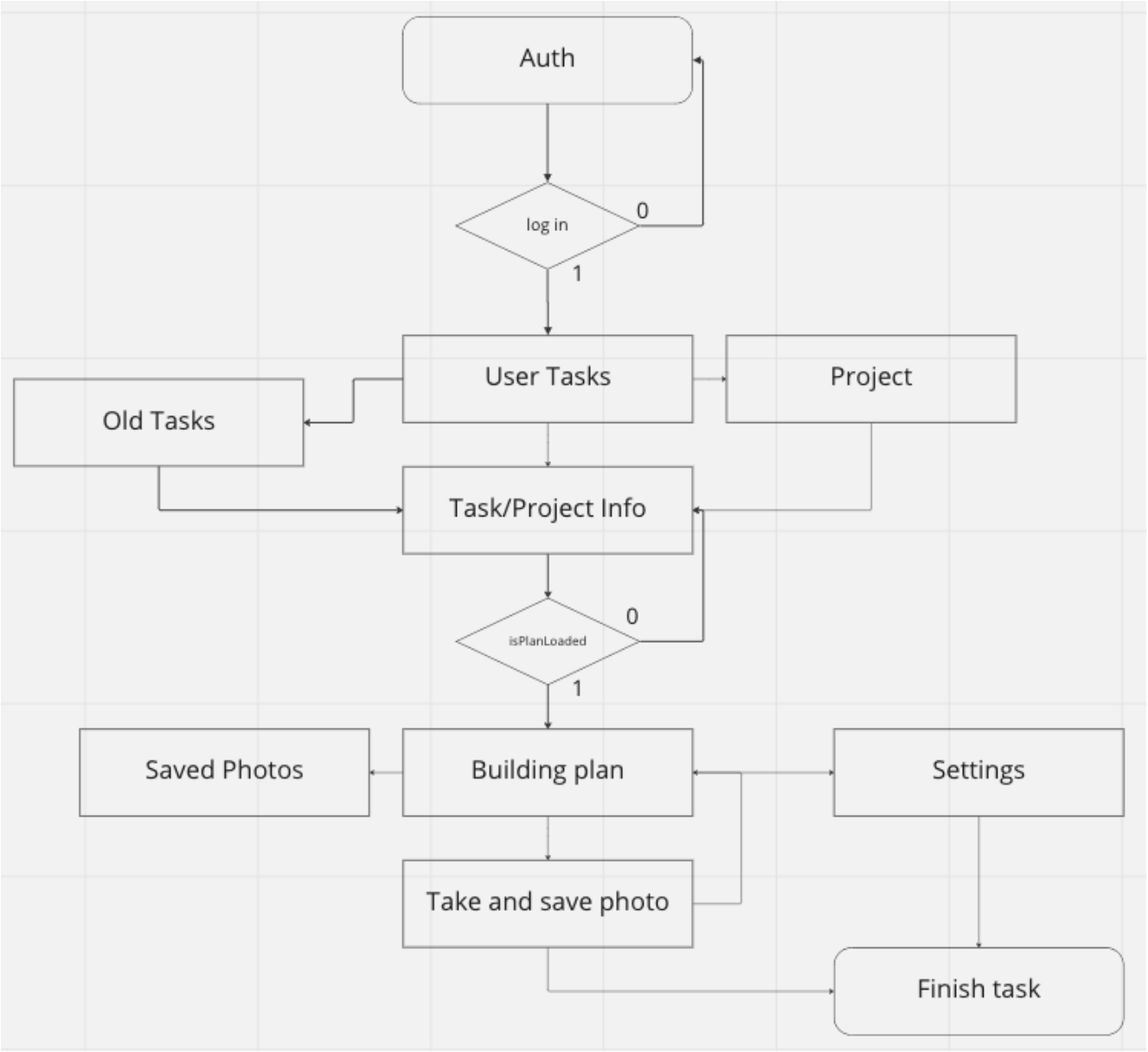


Схема таблиць локальної бази даних.



Блок схема навігації проекту.

Додаток В Лістинг програми

```
protocol CameraManager {
   /// Connects to the camera and sends the results.
   func connect() -> AnyPublisher<CameraState, Error>

   /// Gets the state of the picture taking process.
   func fetchPhotoState() -> AnyPublisher<PhotoState, Error>

   /// Takes a picture and sends the image taken.
   func takePhoto() -> AnyPublisher<UIImage?, Error>

   func getInformation() -> Future<CameraState, Error>

   }


final class NetworkClient: NetworkProvider {

   private let keychainStorage: KeychainStorage
   private var session: Session
   private let jwtHandler: JWTHandler

   init(keychainStorage: KeychainStorage, jwtHandler: JWTHandler) {
      self.jwtHandler = jwtHandler
      self.keychainStorage = keychainStorage
      self.session = Session.default
   }

   func request(_ info: RequestInfoConvertible) -> AnyPublisher<Data, Error> {
      let requestInfo = info.asRequestInfo()

      var headers: HTTPHeaders = []

      if let headersInfo = requestInfo.headers {
         headers = headersInfo
      }

      if requestInfo.shouldContainAuthToken {
         headers.add(.authorization(keychainStorage.fetch(key: .idToken) ?? ""))
      }

      return session.request(requestInfo.url,
                             method: requestInfo.method,
                             parameters: requestInfo.parameters,
                             encoding: requestInfo.encoding,
                             headers: headers,
                             interceptor: AuthInterceptor(
                              jwtHandler: jwtHandler,
                              shouldContainAuthToken: requestInfo.shouldContainAuthToken,
```

```
                        keychainStorage: keychainStorage
                       ),
                       requestModifier: requestInfo.requestModifier)
        .publishData().tryMap { [weak self] response -> Data in
            let statusCode = "[StatusCode:\(response.response?.statusCode.formatted() ?? "None")]"
            let requestHeaders = requestInfo.headers?.description ?? "[None]"

            switch response.result {
                case .success(let data):

                    return data
                case .failure(let error):
                    throw error
            }
        }
        .eraseToAnyPublisher()
    }
}



import CoreML


/// Model Prediction Input Type
@available(macOS 10.15, iOS 13.0, tvOS 13.0, watchOS 6.0, *)
class PhotoBuilderModelInput : MLFeatureProvider {

    /// input as color (kCVPixelFormatType_32BGRA) image buffer, 512 pixels wide by 256 pixels
high
    var input: CVPixelBuffer

    var featureNames: Set<String> {
        get {
            return ["input"]
        }
    }

    func featureValue(for featureName: String) -> MLFeatureValue? {
        if (featureName == "input") {
            return MLFeatureValue(pixelBuffer: input)
        }
        return nil
    }

    init(input: CVPixelBuffer) {
        self.input = input
    }

    convenience init(inputWith input: CGImage) throws {
        self.init(input: try MLFeatureValue(cgImage: input, pixelsWide: 512, pixelsHigh: 256,
pixelFormatType: kCVPixelFormatType_32ARGB, options: nil).imageBufferValue!)
    }

    convenience init(inputAt input: URL) throws {
```

```
        self.init(input: try MLFeatureValue(imageAt: input, pixelsWide: 512, pixelsHigh: 256,
pixelFormatType: kCVPixelFormatType_32ARGB, options: nil).imageBufferValue!)
    }

    func setInput(with input: CGImage) throws  {
        self.input = try MLFeatureValue(cgImage: input, pixelsWide: 512, pixelsHigh: 256,
pixelFormatType: kCVPixelFormatType_32ARGB, options: nil).imageBufferValue!
    }

    func setInput(with input: URL) throws  {
        self.input = try MLFeatureValue(imageAt: input, pixelsWide: 512, pixelsHigh: 256,
pixelFormatType: kCVPixelFormatType_32ARGB, options: nil).imageBufferValue!
    }

}


protocol LocationManager {
    func getCurrentLocation() -> CLLocation?
}

struct LocationManagerImpl: LocationManager {

    let locationManager = CLLocationManager()

    func getCurrentLocation() -> CLLocation? {
        switch locationManager.authorizationStatus {
        case .authorizedAlways, .authorizedWhenInUse:
            return locationManager.location
        default: return nil
        }
    }

}


struct CoreDataStack {



    private let isStoreLoaded = CurrentValueSubject<Bool, Error>(false)

    private let bgQueue = DispatchQueue(label: "coredataQueue")



    let container: NSPersistentContainer

    lazy var viewContext: NSManagedObjectContext = {

        return container.viewContext

    }()
```

```
    init(directory: FileManager.SearchPathDirectory = .documentDirectory,
         domainMask: FileManager.SearchPathDomainMask = .userDomainMask,
         version vNumber: UInt) {
        let version = Version(vNumber)
        container = NSPersistentContainer(name: version.modelName)
        if let url = version.dbFileURL(directory, domainMask) {
            debugPrint("DB Container URL: \(url)")
            let store = NSPersistentStoreDescription(url: url)
            container.persistentStoreDescriptions = [store]
        }
        bgQueue.async { [weak isStoreLoaded, weak container] in
            container?.loadPersistentStores { _, error in
                DispatchQueue.main.async {
                    if let error = error {
                        isStoreLoaded?.send(completion: .failure(error))
                    } else {
                        container?.viewContext.configureAsReadOnlyContext()
                        isStoreLoaded?.value = true
                    }
                }
            }
        }
    }

}

// MARK: - Versioning

extension CoreDataStack.Version {
```

```
    static var actual: UInt { 1 }
}

extension CoreDataStack {

    struct Version {
        private let keychainStorage: KeychainStorage = KeychainStorageImpl()
        private let number: UInt

        init(_ number: UInt) {
            self.number = number
        }

        var modelName: String {
            return "PhotoBuilderApp"
        }

        func dbFileURL(_ directory: FileManager.SearchPathDirectory,
                       _ domainMask: FileManager.SearchPathDomainMask) -> URL? {
            let path = FileManager.default
                .urls(for: directory, in: domainMask).first?
                .appendingPathComponent(subPathToDB)
            return path
        }
            private var subPathToDB: String {
            return "db.sql.debug"
        }
```

Додаток Г (обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: "Мобільний додаток для роботи з архітектурними планами будівель і обробкою даних з використанням штучного інтелекту"

Тип роботи: Магістерська кваліфікаційна робота

Підрозділ: кафедра АІІТ, ФКСА, 1АКІТ-21м

Науковий керівник: Паламарчук Є.А., проф. каф. АІІТ

Показники звіту подібності

Unicheck	
Оригінальність	96.8%
Схожість	3.2%

Аналіз звіту подібності (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Автор _____ Середюк Г.В.

Опис прийнятого рішення: Допустити до захисту

Особа, відповідальна за перевірку _____ Маслій Р.В.