Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту»

Виконав: студент II курсу, групи 2ПІ-21м
спеціальності
121 – Інженерія програмного забезпечення

Прус Б.В.

(прізвище та ініціали)

Керівник: к.т.н., доц. Ракитянська Г.Б.

«14» 12 2022р.

Опонент: к.т.н., доц. Богомолов С.В.

«14» 12 2022р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«14» 12 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.
« 16 » вересня 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Прусу Богдану Вікторовичу

1. Тема роботи: Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту

Керівник роботи: Ракитянська Ганна Борисівна к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «15» вересня 2022 р. №205-А.

2. Строк подання студентом роботи 9 грудня 2022 р.

3. Вихідні дані до роботи: аналіз сучасних методів та програмних засобів для розробки мобільних додатків з використанням технологій розпізнавання зображень для агрегування медіа контенту, аналіз технологій створення мобільних додатків з інтерактивним інтерфейсом; базові поняття структури сторінок мобільного додатку.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз питання та постановка задач дослідження; розробка структури мобільного додатку та алгоритму агрегування медіа контенту; розробка програмного забезпечення; тестування роботи мобільного додатку; економічна частина; висновки; додатки.

5. Перелік графічного матеріалу: актуальність теми; мета, об'єкт та предмет дослідження; наукова новизна отриманих результатів; порівняльний аналіз аналогів; основні задачі розробки; створення елементів дизайну; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ракитянська Г.Б., к.т.н., доц. кафедри ПЗ	16.09.22	17.09.22
5	Глущенко Л.Д., к.е.н., доц. кафедри ЕПВМ	1.12.22	2.12.22

7. Дата видачі завдання 16 вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз, вибір та обґрунтування актуальності розробки	17.09.2022 – 24.09.2022	Вик.
2	Аналіз існуючих аналогів	25.09.2022- 30.09.2022	Вик.
3	Розробка структури мобільного додатку та алгоритму агрегування медіа контенту	01.10.2022- 08.10.2022	Вик.
4	Проектування бази даних	09.10.2022- 10.10.2022	Вик.
5	Розробка структури та інтерфейсу мобільного додатку	11.10.2022- 21.10.2022	Вик.
6	Розробка програмного забезпечення	22.10.2022- 22.11.2022	Вик.
7	Тестування програмного забезпечення	23.11.2022- 30.11.2022	Вик.
8	Економічна частина	01.12.2022- 08.12.2022	Вик.

Студент _____ **Прус Б.В.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Ракитянська Г.Б.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена розробці методів та засобів розпізнавання зображень для агрегування медіа контенту, відмінністю якого є використання технологій штучного інтелекту на мобільних пристроях для роботи методів агрегування медіа контенту з акцентом на приватність даних користувача.

В процесі виконання роботи проведено дослідження предметної області, здійснено порівняльний аналіз аналогів, що використовують методи та засоби розпізнавання зображень для агрегування медіа контенту. Враховуючи всі недоліки та переваги проаналізованих аналогів, визначено основні задачі розробки, розроблено діаграми варіантів використання системи, спроектовано логічну та фізичну структури мобільного додатку, розроблено алгоритм агрегування медіа контенту.

Проведено аналіз відомих засобів розробки мобільних додатків. Розроблено модуль агрегування медіа контенту. Виконано розробку для платформ Android та iOS.

Результати тестування мобільного додатку та модулів системи виявились успішними.

Для розробки мобільного додатку було обрано Flutter та мову програмування Dart, у якості середовища розробки використано Android Studio та Xcode.

ANNOTATION

The master's qualification work is devoted to the development of image recognition methods and tools for aggregating media content, the difference of which is the use of artificial intelligence technologies on mobile devices for the operation of methods of aggregating media content with an emphasis on the privacy of user data.

In the course of the work, a study of the subject area was carried out, a comparative analysis of analogues using image recognition methods and tools for aggregating media content was carried out. Taking into account all the shortcomings and advantages of the analyzed analogues, the main tasks of the development were defined, diagrams of options for the use of the system were developed, the logical and physical structure of the mobile application was designed, and the algorithm of media content aggregation was developed.

An analysis of well-known tools for developing mobile applications has been carried out. A media content aggregation module has been developed. Developed for Android and iOS platforms.

The results of testing the mobile application and system modules were successful.

Flutter and the Dart programming language were chosen for the development of the mobile application, and Android Studio and Xcode were used as the development environment.

.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Мобільні пристрої в житті сучасної людини займають особливе місце. Вони допомагають бути на зв'язку зі світом в будь-який час і активно використовуються в різних сферах нашого життя. Особливо з розвитком технологій мобільної фотографії вони почали замінювати собою необхідність використання професійних та аматорських камер, оскільки дають змогу робити якісні знімки без особливих умов та навиків.

На даний момент часу розробка мобільних додатків є одним із пріоритетних напрямків процесу інформатизації сучасного суспільства. Над зручністю і простотою створення мобільних додатків з інтерактивним інтерфейсом працює велика кількість працівників різних фахів. Мобільні додатки з кожним днем стають дедалі популярнішими, і часто замінюють собою використання WEB сайтів, а інколи мають унікальний або розширений функціонал у порівнянні з WEB додатками, оскільки розраховуються на більшу кількість активних користувачів.

У магістерській кваліфікаційній роботі розроблено програмне забезпечення у вигляді мобільного додатку, що використовує методи та засоби розпізнавання зображень для агрегування медіа контенту, основною задачею якого є сканування галереї мобільного пристрою та об'єднання знайдених медіа файлів у події за розробленим алгоритмом. Завдяки тому, що розпізнавання зображень виконується на мобільному пристрої, забезпечується повна приватність відео та фото матеріалів користувача, та дозволяє виконувати операції з розпізнавання об'єктів швидко.

Більшість звичайних людей використовують мобільні телефони як головний пристрій для здійснення відео та фото знімків, тому що він завжди під рукою та його можливостей в більшості достатньо для здійснення прийнятих фотографій [2]. Таким чином з часом в галереї пристрою накопичується досить велика кількість знімків та відео матеріалів. З часом, переглядаючи здійснені знімки, можуть виникнути проблеми з їх відокремленням до певних життєвих подій. Для цього необхідний програмний засіб, який допоможе здійснити агрегування фото та відеоматеріалів та віднести їх до певних життєвих подій, при цьому залишивши лише ті знімки, які мають певну цінність. Тому тематика даного дослідження щодо розробки методів і

програмних засобів розпізнавання зображень для агрегування медіа контенту є досить актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до координаційного плану науково-дослідницьких робіт Міністерства освіти і науки України з спеціальністю 121 – «Інженерія програмного забезпечення».

Об’єктом дослідження є процеси розробки методів та програмних засобів розпізнавання зображень для агрегування медіа контенту.

Предметом магістерської кваліфікаційної роботи є методи та засоби агрегування медіа контенту.

Метою магістерської кваліфікаційної роботи є зменшення ресурсів для реалізації алгоритму агрегування зображень у події за рахунок застосування технологій трансферного навчання.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

– ідентифікація проблеми агрегування медіа контенту;
– здійснення порівняльного аналізу аналогів;
– аналіз методів та засобів реалізації;
– розробка алгоритму агрегування медіа контенту;
– методи побудови UML-діаграм для відображення основних функцій системи;
– розробка програмного засобу;
– оцінка ефективності роботи програмного засобу.

В ході дослідження також були застосовані сучасні методи створення мобільних додатків, розробки мобільних додатків з інтерактивним інтерфейсом та баз даних.

Наукова новизна одержаних результатів:

– Отримав подальший розвиток метод розпізнавання зображень для агрегування медіа контенту, відмінністю якого є використання технологій штучного інтелекту безпосередньо на мобільному пристрої, за рахунок чого досягається приватність даних користувача.

– Отримала подальший розвиток модель мобільної системи, яка на відміну від відомих моделей трансферного навчання дозволяє скоротити об'єм пам’яті і робоче навантаження мережі при розгортанні на мобільних пристроях.

Практичне значення одержаних результатів. Робота має практичну цінність, оскільки розроблена система методів та програмних засобів використовує низку сучасних технологій, що забезпечують створеному програмному продукту вискоку конкурентоспроможність на ринку.

Апробація результатів роботи. Результати роботи доповідалися на Міжнародній науково-практичній Інтернет-конференції 2022 року «Електронні інформаційні ресурси: створення, використання, доступ».

Публікації. Результати роботи опубліковані у збірнику матеріалів Міжнародної науково-практичної Інтернет-конференції 2022 року «Електронні інформаційні ресурси: створення, використання, доступ» [1].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається з п’яти розділів. У першому розділі проводиться обґрунтування доцільності розробки, виконується аналіз стану проблеми, досліджуються існуючі аналоги, розглядаються методи розв’язання поставленої задачі та формулюються основні завдання розробки.

У другому розділі виконується розробка структури програмного продукту. Зокрема, розробляється алгоритм методу агрегування медіа контенту, описується структура інтерфейсу, проектується структура сторінок та розробляється дизайн.

Третій розділ містить процес розробки системи. Розглядається багатоваріантний аналіз та обґрунтування вибору засобів реалізації та середовища розробки, а також описується процес мобільного додатку.

У четвертому розділі розглядаються існуючі методики тестування та виконується тестування розробленої системи.

П’ятий розділ містить економічні розрахунки, які підтверджують доцільність розробки.

У додатки винесено технічне завдання, лістинг вихідного коду та ілюстративний матеріал до захисту магістерської кваліфікаційної роботи.

1 АНАЛІЗ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Суть технічної проблеми, що виникла на сучасному етапі розвитку науки, техніки та технологій

Мобільні телефони правлять світом і з цим важко сперечатися. Сьогодні у світі налічується понад 5,13 мільярда користувачів мобільних телефонів, що становить 65% населення. До появи смартфонів зробити чудове фото було трудомістким процесом. Тепер з розвитком технологій обробки зображень та самих камер зробити якісне фото не є особливо складним процесом. Смартфон, який завжди знаходиться у кишені, завжди є найкращою камерою, оскільки він завжди готовий робити знімки, і допомагає заморожувати незабутні моменти у житті. Подорожі, дні народження або просто зустріч з друзями – смартфон завжди під рукою і здійснивши декілька натискань він збереже момент життя, і допоможе згадати його у будь-який момент, коли користувачу цього захочеться. В цьому і є найбільша перевага мобільних телефонів над професійними чи аматорськими камерами.

Творче самовираження має бути доступним кожному, а не лише тим, хто може дозволити собі професійні інструменти та обладнання. Це одна з причин, чому рух мобільної фотографії вважається таким захоплюючим [2]. Оскільки наші смартфони та планшети продовжують удосконалюватись і стають нашими улюбленими творчими супутниками, ми бачимо, що все більше людей, ніж будь-коли, працюють над демократизацією творчості та розширюють межі того, що можливо за допомогою фотографії. Зі збільшенням технічних характеристик смартфонів, та в основному об’єму вбудованої пам’яті, збереження великої кількості зображень на ньому не є проблемою. Навпаки це є великим бонусом, оскільки дозволяє тримати всі сфотографовані моменти життя в одному місці, допомагаючи швидко знайти їх за необхідністю.

Однак зі збільшенням кількості фотографій на смартфоні збільшується і складність пошуку спогадів на ньому. Також за час експлуатації на смартфоні збирається багато і непотрібних фотографій, наприклад, документів, лічильників тощо. Певні моменти в році викликають у мене додаткову ностальгію — дні

народження, подорожі та свята найбільше — тому я дістаю свій телефон, щоб переглянути старі фотографії. Ви втрачаєте тепле й невиразне ностальгічне відчуття, коли вам доводиться гортати сотні дублікатів фотографій.

Для вирішення даної проблеми необхідне рішення, яке допоможе відокремити усі відзняті знімки від непотребу у галереї, і агрегувати їх у спогади, які потім з легкістю можна буде знайти в одному місці, та перенести свій мозок у момент здійснення знімку отримавши відчуття ностальгії.

1.2 Порівняльний аналіз аналогів

Для розробки програмного продукту необхідно визначитись з доцільністю розробки за допомогою порівняльного аналізу існуючих аналогів на ринку. Це допоможе не тільки визначити актуальність розробки, а й допоможе підкреслити переваги та недоліки існуючих програмних рішень, що допоможе імплементувати існуючі переваги продуктів, і таким чином отримати продукт, який ще на етапі проектування буде мати меншу кількість недоліків ніж у аналогів.

Крім основного завдання, з агрегування медіа контенту, розроблюваний програмний продукт має працювати на основних операційних системах, для сучасних мобільних пристроїв, тому необхідно ознайомитись з аналогами на обох платформах.

Вирішенням даної проблеми займаються в основному розробники головних операційних систем для мобільних пристроїв. На сьогоднішній день існують різні рішення даної проблеми. Програмні продукти допомагають зберігати зображення у хмарних сховищах, а також групувати їх і створювати спогади. Для основних операційних систем на сьогоднішній день існують такі рішення:

1) «Фотографії» iOS [3].

Photos – це програма для керування та редагування фотографій, розроблена Apple. Даний програмний продукт є вбудованим додатком для смартфонів, які працюють на операційній системі iOS. За його допомогою можна як просто переглядати знімки, які є на телефоні, так і переглядати згенеровані спогади, редагувати фото та відео контент. Додаток впорядковував фотографії за «моментом», як це було визначено за допомогою комбінації метаданих часу та місця, доданих до

фотографії. Починаючи з версії 5.0 (випущеної в 2019 році з macOS 10.15 Catalina), фотографії можна переглядати за роком, місяцем або днем. Згенеровані спогади за допомогою цього додатку зображено на рисунку 1.1.

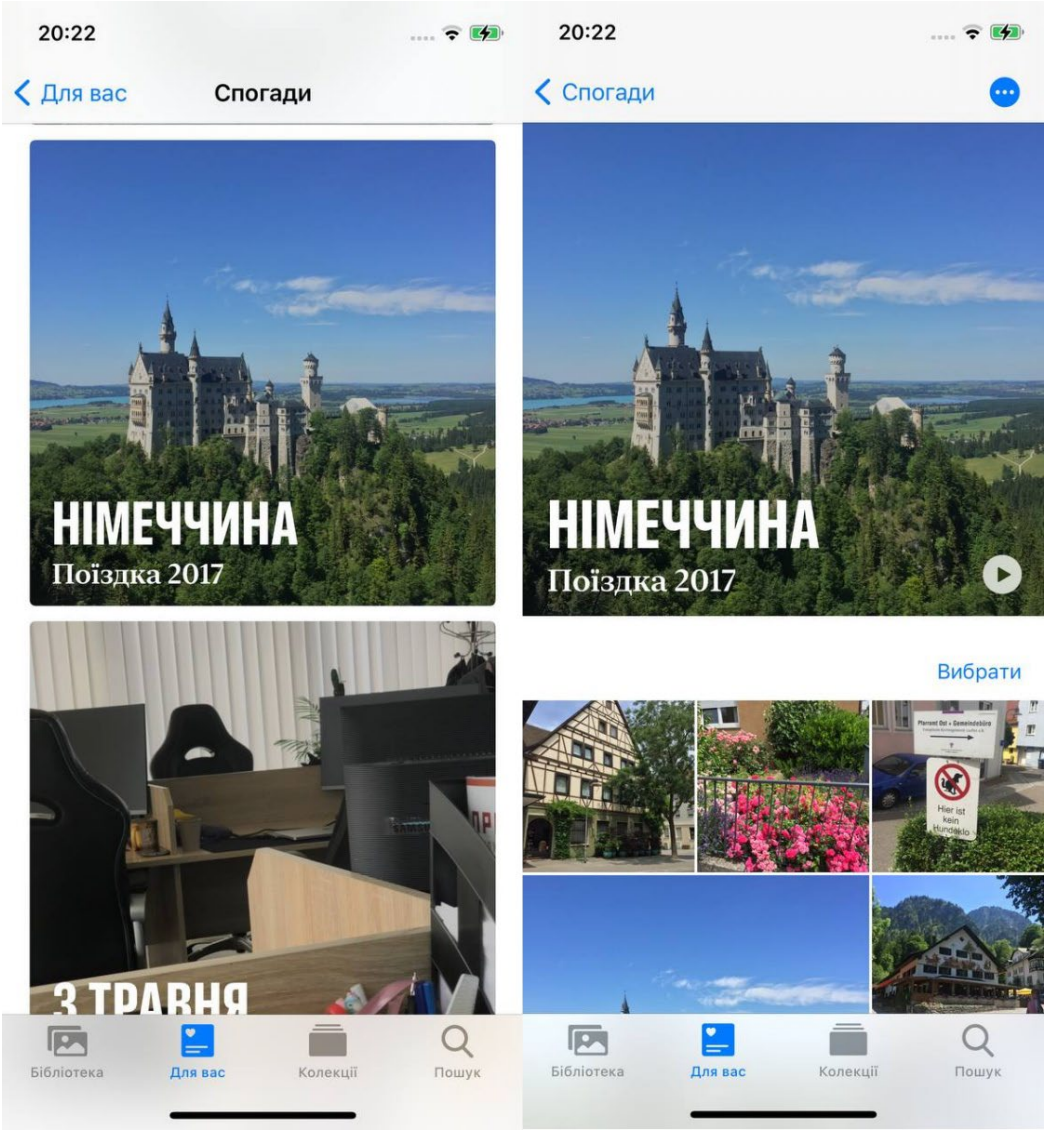


Рисунок 1.1 – Згенеровані спогади за допомогою додатку «Фотографії»

2) Google Photos [4]

Доволі популярний додаток для перегляду знімків на смартфоні [5]. Він попередньо встановлюється на усі нові смартфони, що працюють під операційною системою Android, а також мають додаток для iOS.

Даний додаток також має автоматичну генерацію спогадів, що і аналог на платформі iOS. Однак має більшу кількість функціоналу та категорій для групування. Спогади, згенеровані за допомогою додатку «Google Photos» зображено на рисунку 1.2.

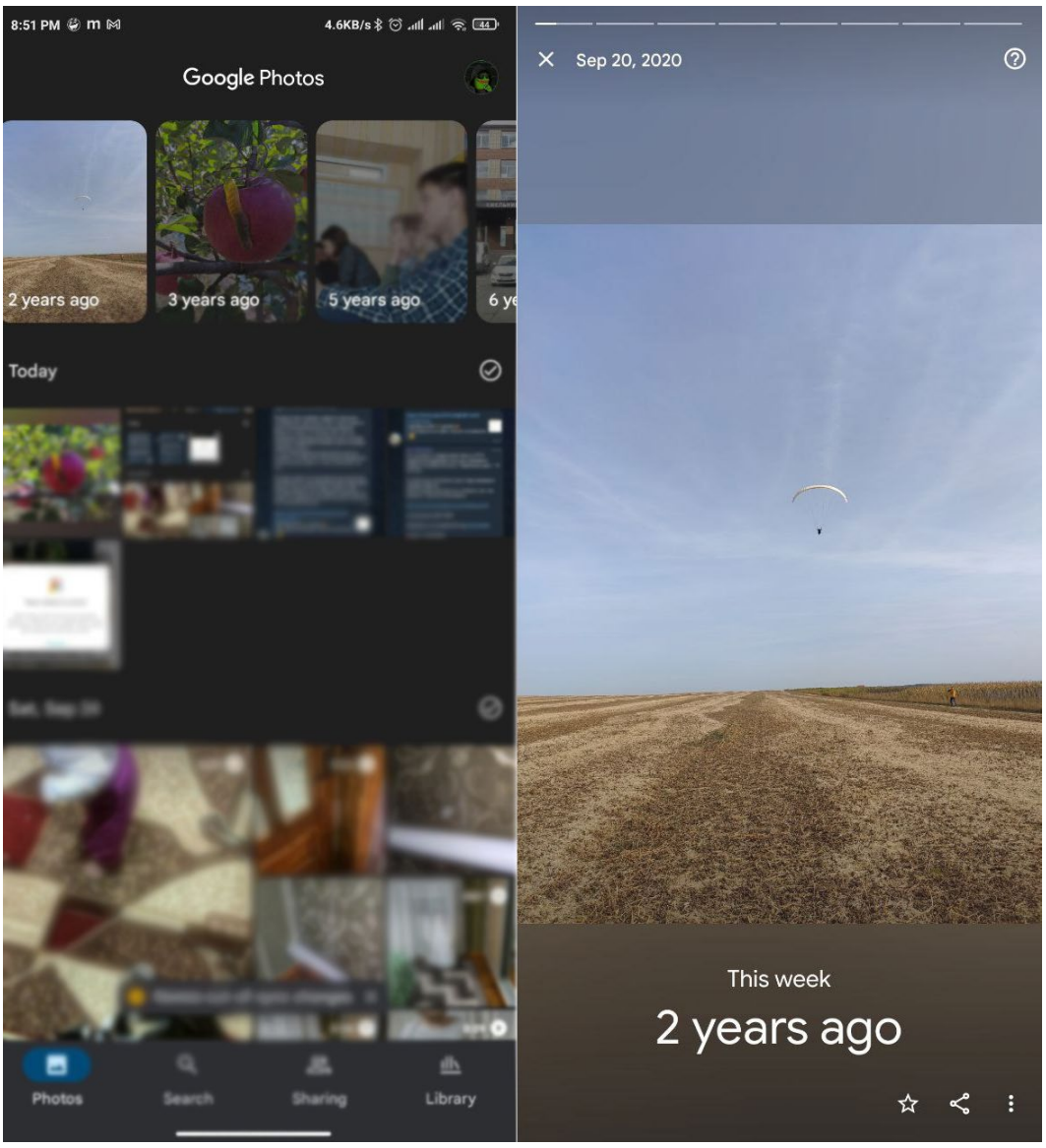


Рисунок 1.2 – Згенеровані спогади за допомогою додатку «Google Photos»

Обидва додатки мають вбудовану підтримку збереження медіа контенту на хмарних сховищах. Однак «Google Photo» для цього використовує «Google Drive» [6], а додаток «Фотографії» використовує «Apple iCloud» [7]. Головним недоліком, який присутній в обох рішеннях є відсутність приватності персональних даних, а саме, оскільки користувацькі медіа файли завантажуються на хмарні сховища, що належать корпораціям, вони не є максимально захищеними від взломів. Це означає, що корпорації Google та Apple мають доступ до Ваших персональних фотографій та відео без вашого відому. Досить частою практикою є так званий «злом аккаунтів», коли персональні дані переходять до рук зловмисників, найчастіше подібні випадки відбуваються у зіркових персон, оскільки їх дані мають особливу цінність.

Таким чином, однією з основних задач розроблюваного програмного продукту буде забезпечення максимальної приватності персональних даних з використанням різних технологій шифрування даних та обробка даних на самому пристрої.

Здійснивши опис існуючих аналогічних рішень необхідно створити порівняльну таблицю для виявлення переваг та недоліків існуючих рішень. Порівняльну характеристику аналогів зображено на таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Google Photos	Фотографії iOS	Memori
Агрегація медіа контенту	Так	Так	Так
Підтримка обох платформ	Так	Ні	Так
Створення спогадів власноруч	Ні	Ні	Так
Редагування автоматично створених спогадів	Ні	Ні	Так
Збереження спогадів у хмарному середовищі	Так	Так	Так
Додавання аудіо файлів до спогадів	Ні	Ні	Так
Додавання текстового опису до спогадів	Ні	Ні	Так
Відображення спогадів на карті	Так	Так	Так
Можливість генерації спогадів без з’єднання з мережею	Ні	Так	Так
Обробка медіа даних ресурсами пристрою	Ні	Ні	Так
Забезпечення приватності персональних даних через локальну обробку даних	Ні	Ні	Так
Зміна назви локації для одного або групи спогадів	Ні	Ні	Так

Таким чином, розроблювана система «Memori» є кращою за існуючі аналоги, що підтверджує актуальність розробки даної системи.

1.3 Аналіз методів і засобів реалізації

Для здійснення агрегування медіа контенту галереї мобільного пристрою необхідно відокремити зображення, що мають певну цінність від не потрібних фотографій для подальшої генерації спогадів. Для цього було вирішено використовувати методи та програмні засоби розпізнавання зображень, що допоможе знайти на сканованих зображеннях та відео об’єкти, які можна буде віднести до певних груп, на основі яких буде згенеровано спогади.

Розпізнавання зображень — це віднесення вихідних даних до певного класу за допомогою виділення істотних ознак, що характеризують ці дані, із загальної маси несуттєвих даних. При постановці задач розпізнавання намагаються користуватися математичною мовою.

Для оптичного розпізнавання образів можна застосувати метод перебору вигляду об'єкта під різними кутами, масштабами, зсувами й т. д. Для букв потрібно перебирати шрифт, властивості шрифту й т. д [8].

Другий підхід — знайти контур об'єкта й досліджувати його властивості (зв'язність, наявність кутів і т. д.)

Ще один підхід — використовувати штучні нейронні мережі. Цей метод вимагає або великої кількості прикладів задачі розпізнавання (із правильними відповідями), або спеціальної структури нейронної мережі, що враховує специфіку даної задачі.

Для реалізації даної роботи буде використано другий підхід. Він не потребує створення і навчання власної нейронної мережі, оскільки дозволяє використовувати вже готові приклади вирішення. Результат роботи штучних нейронних мереж для розпізнавання зображень зображено на рисунку 1.3.

Оскільки програмний продукт передбачено для мобільних пристроїв, то необхідно використати максимально оптимізоване вирішення проблеми. Серед таких готових рішень є Tensor Flow. TensorFlow — відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для

виявляння та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди [9].

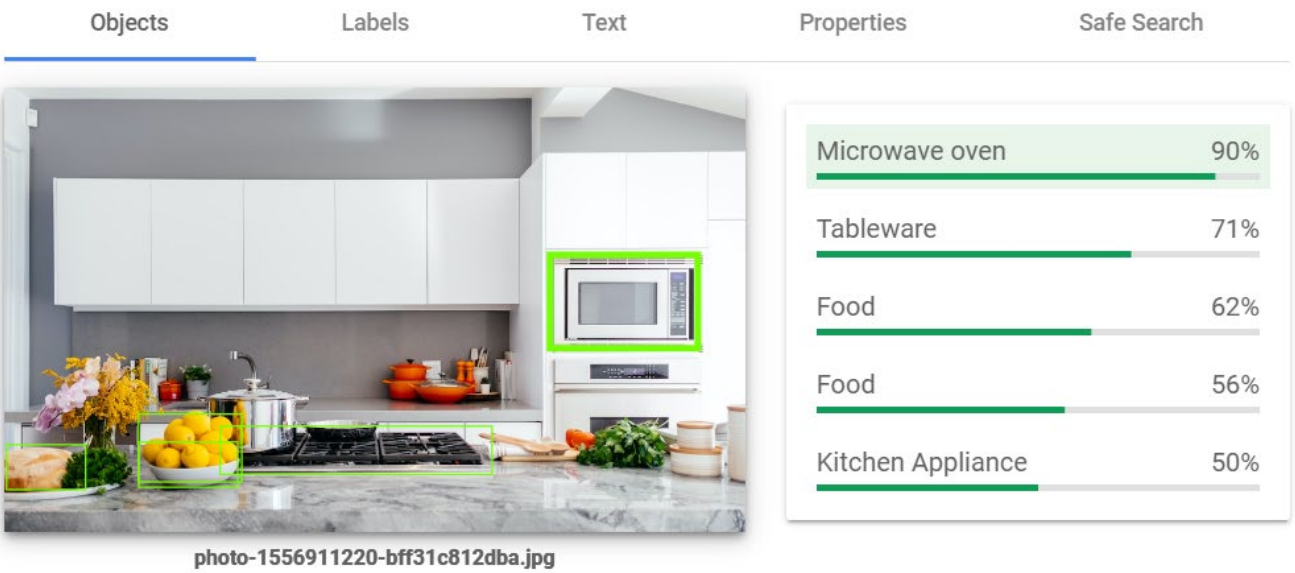


Рисунок 1.3 – Результат роботи штучних нейронних мереж для розпізнавання зображень

Головною перевагою Tensor Flow є те, що дана бібліотека є відкритою, та вміє використовувати тензорні ядра процесора, що пришвидшує обробку зображень в рази [10]. Функціональний потік виконання для моделей Tensor Flow зображено на рисунку 1.4.

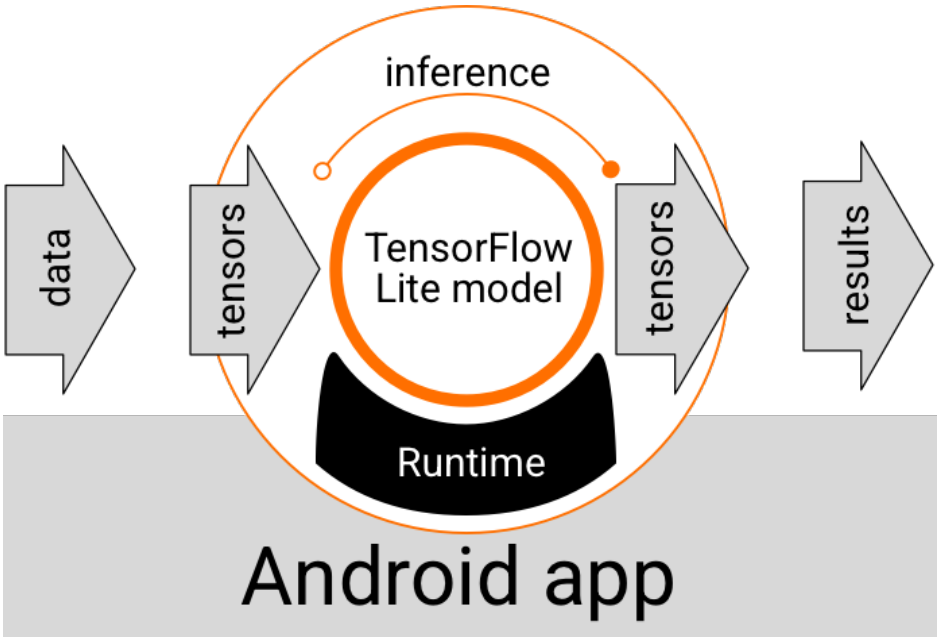


Рисунок 1.4 – Функціональний потік виконання для моделей TensorFlow

1.4 Постановка задач дослідження

Після аналізу питання розробки методів і програмних засобів розпізнавання зображень для агрегування медіа контенту було визначено наступні завдання, які потрібно виконати для розробки програмного продукту:

– дослідити та змоделювати предметну область;

– розробити алгоритм агрегування медіа контенту;

– сформувати вимоги до програмної системи;

– розробити специфікації вимог у обраному інструментальному програмному забезпеченні;

– виконати моделювання у інструментальному ПЗ;

– виконати моделювання об’єктно-орієнтованої програмної системи;

– розробити програмний продукт — виконати тестування розробленого продукту.

Для інформаційної системи виділені наступні нефункціональні вимоги:

– система має бути стабільною в роботі;

– швидко відповідати на дії користувача;

– для роботи з програмою має бути створений графічний інтерфейс користувача;

– захист приватних даних – система має використовувати локальні засоби розпізнавання зображень для забезпечення швидкодії її роботи, а також приватності особистих даних користувача.

Сформовано наступні функціональні вимоги для інформаційної системи:

– Спогади – повинна надавати можливість агрегування медіа контенту у спогади;

– Редагування – повинна надавати можливість редагування згенерованих спогадів;

– Створення – повинна надавати можливість створення спогадів власноруч;

– Резервне копіювання – повинна надавати можливість створювати резервні копії спогадів у хмарних середовищах.

1.5 Висновки

Таким чином у результаті аналізу предметної області було виявлено, що в сучасній сфері мобільних додатків існує потреба в реалізації програмного засобу для агрегування медіа контенту, яку обумовлює низька ефективність захисту приватних даних в вже існуючих програмних рішеннях.

Методи і засоби для агрегування медіа контенту з використанням програмної бібліотеки Tensor Flow забезпечують низку переваг, зокрема можливість обробки зображень безпосередньо за допомогою ресурсів самого мобільного пристрою, з використанням переваг архітектурних рішень, що наявні в сучасних процесорах. Також це означає, що користувацькі дані, а саме медіа файли не будуть відправлятись на серверні засоби і цим забезпечується саме приватність даних.

За допомогою створеної порівняльної таблиці з такими програмними продуктами як Apple Photos та Google Photos було визначено переваги та недоліки розроблюваної системи. Зокрема визначено, що розроблювана система має переваги в забезпеченні приватності користувацьких даних, а також матиме додатковий функціонал, який допомагатиме редагувати автоматично агреговані медіа файли.

Також визначено етапи розробки програмного продукту та його основні функціональні та не функціональні вимоги.

2 РОЗРОБКА СТРУКТУРИ МОБІЛЬНОГО ДОДАТКУ ТА АЛГОРИТМУ АГРЕГУВАННЯ МЕДІА КОНТЕНТУ

2.1 Проектування діаграм варіантів використання

Для чіткого визначення алгоритму дій користувача у системі необхідно спроектувати діаграму використанням [11]. Розглянемо варіанти використання для мобільного додатку, що відображає взаємодію користувача з застосунком.

Рисунок 2.1 – Діаграма варіантів використання системи

На даній діаграмі можна побачити усі можливі варіанти використання програмного засобу користувачем, зокрема:

– створення резервної копії – додаток має можливість створення резервної копії згенерованих зображень та бази даних додатку на хмарні сервіси в зашифрованому вигляді;

– автоматичне агрегування медіа – створення спогадів автоматичним способом за допомогою алгоритмів мобільного додатку;

– ручне створення спогадів;

– додавання малюнків до спогадів – користувач має можливість створення малюнків у спогадах;

– додавання аудіо файлів до спогадів – користувач має можливість записувати та відтворювати голосові повідомлення та прикріплювати їх до спогадів;

– редагування спогадів – користувач має можливість редагувати як автоматично створені так і створені власноруч спогади, в тому числі додавати до них нові фото та відео матеріали, додавати опис, та змінювати шаблон відображення фотографій на головному екрані;

– перегляд спогадів – перегляд як створених вручну так і автоматично згенерованих спогадів.

Переваги моделі варіантів використання:

– визначає користувачів і межі системи;

– визначає системний інтерфейс;

– зручна для спілкування користувачів з розробниками;

– використовується для написання тестів;

– є основою для написання документації користувача;

– добре вписується в будь-які методи проектування (як об'єктно-орієнтовані, так і структурні).

2.2 Розробка архітектури мобільного додатку

Добре опрацьована архітектура потрібна всім додаткам і складним, і шаблонним. З її допомогою економиться час, зусилля та гроші.

Архітектура мобільних додатків – сукупність рішень, як організувати програму. До неї входять: структурні елементи та інтерфейси, зв'язки між вибраними елементами, загальний стиль програми [12].

Хороша архітектура означає вигоду: простота та ефективність. Програму з такою архітектурою легше змінювати, тестувати та налагоджувати. Правильна архітектура мобільного додатку має відповідати на базові питання:

Ефективність: програма виконує поставлені завдання та виконує функції у будь-яких умовах. Система продуктивна, надійна та справляється з усіма навантаженнями.

Гнучкість: вибране рішення легко змінювати, і помилок стає менше. Можна змінити один елемент і це не стане фатальним для інших складових.

Розширюваність: у програму можна додавати скільки завгодно функцій, якщо потрібно.

Масштабованість: час на розробку та доповнення зменшується. Хороша архітектура дозволяє направити розробку у кілька паралельних потоків.

Тестування: додаток легко тестується, а значить, зменшується кількість помилок і збільшується його надійність.

Повторне використання: елементи та структуру можна використовувати в інших проектах.

Зрозумілість: код має бути зрозумілим якомога більшою кількістю людей. Над програмою працює багато людей. Хороша архітектура дозволяє новачкам швидко розібратися у проекті.

Для побудови програмного рішення використаємо патерн архітектури, а саме MVC (Model View Controller) [13]. Схема роботи даного патерну зображена на рисунку 2.2.

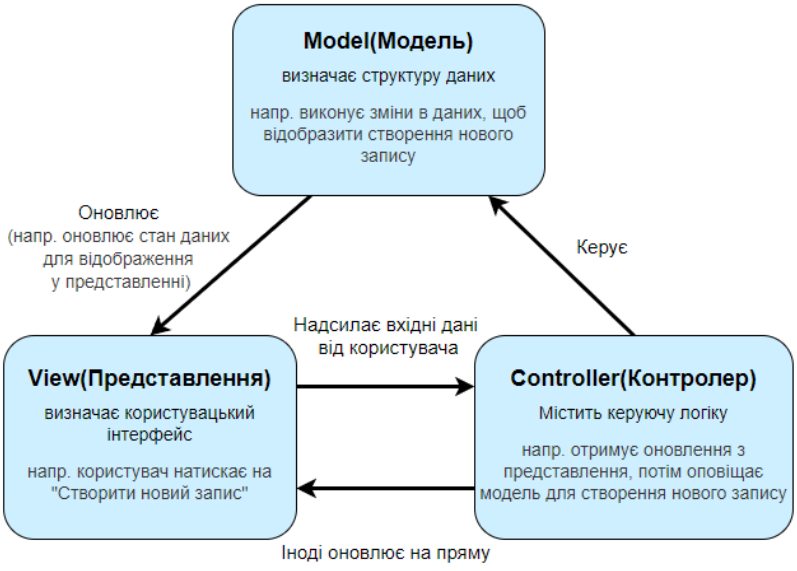


Рисунок 2.2 – Схема роботи Model-View-Controller.

Модель представляє єдине джерело істини, яке передає дані отримання в реальному часі або запити, пов’язані з базою даних.

Цей рівень може містити бізнес-логіку, перевірку коду тощо. Цей рівень взаємодіє з ViewModel для локальних даних або для реального часу. Дані надаються у відповідь на ViewModel.

ViewModel є посередником між View і Model, який приймає всі події користувача та запитує їх у Model для даних. Коли модель має дані, вона повертається до ViewModel, а потім ViewModel сповіщає ці дані в View.

ViewModel може використовуватися кількома представленнями, що означає, що одна ViewModel може надавати дані більш ніж одному View.

Представлення – це місце, де користувач взаємодіє з віджетами, які відображаються на екрані. Ці події користувача вимагають певних дій, які переходять до ViewModel, а решта ViewModel виконує цю роботу. Коли ViewModel має необхідні дані, він оновлює View.

Таким чином для розробки мобільного додатку буде використано архітектурний патерн Model View Controller, що допоможе розбити програму на модулі, і таким чином полегшаться розробка та тестування розроблюваного програмного забезпечення.

2.3 Розробка логічної та фізичної структури мобільного додатку

Для коректної роботи мобільного додатку необхідно розробити чітку і продуману структуру проєкту.

Структура мобільного додатку – це внутрішній устрій застосунку, розташування у ньому сторінок, розділів, підрозділів, додаткових матеріалів. З позиції розробника структуру сайту умовно можна поділити на два рівні: логічний і фізичний [14].

На логічному рівні структура мобільного додатку є сукупністю сторінок, які об'єднані між собою єдиним дизайном, стилем і посиланнями. Логічне проектування передбачає організацію інформації у додатку, побудову його структури та навігацію по розділах.

На рисунку 2.3 зображено логічну структуру мобільного додатку з інтерактивним інтерфейсом для агрегування медіа контенту.

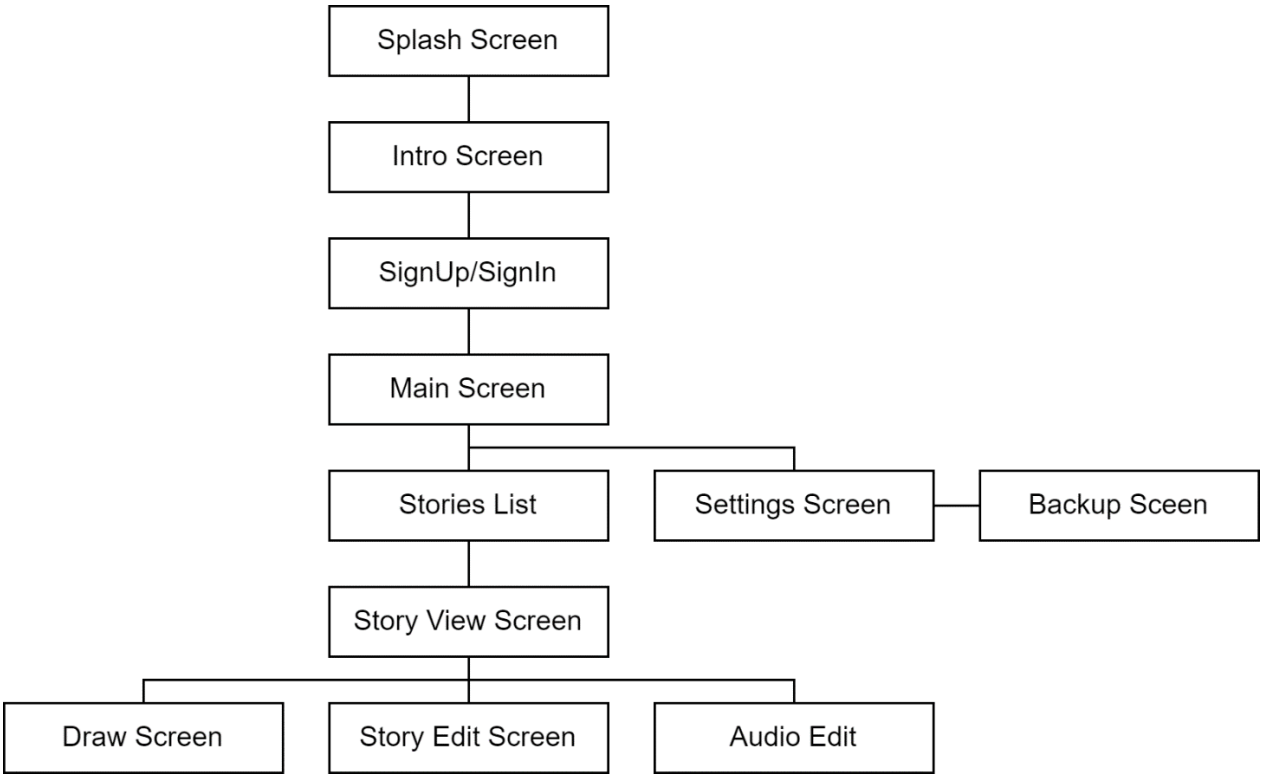


Рисунок 2.3 – Логічна структура мобільного додатку

При проектуванні сторінок мобільного додатку та засобів навігації краще дотримуватися чотирьох базових принципів, що засновані на сприйнятті інформації людиною:

– використання позначень – застосування слів і термінів, які є сталими і зрозумілими для більшості користувачів додатку, слід розміщувати змістовні елементи, посилання і елементи навігації з їх позначеннями, а також уникати невиразних термінів на екрані;

– доцільність – розділи мобільного додатку повинні містити інформацію та елементи, які відповідають даному розділу чи фрагменту екрану, невідповідні елементи краще перенести в інше місце або взагалі прибрати;

– одноманітність – однакові елементи на екранах повинні мати один і той же розмір і знаходитися в одному і тому ж місці;

– розділення на частини – користувачі мобільного додатку краще орієнтуються і швидше знаходять потрібні для них матеріали, коли вони візуально поділені на групи.

На фізичному рівні структура мобільного додатку є впорядкованим набором файлів різного типу (Екрани, зображення, мультимедійні файли).

Для правильної роботи скомпільованого проекту, для уникнення помилок з завантаженням вбудованих файлів, а також для полегшення роботи розробникам необхідно притримуватись таких правил для створення назв директорій та файлів:

– призначати назви папок, назви і розширення всіх файлів сайту з використанням символів лише латинського алфавіту і лише в рядковому регістрі, що гарантує універсальність і працездатність на різних платформах;

– призначати назви файлів логічними назвами, використовуючи символи латинського алфавіту, що буде корисно як під час створення сайту, так і згодом, коли потрібно буде внести певні зміни у проекті, а також для інших розробників, які працюють над сайтом.

Продумана і зручна файлова структура допомагає розробнику оптимізувати свою роботу, а також буде зрозумілою для інших фахівців, що працюють над проєктом.

На основі описаних правил було розроблено фізичну структуру програмного засобу для агрегування медіа контенту у вигляді мобільного додатку з інтерактивним інтерфейсом, яку наведено на рисунку 2.4.

Проектний код складається з основних компонентів, таких як:

– resources – використовується для збереження ресурсів додатку, таких як констант, посилань на зображення, файлів локалізації;

– router – містить файли конфігурацій всіх екранів додатку, а також переходів між ними;

– src – головна директорія файлів проекту, що містить базові класи екранів, роботи з репозиторіями, локальними даними, базою даних, містить класи логіки проекту, а також реалізацію всіх можливих екранів.

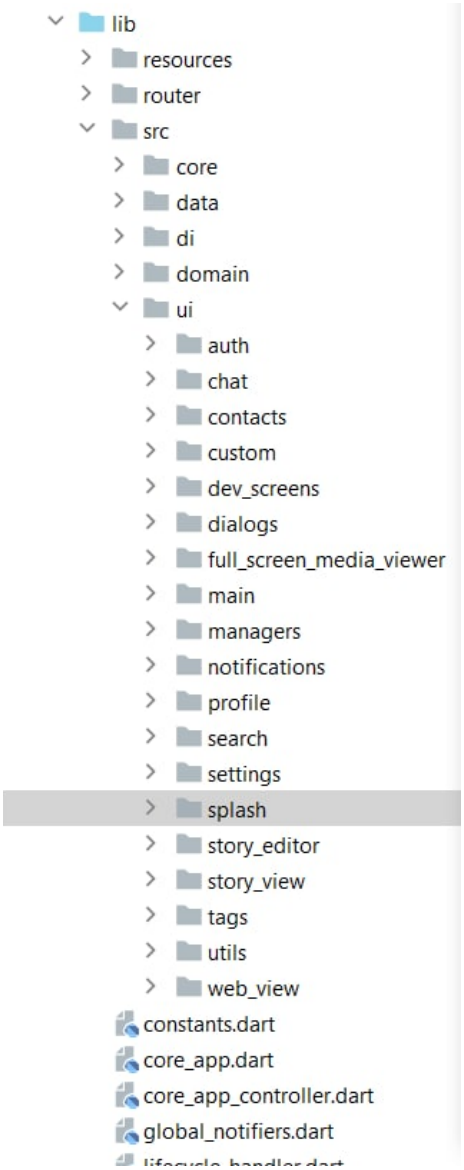


Рисунок 2.4 – Фізична структура мобільного додатку

2.4 Проектування алгоритму агрегування медіа контенту

Для роботи з зображеннями, на платформі pub.dev є необхідна бібліотека Photo Manager [15]. Даний плагін надає змогу працювати з нативними методами обох платформ, для отримання списку усіх медіа ресурсів пристрою. Тому для роботи алгоритму буде використано дану бібліотеку.

За допомогою вище згаданої бібліотеки ми отримуємо список усіх медіа файлів, що є на пристрої. Далі, в параметрах медіа файлів ми маємо дату їх створення. Використовуємо цю дату для подальшого агрегування.

Отримавши список дат медіа файлів, ми можемо поділити їх на умовні спогади. Тобто ділимо список медіа файлів на невеликі списки. За проміжок у цьому списку

відповідає часовий інтервал 2 години. Тобто якщо між створеними фотографіями інтервал більше ніж 2 години, то ми вважаємо, що ці фотографії належать до різних спогадів. Якщо проміжок між ними менше то ці фотографії попадають до одного списку. Таким чином ми отримали список спогадів, кожен з яких має декілька медіа файлів, які необхідно обробити далі. Список проміжків зображено на рисунку 2.5.

```
2021-09-15 10:41:50.579478 Range 1 = 2021-09-14 18:35:56.000 - 2021-09-14 18:35:56.000
2021-09-15 10:41:50.579483 Range 2 = 2021-09-13 20:47:32.000 - 2021-09-13 20:47:21.000
2021-09-15 10:41:50.579489 Range 3 = 2021-09-13 16:25:57.000 - 2021-09-13 16:25:57.000
2021-09-15 10:41:50.579494 Range 4 = 2021-09-13 07:49:55.000 - 2021-09-13 07:49:55.000
2021-09-15 10:41:50.579500 Range 5 = 2021-09-12 16:17:48.000 - 2021-09-12 16:13:36.000
2021-09-15 10:41:50.579505 Range 6 = 2021-09-12 08:13:47.000 - 2021-09-12 08:13:47.000
2021-09-15 10:41:50.579510 Range 7 = 2021-09-09 18:24:55.000 - 2021-09-09 18:24:55.000
2021-09-15 10:41:50.579515 Range 8 = 2021-09-06 15:37:53.000 - 2021-09-06 15:37:53.000
2021-09-15 10:41:50.579521 Range 9 = 2021-09-04 14:12:14.000 - 2021-09-04 10:45:09.000
2021-09-15 10:41:50.579526 Range 10 = 2021-09-03 09:02:46.000 - 2021-09-03 09:02:46.000
2021-09-15 10:41:50.579531 Range 11 = 2021-09-02 09:58:03.000 - 2021-09-02 08:36:50.000
2021-09-15 10:41:50.579536 Range 12 = 2021-09-01 17:05:40.000 - 2021-09-01 17:05:01.000
2021-09-15 10:41:50.579542 Range 13 = 2021-08-31 15:44:02.000 - 2021-08-31 15:44:02.000
2021-09-15 10:41:50.579547 Range 14 = 2021-08-29 21:40:41.000 - 2021-08-29 21:22:16.000
2021-09-15 10:41:50.579552 Range 15 = 2021-08-29 09:41:52.000 - 2021-08-29 09:41:52.000
2021-09-15 10:41:50.579557 Range 16 = 2021-08-29 00:25:24.000 - 2021-08-29 00:25:24.000
2021-09-15 10:41:50.579562 Range 17 = 2021-08-21 18:07:08.000 - 2021-08-21 18:07:08.000
2021-09-15 10:41:50.579566 Range 18 = 2021-08-15 13:55:51.000 - 2021-08-15 13:55:51.000
```

Рисунок 2.5 – Список проміжків зі спогадами

Далі нам необхідно створити спогад з кожного проміжку, якщо це можливо.

Для цього ми використовуємо алгоритм розпізнавання зображень TensorFlow. За допомогою вже навчаних моделей на кожному зображенні, яке відноситься до проміжку ми намагаємось знайти об’єкти. Список отриманих об’єктів з медіа файлу зображено на рисунку 2.6.

```
-> started detection
-> -> got image = /var/mobile/Media/PhotoData/CPLAssets/group111/B3B086FD-1C2A-4F34-BA33-88A00B8A479C.HEIC
-> -> -> image saved with milliseconds = 9
-> -> -> Tflite.detectObjectOnImage finished with milliseconds = 178 2021-09-15 10:42:12.877486 and returned 5 detected objects
-> -> -> -> added detected object = DetectedObject{name: laptop, confidence: 0.6796875}
-> -> -> -> added detected object = DetectedObject{name: person, confidence: 0.64453125}
-> -> -> -> added detected object = DetectedObject{name: potted plant, confidence: 0.5859375}
-> -> -> -> added detected object = DetectedObject{name: dining table, confidence: 0.5234375}
-> -> -> -> added detected object = DetectedObject{name: tv, confidence: 0.5}
```

Рисунок 2.6 – Список знайдених об’єктів

У списку знайдених об’єктів ми бачимо назву об’єкту та вірогідність того, що цей об’єкт є на цьому зображені. Таким чином ми можемо зрозуміти чи є на даному

медіа файлі об’єкти і чи варто включати даний медіа файл у спогад. Якщо медіа файл взагалі не має жодного об’єкту, то можна вважати що він непотрібний.

Далі використовуємо таблицю порогових значень для кожного об’єкта. Використавши цю таблицю ми бачимо які об’єкти і які мінімальні порогові значення для них є. Тобто, якщо на медіа файлі знайдено об’єкт «Apple» з вірогідністю 0.7, а порогове значення у таблиці це 0.6, то ми можемо вважати, що даний медіа файл можна використати для спогаду. Якщо б вірогідність була меншою, то даний файл було б відкинуто як не потрібний.

Відфільтрувавши таким чином медіа файли ми отримаємо зображення, з яких можна далі генерувати спогад. Для головного зображення спогаду буде використано медіа файл, у якого найбільша вірогідність знайденого об’єкту.

Далі необхідно виконати створення назви даного спогаду. Для цього ми використовуємо медіа файл, який ми обрали як головний і переходимо до його обробки. Серед мета даних медіа файлу також можна знайти дані геолокації, де було створено медіа файл. Якщо такі дані знайдено, то використовуючи сервіси Google знаходимо назву місця за координатами.

Далі використовуємо такі дані, як час створення медіа файлу, геолокація, об’єкти, знайдені на зображенні.

За допомогою часу, у який було створено медіа файл визначаємо першу частину назви, за таблицею 2.1.

Таблиця 2.1 – Таблиця визначення частини дня

Час	Назва
01:30 – 04:00	Early morning
04:00 – 06:00	Morning
06:00 – 10:30	Late morning
10:30 – 12:00	Early afternoon
12:00 – 13:00	Afternoon
13:00 – 15:00	Late afternoon
15:00 – 16:00	Early evening

16:00 – 18:30	Evening
18:30 – 21:30	Late evening
21:30 – 23:00	Night
23:00 – 01:30	Late night

За допомогою знайдених об'єктів на медіа файлі можна визначити категорію спогаду, яка піде до назви. За допомогою таблиці 2.2 визначаємо назву категорії та додаємо її до назви спогаду.

Таблиця 2.2 – Таблиця категорій визначених об’єктів

Назва категорії	Список об’єктів
Travel	airplane, backpack, suitcase
Nature	bear, bird, giraffe, sheep, zebra
Sport	baseball bat, baseball glove, frisbee, skateboard, sports ball, tennis racket, skis, snowboard
Meal	apple, banana, bottle, bowl, broccoli, cake, carrot, dining table, donut, fork, hot dog, knife, orange, pizza, sandwich. spoon, wine glass
Indoors	bed, couch
Outdoor	bench, bicycle, boat, bus, car, cow, elephant, fire hydrant, handbag, horse, kite, motorcycle, parking meter, stop sign, traffic light, train, truck
Home	book, chair, clock, hair drier, microwave, potted plant, refrigerator, scissors, teddy bear, toilet, toothbrush, vase
Pets	cat, dog
Gadgets	cell phone, keyboard, laptop, mouse, remote

Kitchen	oven, sink
People	person
Clothes	tie

За допомогою цих даних можемо згенерувати назву спогаду за таким алгоритмом: [Частина дня] + [Категорія] + [Назва локації] + [Місяць, рік]. Блок схему алгоритму зображено на рисунку 2.7.

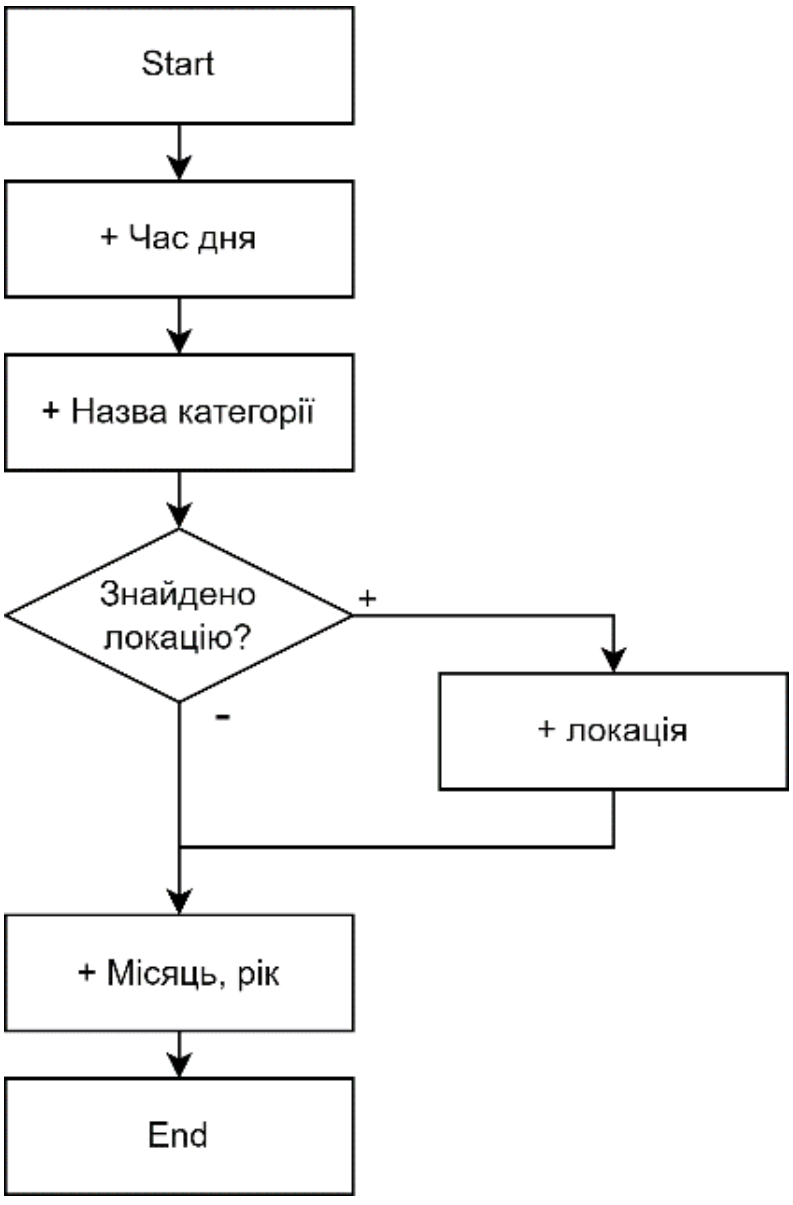


Рисунок 2.7 – Алгоритм створення назви спогаду

Таким чином отримаємо назву, наприклад «Late evening pets at Berlin in March`20» та додамо її вже до згенерованого спогаду.

2.5 Проектування бази даних

Для збереження згенерованих спогадів буде використовуватись база даних, яку в разі необхідності можна зберегти разом з медіа файлами на хмарному сховищі таким чином отримавши резервну копію згенерованих спогадів.

Для збереження даних локально буде використано Sqflite [16], оскільки вона має реалізацію для платформ iOS та Android.

SQLite — це система баз даних, написана мовою програмування C. Це не окрема програма; скоріше це бібліотека, яку розробники програмного забезпечення вбудовують у свої програми. Таким чином, він належить до сімейства вбудованих баз даних.

Дана СУБД є досить легкою і не потребує великої продуктивності, що досить важливо коли розробка ведеться для мобільних пристроїв. Проте вона також має свої недоліки, зокрема вона не має всіх можливостей, які має звичайний SQL, трохи відрізняється за своїм синтаксисом та обмежена в типах підтримуваних даних. Проте всі ці недоліки не є критичними для розроблюваної системи.

Для збереження даних історії було створено таблицю Story. Вона містить наступні поля:

– id;
– date_Start;
– date_General;
– date_End;
– main_Image;
– canvas_File;
– audio_File;
– data.

Поле data пов’язано зв’язком один до одного з таблицею StoryData, що містить наступні поля:

– -id;
– -story_id;
– -medias.

Для збереження даних про таги було створено таблицю Tag, вона має наступні поля:

– id;

– name.

Вона має зв’язок з таблицею Story через суміжну таблицю Story_Tag, оскільки інакше матиме зв’язок багато до багатьох.

Для збереження даних про локації було створено таблицю Location, яка пов'язана з таблицею Story зв’язком один до одного та містить наступні поля:

– id;

– latitude;

– longitude;

– name;

– story_id.

Таким чином обравши СУБД за допомогою якої буде реалізовано збереження даних приступимо до створення схеми бази даних.

В результаті отримаємо схему бази даних, зображеної на рисунку 2.8.

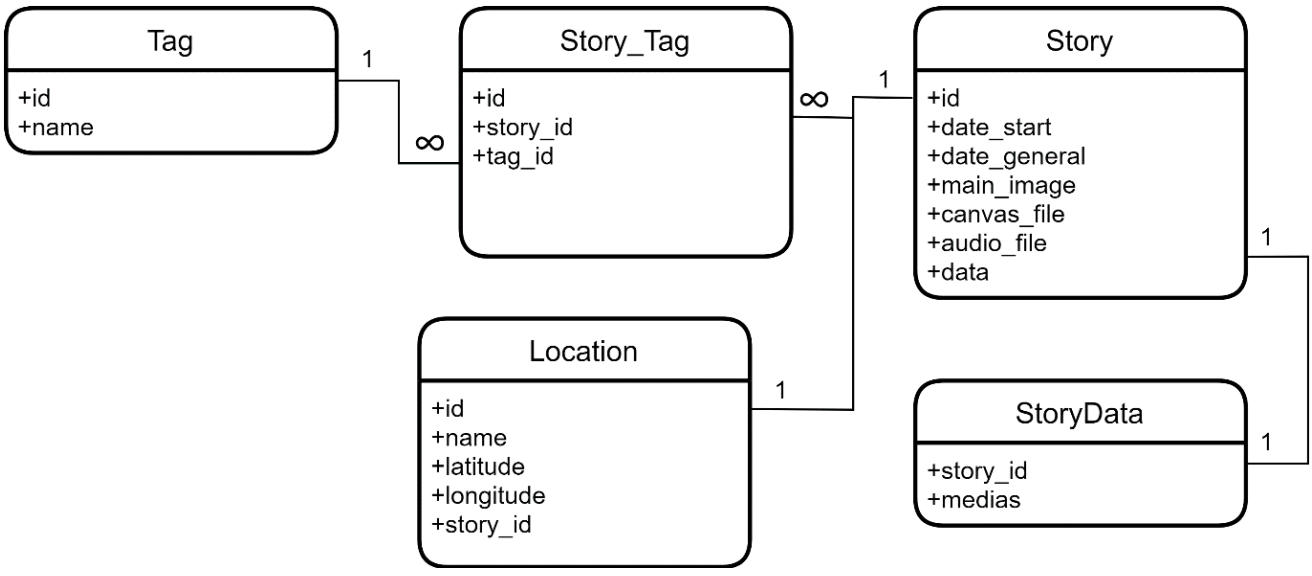


Рисунок 2.8 – Розроблена схема бази даних

2.6 Розробка структури екранів мобільного додатку

Структура додатку - екрани, розділи, навігація (посилання); розташування цих елементів і їх зв'язок між собою. Для того, щоб зробити структуру додатку наочною,

можна використовувати нескладні геометричні фігури та написи [14]. Розміщуючи елементи на робочій області і стрілочками з'єднавши їх, можна отримати схему як відвідувач повинен переходити по додатку.

При розробці структури додатку важливо знати той факт, що екрани додатку можуть бути різного ступеня вкладеності. Різниця рівня визначається ступенем важливості, як розділ і підрозділ.

Найпростіша структура додатку – лінійна. Екрани йдуть один за одним, і користувач повинен переглядати їх як слайд-шоу. У лінійній структурі не існує поділу контенту на рівні. Всі екрани на таких додатках рівноправні, і їх повинен побачити кожен відвідувач. Незважаючи на простоту реалізації лінійної структури, недоліків у неї набагато більше, ніж переваг.

Реалізація лінійної структури не є абсолютно ніякої складності. Найпростіший варіант додатку - набір екранів кожен з яких є посилання на наступну попередню сторінку. Але при цьому на кожній сторінки обов'язково має бути якийсь заголовок і посилання на першу сторінку. Інакше відвідувачі, що потрапили в середину сайту, наприклад, з пошукової системи, нічого не зрозуміють і майже напевно покинуть проект розчарованими. Крім того, дуже корисним є представлення загального числа екранів і певне виділення номеру тієї з них, на якій користувач перебуває в даний момент. Інакше перегляд проекту перетвориться для відвідувачів в подорож у невідомість.

Лінійна структура додатку з альтернативами і варіантами. Основою цієї структури є просте лінійне розміщення екранів. Однак у додатках, побудованих за цим принципом, відвідувачі можуть проявити деяку ініціативу, полегшивши для себе пошук потрібної інформації. Під альтернативами в даному випадку розуміється вибір між двома гілками. Найчастіше подібна структура використовується для збору інформації про відвідувача.

Лінійна структура з альтернативами і варіантами зручна в тому плані, що з одного боку вона дозволяє інженеру контролювати діяльність відвідувачів, направивши їх в певне русло. А з іншого боку проявити деяку ініціативу, яка

дозволить їм, по-перше, відчути свободу, а по-друге, полегшити доступ до потрібної саме їм інформації.

Лінійна структура сайту з відгалуженнями. Це теж контрольована структура, яка нагадує дорогу з тупиковими стежками, які час від часу відгалужуються від неї. Тобто відвідувач послідовно переходить з однієї сторінки на іншу. Якщо інформація, розміщена на якийсь із них його зацікавила, і він хоче дізнатися подробиці, то може перейти на відгалуження, а потім повернутися назад на основну «доріжку».

Головною перевагою даної структури є те, що до неї легко перейти зі звичайного лінійного розміщення екранів. Таке часто буває, коли створений одного разу сайт перестає задовольняти збільшеним вимогам, а глобальна переробка з тих чи інших причин неможлива. В цьому випадку інженер може швидко і без всяких проблем розширити проект.

Деревоподібна структура додатку – універсальний спосіб розміщення екранів. Вона підходить для створення практично будь-яких типів додатків [14]. Її принцип полягає в тому, що користувач при заході на головну сторінку виявляється перед вибором, куди йти далі. Після переходу в потрібний розділ, він підбирає необхідний підрозділ і т.п. У деревовидної структури дуже багато достоїнств, але так само є і недолік.

Йдеться про те, що в структурі дерева дуже складно дотримуватися балансу між глибиною і шириною. Якщо «дерево» додатку буде рости тільки вглиб, то користувачам, щоб дійти до якоїсь інформації, доведеться завантажити і обробити багато екранів, що буде дратувати користувачів. А якщо створити дуже широку деревоподібну структуру, то відвідувачі будуть змушені щоразу витрачати дуже багато часу для вибору потрібної їм гілки. А це теж погано. Таким чином, при використанні деревовидної структури додатку необхідно постійно стежити за її розростанням і дотримуватися золотої середини.

Гратчаста структура вже на порядок складніше всіх розглянутих вище. У ній всі екрани також розміщуються в різних гілках. Але у користувача є можливість переміщатися по ним не тільки вертикально (вгору-вниз), а й горизонтально (тобто між гілками на різних рівнях). Використовується решітка в основному тільки в

каталогах. При цьому переміщення між гілками на глибинних рівнях здійснюється за допомогою відсилань на рубрики в інших розділах.

Використання гратчастої структури в інших проектах недоцільне. По-перше, вона відносно складна в реалізації. По-друге, звертатися з «гратами» потрібно з дуже великою обережністю, так як структура додатку може вийти дуже заплутаною, і відвідувачі будуть змушені довго блукати в пошуках потрібної їм інформації.

2.7 Розробка інтерфейсу мобільного додатку

Розробка структури інтерфейсу є одним з основних етапів ескізних проектів, які робляться перед виготовленням оригінал-макету мобільного додатку, і використовується при розробці дизайну мобільних додатків та сайтів, щоб показати структуру мобільного додатку, налаштовуються схеми шляху користувача, найбільш важливі елементи інтерфейсу користувача, їх положення і взаємозв'язок між сторінками сайту [18]. Каркаси сторінок сайту відображають в чорно-білому кольорі найбільш важливі елементи інтерфейсу, такі як заголовок і нижній колонтитул сайту, форма контактів, навігація і т.п.

Каркасне моделювання – це процес, який може значно скоротити час, необхідний для проектування і розробки, усуваючи потенційні візуальні відволікаючі фактори і фокусуючи увагу розробників проекту на базовій функціональності і стратегічних факторах маркетингу.

Ось кілька ключових причин, що визначають, чому перед створенням сайту бажано (обов'язково) потрібно спочатку зробити його прототип:

– це дозволить отримати чітку картину того, яка саме інформація буде необхідна на кожній сторінці сайту до розробки його дизайну;

– це дозволить раціонально витрачати час і зосереджуватися саме на тому, для чого призначена кожна сторінка. Ретельне планування має першорядне значення;

– це дозволить розробнику підстрахувати себе, в разі необізнаних клієнтів, які схильні змінювати свою думку на стадії розробки проекту. Якщо сайт належним чином планується на етапі каркасного моделювання, то функціональність окремих сторінок не повинна сильно змінитися;

– це дозволить встановити точку відновлення. Коли клієнт підписує каркасну модель, це означає, що він погоджується з тим, що буде перебувати на цій сторінці;

– обізнаність клієнта також грає тут важливу роль, і він повинен знати, що зміна вже підписаних елементів на каркасній моделі може з дуже великою ймовірністю збільшити бюджет;

– це дозволить отримати чітке уявлення про те, як будуть реагувати відвідувачі на сайт без колірної схеми або елементів дизайну;

– це дозволить видалити зайві елементи, які можуть виявитися непотрібними для майбутнього сайту;

– прототип досить легко створити, дозволяючи плавно і ефективно здійснювати процес планування;

– цей процес знижує ймовірність збільшення обсягу роботи з розробки дизайну;

– це дає дизайнеру чітке уявлення про те, що потрібно зробити;

– це дозволяє інтенсивно залучити клієнта в процес планування на ранньому етапі розробки сайту і дозволяє активно погоджувати весь процес планування між обома сторонами.

2.8 Розробка дизайну мобільного додатку

Користувацький інтерфейс застосунку має бути зручним для використання. Перш за все, інтерфейс має бути простим. Найкращі інтерфейси практично непомітні для користувача. Вони уникають непотрібних елементів і чітко викладають необхідну інформацію повідомленнях застосунку.

В інтерфейсі має бути узгодженість елементів. Використовуючи загально прийняті, типові елементи інтерфейсу, користувачі відчуватимуть себе комфортніше та зможуть виконати бажані операції швидше, адже вони вже будуть знайомі з різними елементами інтерфейсу та приблизно знатимуть чого очікувати. Також важливо створювати шаблони в мові, макеті та дизайні різних функціональних застосунку, щоб підвищити ефективність [17].

Макет різних сторінок інтерфейсу має бути цілеспрямованим, в кожного елемента має бути чітке призначення і причина його місцезнаходження. Елементи інтерфейсу які часто використовуються користувачем слід розташувати внизу, адже при використанні мобільних пристроїв, користувачі найчастіше виконують дії саме великим пальцем руки. Ретельне розміщення елементів також може допомогти привернути увагу до найважливіших частин інформації, а також спростити її сприйняття.

Інтерфейс застосунку має швидко відповідати на дії користувача та завжди показувати що саме відбувається в даний момент. Велика кількість прихованих процесів та операцій зменшують зрозумілість та ускладнюють сприйняття інформації в застосунку.

Усе залежить від того на скільки розробник розуміє своїх користувачів, включаючи розуміння їхніх цілей, навичок, уподобань та тенденцій, що неможливо ідеально зробити одразу. Цілком розумно в ході експлуатації застосунку користувачами отримувати від них відгуки які сприятимуть позитивним змінам інтерфейсу в майбутньому.

Основуючись на фактори, описані вище приступаємо до роботи над інтерфейсом програмного продукту.

Для розробки макету дизайну додатку буде використано програмне рішення Figma [19]. Figma — векторний онлайн-сервіс розробки інтерфейсів та прототипування з можливістю організації спільної роботи, що розробляється однойменною компанією. Працює у двох форматах: у браузері та як клієнтський додаток на десктопі користувача. Зберігає онлайн-версії файлів, з якими працював користувач.

Було розроблено стартові екрани, що зображені на рисунку 2.9, де зображено логотип додатку на стартовому вікні. Екран привітання нового користувача, що реалізований у вигляді слайдеру з зображеннями, на яких користувач може побачити можливості застосунку.

Розроблено інтерфейс головного вікна, що зображений на рисунку 2.10. На головному вікні користувач матиме змогу переглядати вже агреговані медіа файли, а

також запускати нове сканування галереї. З цього екрану користувач може перейти на всі інші екрани додатку, такі як перегляд історій, налаштування, резервне копіювання та створення спогадів власноруч. Також на цьому екрані присутній календар та карта. За допомогою календаря користувач зможе переглянути в які дні він має згенеровані спогади та легко перейти до потрібної дати. На екрані карти зображені усі спогади, у яких було знайдено локацію у мета даних медіа файлів, натиснувши на зображення спогаду на карті користувач перейде на екран перегляду даного спогаду. Також на головному екрані користувач може додати тег до спогаду. Таким чином він зможе здійснювати пошук не лише по назві спогадів, а й фільтрувати спогади по тегах, наприклад, здійснивши фільтрування, по тегу «Сім'я» користувач отримає список усіх спогадів, що прив'язані до даного тегу. Дизайн вікна, призначеного для створення тегів, зображено на рисунку 2.12.

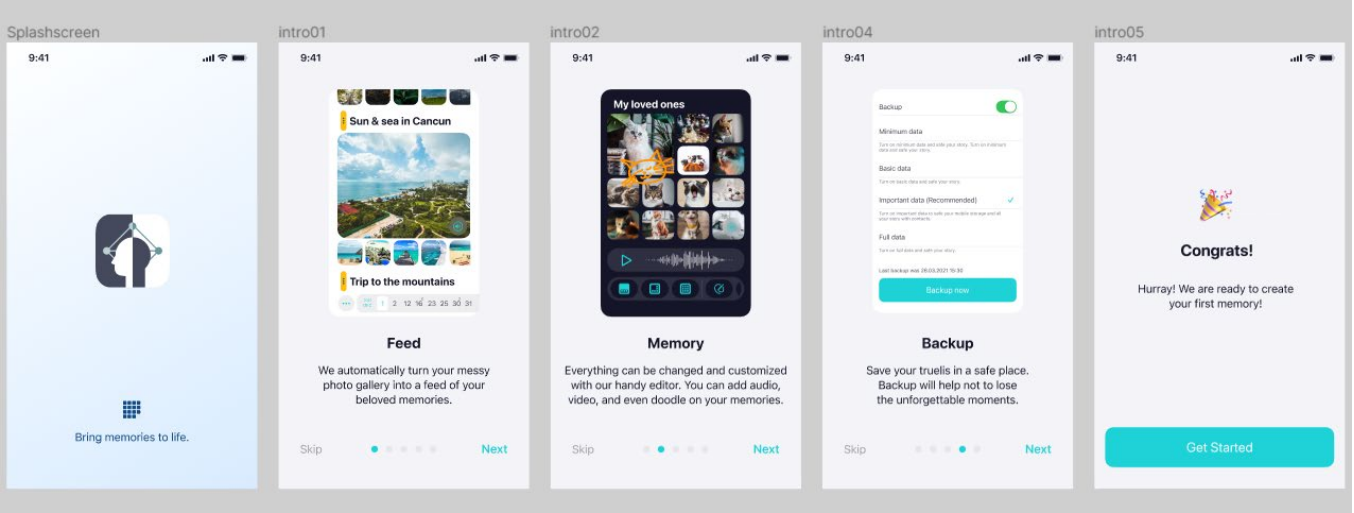


Рисунок 2.9 – Дизайн стартових екранів

Екрани редагування спогадів, створення малюнків та створення аудіо запису зображені на рисунку 2.11. На екрані редагування користувач матиме змогу здійснити наступний функціонал:

– зміна медіа контенту місцями;
– додавання назви спогаду;
– додавання опису до спогаду;
– додавання нових фото та відео матеріалів до спогаду;
– перехід на екран додавання аудіо фалу;

– перехід на екран додавання малюнку;

– зміна головного шаблону;

– зміна шаблонів комірок головного шаблону.

На екрані створення малюнку користувач матиме змогу власноруч створити малюнок, який буде відображено поверх головного шаблону спогаду. Для малювання користувач матиме змогу змінювати колір курсору, а також стирати ділянки малюнку за допомогою ластику.

На екрані додавання аудіо файлу користувач матиме змогу записати аудіо повідомлення за допомогою мікрофону, або завантажити аудіо файл з мобільного пристрою. Цей аудіо файл буде прикріплений до спогаду, та його можна буде відтворити на екрані редагування або на екрані перегляду спогаду.

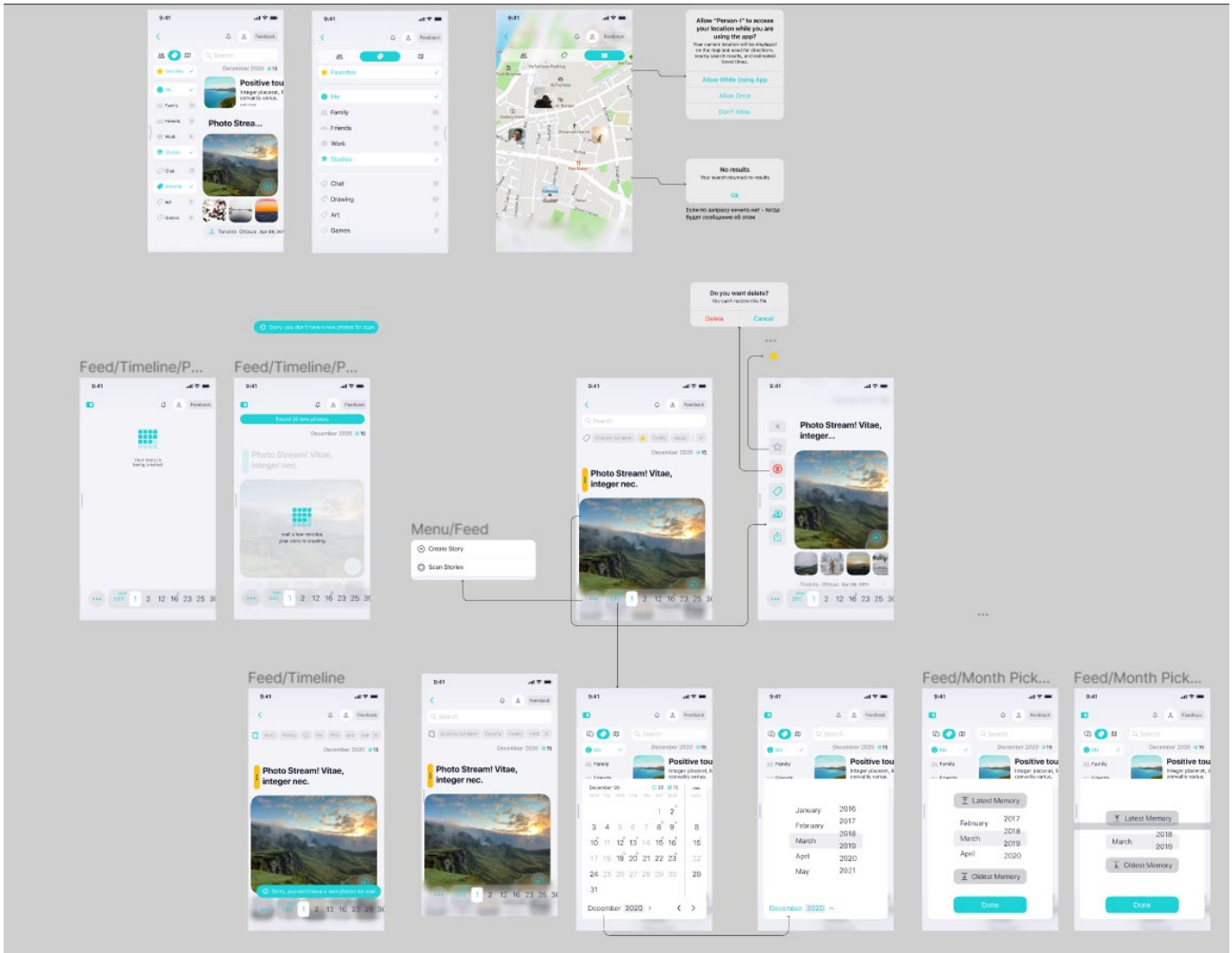


Рисунок 2.10 – Дизайн головного екрану

Екран перегляду спогадів зображено на рисунку 2.13. На даному екрані користувач може як переглядати спогади згенеровані додатком, або створені власноруч. На даному екрані присутній функціонал зміни дати історії та зміни локації. Користувач має можливість перегляду фото та відео матеріалів у повноекранному режимі за допомогою екрану слайдеру, а також з даного екрану користувач може додати тег до спогаду або перейти на екран редагування даного спогаду.

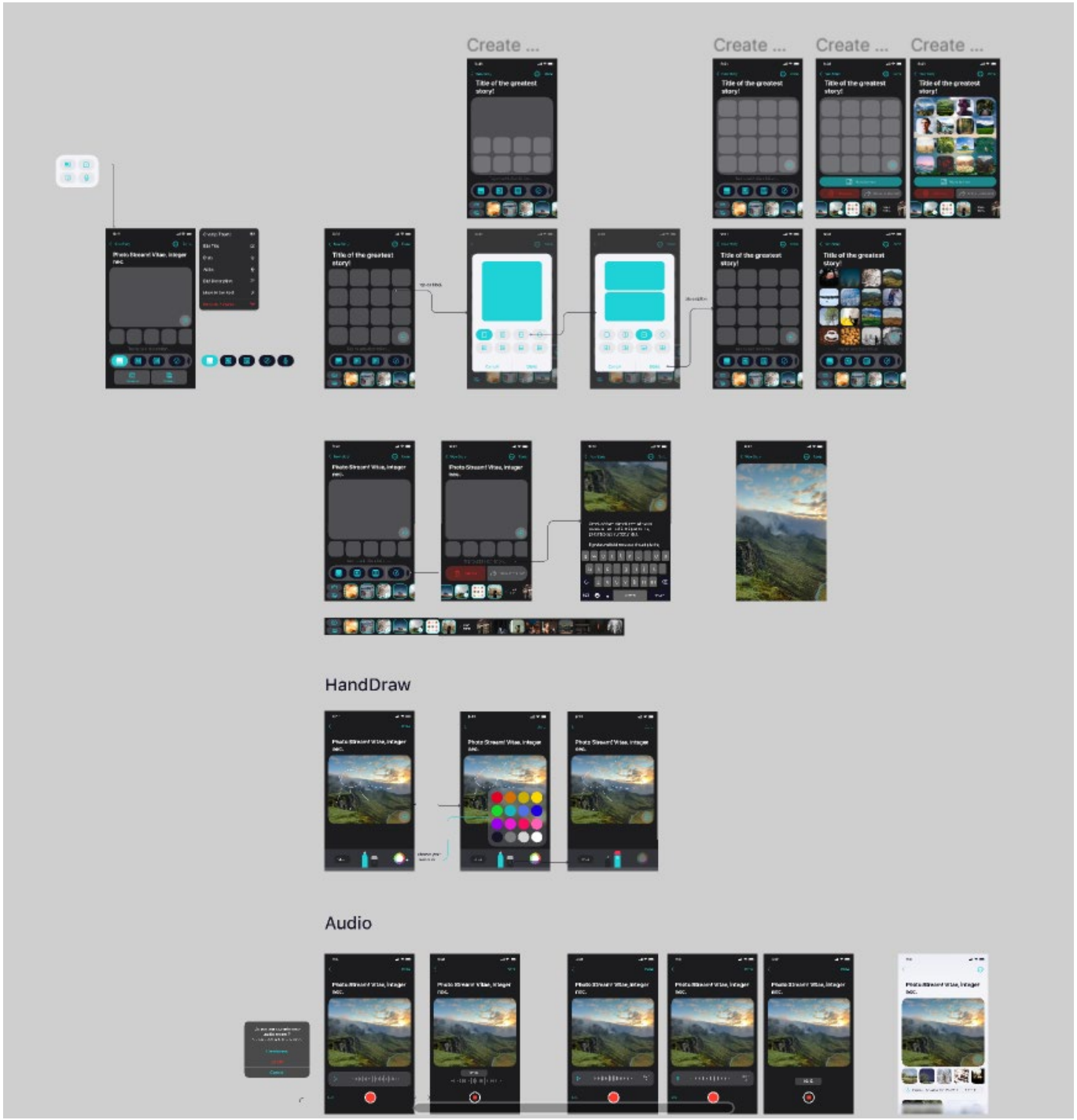


Рисунок 2.11 – Дизайн екрану редагування

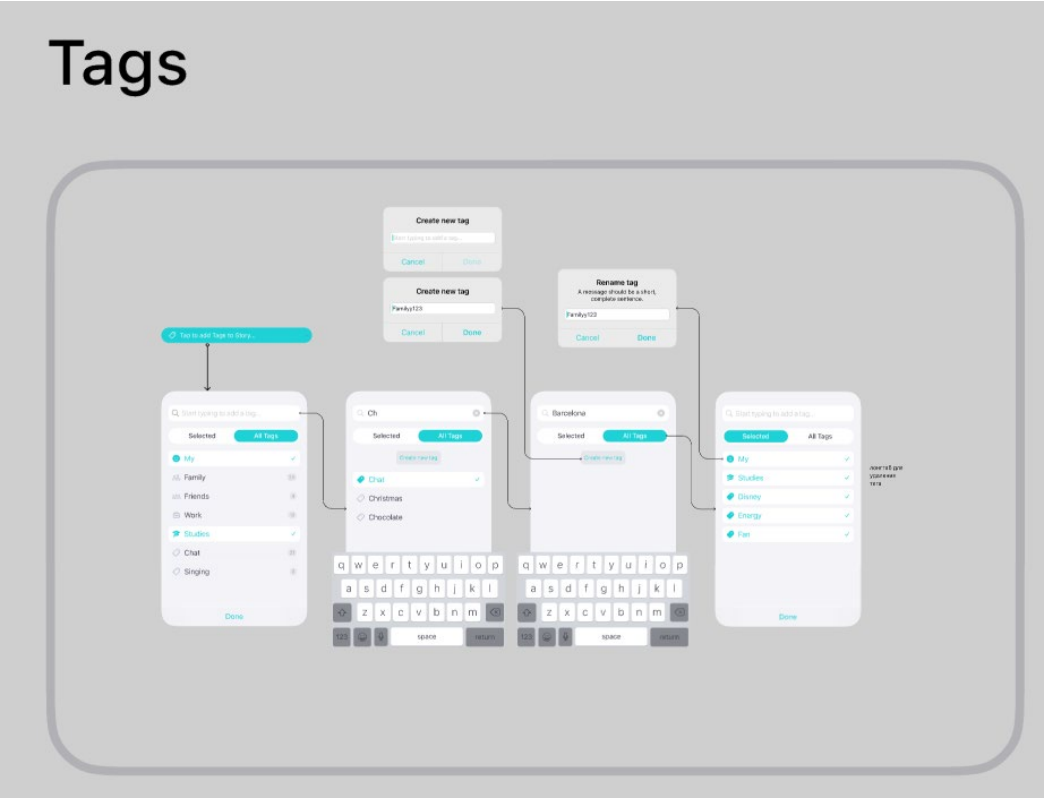


Рисунок 2.12 – Дизайн вікна створення тагів

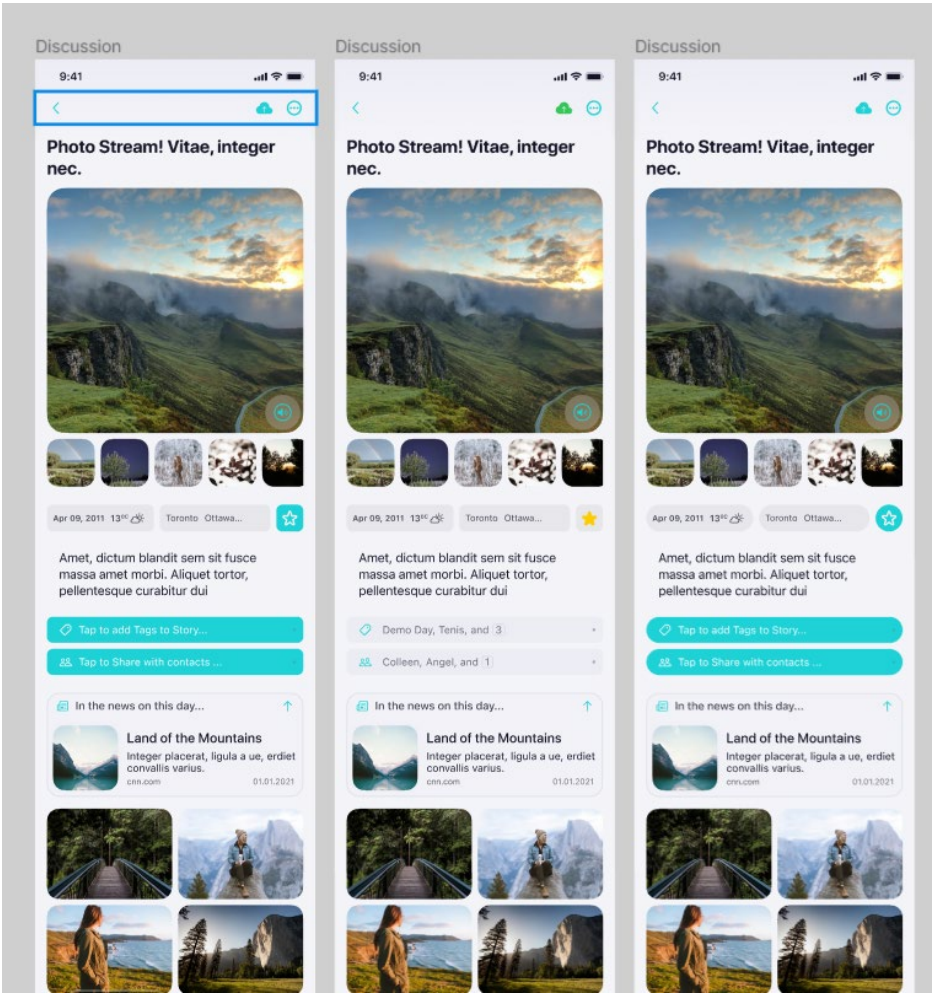


Рисунок 2.13 – Дизайн екрану повного перегляду спогаду

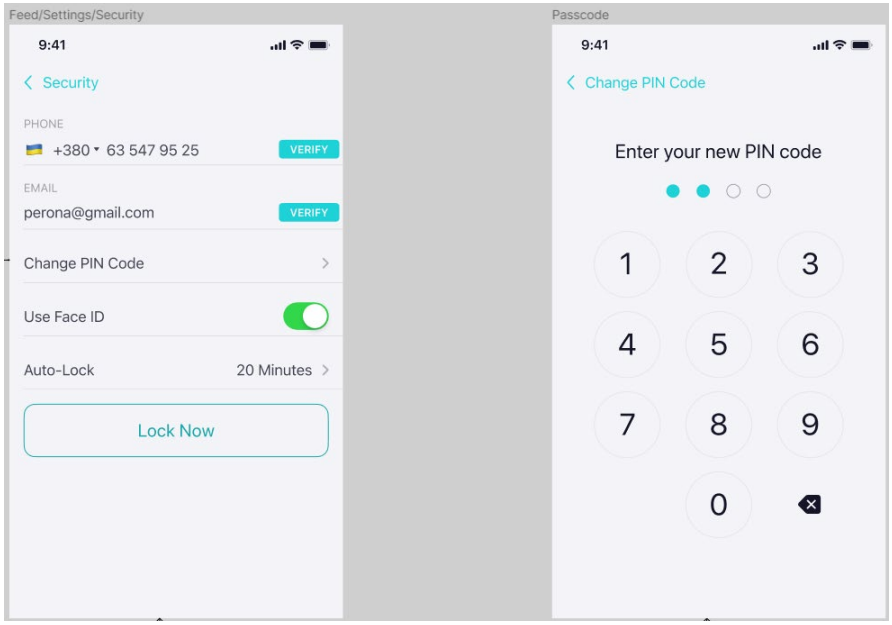


Рисунок 2.13 – Дизайн екранів верифікації електронної пошти та телефону та екрану введення пін коду для входу в додаток

Таким чином було розроблено інтерфейс мобільного додатку, та можна приступати до його програмної реалізації.

2.9 Висновки

1. Спроектовано архітектуру системи. Визначено мову програмування та програмні засоби за допомогою яких буде вестись розробка програмного засобу, а також визначено патерн програмування за допомогою якого буде створено ядро системи.

2. Проведено проектування алгоритму створення спогадів за допомогою аналізу зображень та агрегування медіа контенту. Визначено ключові параметри за якими буде проведено відбір медіа контенту, який попаде до вибірки спогадів, а також створено алгоритм створення назви спогаду за допомогою таких параметрів як, час створення, локація та знайдені об’єкти на зображенні.

3. Створено структуру бази даних для зберігання спогадів та подальшого створення резервної копії.

4. Спроектовано інтерфейс системи за допомогою програмного засобу Figma. Розроблено логотип, дизайн стартового, головного вікна, а також редагування та перегляду спогадів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз засобів реалізації програмного продукту

На основі проведеного у першому розділі аналізу методів розв’язання поставленої задачі було обрано створення мобільного додатку за допомогою програмного засобу Flutter.

На сьогоднішній день у розробників існує широкий вибір програмних засобів для реалізації створення мобільного додатку. Для вибору засобів реалізації магістерської кваліфікаційної роботи розглянемо найвідоміші способи реалізації мобільних додатків.

В першу чергу необхідно визначити, що однією з задач магістерської кваліфікаційної роботи є створення мобільного додатку, який може бути доступним для основних сучасних операційних систем, призначених для смартфонів, а саме iOS та Android. Тому для пришвидшення розробки було надано перевагу до кросплатформних програмних засобів, оскільки їх використання забезпечує використання однієї програмної бази, яку можливо скомпілювати під обидві платформи. Серед таких програмних засобів можна виділити наступні:

Kotlin Multiplatform

Технологія Kotlin Multiplatform розроблена для спрощення розробки кросплатформних проектів [20]. Це скорочує час, витрачений на написання та підтримку того самого коду для різних платформ, зберігаючи при цьому гнучкість і переваги нативного програмування.

Спільне використання коду між мобільними платформами є одним із основних варіантів використання Kotlin Multiplatform. За допомогою Kotlin Multiplatform Mobile ви можете створювати кросплатформні мобільні додатки та спільно використовувати спільний код між Android та iOS, наприклад бізнес-логіку, підключення тощо.

Загальний код, написаний на Kotlin, можна впровадити з мінімальними зусиллями у готовий додаток. Він компілюється у звичні для платформ бібліотеки. Для Android це jar чи aar-бібліотека, для iOS — Universal Framework. Підключення та

подальша робота не сильно відрізняються від взаємодії з будь-якою нативною бібліотекою. Синтаксис мови Kotlin близький до Swift. Схожість мов знижує поріг входу для iOS-розробників. Обидві мови поділяють схожу ідеологію – швидкість та зручність роботи для розробника. Зрозуміти, що відбувається у загальному коді, і доповнити його зможе будь-хто в команді.

Недоліком даного підходу є його відносно не великий час на ринку, таким чином підхід має не велику кількість підтримки спільноти, а також доволі велику кількість помилок, оскільки він ще не встиг протестуватись спільнотою як слід.

React Native

React Native (також відомий як RN) — це популярна платформа для мобільних додатків на основі JavaScript, яка дозволяє створювати нативно відтворені мобільні додатки для iOS та Android [21]. Фреймворк дозволяє створювати програми для різних платформ, використовуючи ту саму кодову базу.

React Native був вперше випущений Facebook як проект з відкритим вихідним кодом у 2015 році. Всього за пару років він став одним із найкращих рішень для мобільної розробки. Розробка React Native використовується для роботи деяких провідних у світі мобільних програм, зокрема Instagram, Facebook і Skype. Ми обговорюємо ці та інші приклади додатків на базі React Native далі в цій публікації.

Є кілька причин глобального успіху React Native.

По-перше, використовуючи React Native, компанії можуть створити код лише один раз і використовувати його для роботи своїх додатків для iOS і Android. Це означає величезну економію часу та ресурсів.

По-друге, React Native було створено на основі React – бібліотеки JavaScript, яка вже була надзвичайно популярною на момент випуску мобільного фреймворку. Ми детально обговорюємо відмінності між React і React Native далі в цьому розділі.

По-третє, фреймворк дав змогу розробникам інтерфейсу, які раніше могли працювати лише з веб-технологіями, створювати надійні, готові до виробництва програми для мобільних платформ.

Цікаво, що, як і багато інших революційних винаходів, React Native був розроблений як відповідь на... велику технологічну помилку.

React Native був успішно прийнятий сотнями компаній у всьому світі, включаючи Uber, Microsoft і Facebook, і використовується в багатьох галузях.

React Native дозволяє веб-розробникам створювати мобільні програми, які виглядають і відчуваються «рідними», не виходячи з бібліотеки JavaScript (JS), React. Ця простота використання зробила React Native одним із найпопулярніших мобільних фреймворків, але це також є причиною деяких внутрішніх проблем із продуктивністю.

Flutter

Flutter — це набір програмного забезпечення для розробки інтерфейсу користувача з відкритим вихідним кодом, створений Google [22]. Він використовується для розробки кросплатформних додатків для Android, iOS, Linux, macOS, Windows, Google Fuchsia та Інтернету з єдиної кодової бази. Вперше анонсований у 2015 році, Flutter був випущений у травні 2017 року.

Перша версія Flutter була відома як "Sky" і працювала на операційній системі Android. Він був представлений на саміті розробників Dart у 2015 році з заявленим наміром мати можливість постійно відтворювати зі швидкістю 120 кадрів в секунду. Під час основної доповіді Google Developer Days у Шанхаї у вересні 2018 року Google оголосила Flutter Release Preview 2, останній великий випуск перед Flutter 1.0. 4 грудня того ж року на заході Flutter Live був випущений Flutter 1.0, що означає першу стабільну версію фреймворку. 11 грудня 2019 року на заході Flutter Interactive був випущений Flutter 1.12.

6 травня 2020 року було випущено набір для розробки програмного забезпечення Dart версії 2.8 і Flutter 1.17.0, додавши підтримку Metal API, який покращує продуктивність на пристроях iOS приблизно на 50%, а також нові віджети Material і мережеве відстеження. інструменти розробки.

3 березня 2021 року Google випустив Flutter 2 під час онлайн-заходу Flutter Engage. Це велике оновлення принесло офіційну підтримку веб-додатків із новим рендерером CanvasKit та веб-віджетами, підтримкою настільних програм раннього доступу для Windows, macOS та Linux та покращеними API-інтерфейсами Add-to-App. У цій версії також використовувався Dart 2.0, який забезпечував звукову нуль-

безпеку, що спричинило багато критичних змін і проблем із багатьма зовнішніми пакетами; однак команда Flutter включила інструкції та інструменти для пом’якшення цих проблем.

8 вересня 2021 року Google випустила Dart 2.14 і Flutter 2.5. Оновлення принесло покращення в повноекранний режим Android і останню версію Material Design від Google під назвою Material You. Dart отримав два нових оновлення, стандартизуючи умови ворсинок і позначаючи підтримку Apple Silicon як стабільну.

Поточна стабільна версія Flutter — 3.3.9, а версія Dart — 2.18.5.

Основні компоненти Flutter включають:

– Dart платформа;
– Рушій Flutter (Skia Graphics Engine);
– Базова бібліотека;
– Спеціальні віджети для дизайну;
– Допоміжні засоби розробки Flutter DevTools.

Під час написання та налагодження програми Flutter працює у віртуальній машині Dart, яка має механізм виконання «якраз вчасно» (англ. “just in time”). Це забезпечує швидкий час компіляції, а також «гарячого перезавантаження»(англ. “hot reload”), за допомогою якого можна вносити зміни до вихідних файлів у запущену програму. Flutter розширює це ще більше за допомогою підтримки гарячого перезавантаження з урахуванням стану, коли в більшості випадків зміни вихідного коду відразу відображаються в запущеній програмі без необхідності перезавантаження або втрати стану.

Для кращої продуктивності можна випускати версії додатків Flutter на всіх платформах за допомогою завчасної компіляції.

Рушій Flutter, написаний переважно на C++, забезпечує низькорівневу підтримку візуалізації за допомогою графічної бібліотеки Skia від Google. Крім того, він взаємодіє з пакетами SDK для певної платформи, такими як Android та iOS. Flutter Engine — це портативна програма для розміщення програм Flutter. Він реалізує основні бібліотеки Flutter, включаючи анімацію та графіку, файловий і мережевий ввід-вивід, підтримку доступності, архітектуру плагінів, а також ланцюг інструментів

Dart для виконання та компіляції. Більшість розробників взаємодіють з Flutter через Flutter Framework, яка забезпечує реактивну структуру та набір віджетів платформи, макета та основи.

Дизайн інтерфейсу користувача у Flutter будують з віджетів. Віджет у Flutter являє собою незмінний об'єкт, який описує частину інтерфейсу користувача. Вся графіка, текст, фігури та анімації створюють за допомогою віджетів. Складні віджети створюють шляхом об'єднання простих. На поточний час Flutter містить два набори віджетів, які відповідають відповідним принципам побудови:

– віджети Material Design використовують дизайн Google;
– віджети Cupertino імітують дизайн Apple iOS.

Як було описано раніше, Flutter використовує Dart як основну мову програмування для написання програмного коду додатку.

Dart — мова програмування, яку розробляє компанія Google, позиціонуючи як мову структурованого програмування для Веб [23]. Розробники вважали, що в довгостроковій перспективі Dart може стати прогресивною заміною JavaScript, котрий потерпає від наявних в даний час проблем з розширюваністю, продуктивністю і підтримкою розробки складних застосунків. Мова має схожий на Java синтаксис, не вимагає явного визначення типів і її можна використовувати для створення серверних та клієнтських застосунків.

Мова має схожий на Java синтаксис, не вимагає явного визначення типів і може використовуватися для створення серверних і клієнтських застосунків. Для запуску всередині браузера код мовою Dart може бути перетворений в JavaScript-подання або запущений безпосередньо під управлінням спеціального JavaScript-інтерпретатора Dartboard. Підтримується вбудовування коду мовою Dart в HTML-сторінки, використовуючи MIME тип "application/dart". На стороні сервера застосунок на мові Dart може бути виконаний всередині спеціальної віртуальної машини, яка забезпечує продуктивність виконання близьку до компільованих в машинний код мов. Віртуальну машину Dart планують інтегрувати в майбутні версії браузера Chrome, що дозволить виконувати застосунки мовою Dart без компіляції в JavaScript.

Мова підходить як для розробки одним програмістом невеликих скриптів без жорсткої структури, так і для створення високо масштабованих великих модульних проектів, підтримуваних великим колективом з потребою більш явної типізації для того, щоб уникнути плутанини і помилок. При цьому явне задання типів не обов'язкове, наприклад, можна почати розробку без вказання типів, а надалі при необхідності додати їх (наприклад, спочатку написати "var x", а потім замінити на "num x"). Код Dart завжди виконується тільки в рамках одної потоку, для організації паралельного виконання пропонується використовувати класи з атрибутом isolate. У кожному скрипті використовується власний простір імен, для використання зовнішніх об'єктів, функцій або змінних слід їх явно імпортувати за допомогою конструкції "import". Всі змінні, початково, діють тільки в межах поточного скрипту і не експортуються глобально.

Для спрощення розробки мовою Dart поставляється SDK, який включає в себе компілятор dart2js, віртуальну машину Dart VM, пакетний менеджер pub, статичний аналізатор коду dart_analyzer, і набір бібліотек. Для виконання і відлагодження застосунків на мові Dart, без компіляції в JavaScript, поширюється Dartium — складання браузера Chromium з інтегрованою віртуальною машиною Dart VM.

Завдяки вибору Flutter процес розробки для мобільної платформи значно спрощується. На публічному ресурсі бібліотек «pub.dev» завжди можна знайти корисну бібліотеку чи пакунок який допоможе вирішити різноманітні проблеми пов’язані з розробкою даного застосунку.

Після виконання аналізу засобів реалізації програмного додатку, розглянувши усі можливі рішення було здійснено висновок, що для розробки даного програмного застосунку найкращим варіантом для розробки є Flutter. Причиною для цього є доволі низький поріг входу в розробку, можливість підключення нативних бібліотек до проєкту, швидкість розробки, а також синтаксис мови програмування Dart, яка є більш зручною для розробника ніж інші наведені мови програмування, що використовуються в Kotlin Multiplatform або React Native. Проте головною причиною вибору саме Flutter є його висока продуктивність за рахунок компіляції в нативні мови програмування Kotlin для Android та Swift для iOS, для прикладу на рисунку 3.1

зображено тест швидкості роботи Flutter та React Native, де Flutter в декілька разів швидше виконує операції ніж React Native, при цьому працюючи не сильно повільніше ніж нативний код [24].

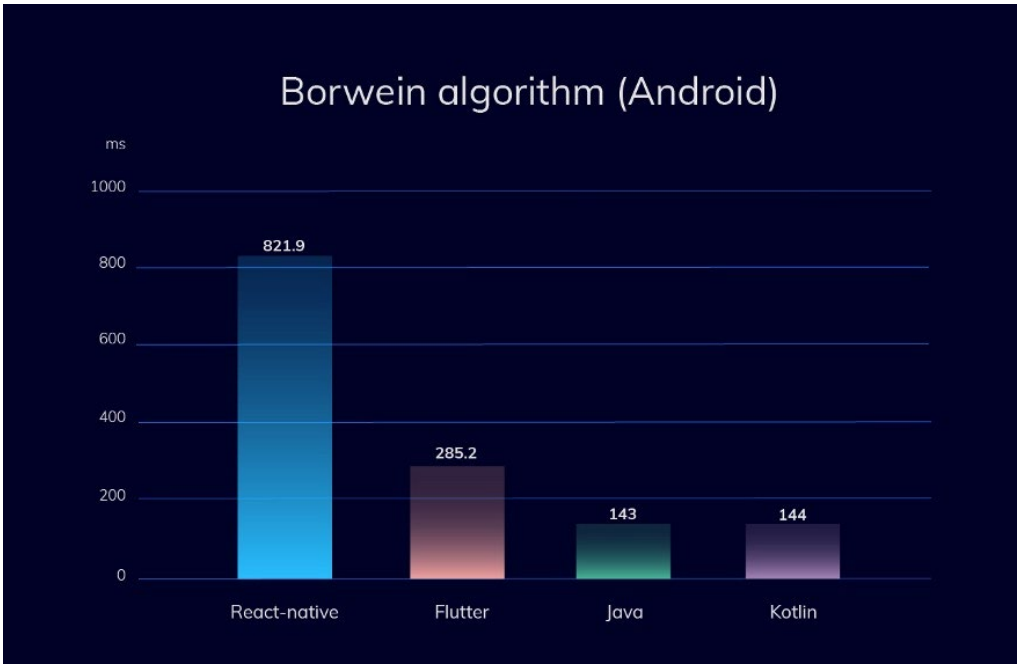


Рисунок 3.1 – Порівняння швидкості роботи Flutter та React Native

3.2 Аналіз середовища розробки

В попередньому підрозділі було визначено, що для реалізації програмного забезпечення було обрано фреймворк Flutter. Для роботи з кодовою базою та розробки програмного забезпечення можна використовувати такі середовища розробки, як Visual Studio Code та Android Studio.

Visual Studio Code – це засіб для створення, редагування та зневадження сучасних веб застосунків і програм для хмарних систем [25]. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

За основу для Visual Studio Code використовуються напрацювання вільного проєкту Atom, що розвивається компанією GitHub [26]. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовує браузерний рушій Chromium і Node.js.

Переваги:

– зручність роботи при умові досвіду роботи в IDE Visual Studio;

– низькі системні вимоги;

– підтримка плагінів Flutter.

Недоліки:

– низька кількість додаткового функціоналу для відлагодження.

Android Studio — це офіційне інтегроване середовище розробки (IDE) для розробки додатків Android [27]. Він заснований на IntelliJ IDEA, інтегрованому середовищі розробки Java для програмного забезпечення, і містить його інструменти для редагування коду та розробника.

Для підтримки розробки додатків в операційній системі Android Android Studio використовує систему збірки на основі Gradle, емулятор, шаблони коду та інтеграцію з Github. Кожен проект в Android Studio має одну або кілька модальностей із вихідним кодом і файлами ресурсів. Ці модальності включають модулі програми Android, модулі бібліотеки та модулі Google App Engine.

Android Studio використовує функцію Instant Push для надсилання коду та змін ресурсів у запущену програму. Редактор коду допомагає розробнику писати код і пропонує доповнення, заломлення та аналіз коду. Програми, створені в Android Studio, потім компілюються у формат APK для надсилання в магазин Google Play.

Програмне забезпечення було вперше оголошено на Google I/O у травні 2013 року [28], а перша стабільна збірка була випущена в грудні 2014 року. Android Studio доступний для настільних платформ Mac, Windows і Linux. Він замінив Eclipse Android Development Tools (ADT) як основну IDE для розробки додатків Android. Android Studio та пакет розробки програмного забезпечення можна завантажити безпосередньо з Google. Головне вікно середовища Android Studio зображено на рисунку 3.1.

Головною перевагою Android Studio є велика кількість можливостей для відлагодження додатків. Зокрема Flutter Inspector, Device Inspector, Logcat, Debug Menu. Перелічені можливості дозволяють швидко виправляти помилки.

Не зважаючи на доволі високі системні вимоги Android Studio порівняно з Visual Studio Code було обрано саме Android Studio як головне середовище розробки програмного забезпечення, оскільки підтримка великої кількості плагінів для Flutter

та можливостей відлагодження, створення емуляторів дозволяють розробляти продукт якісніше та швидше.

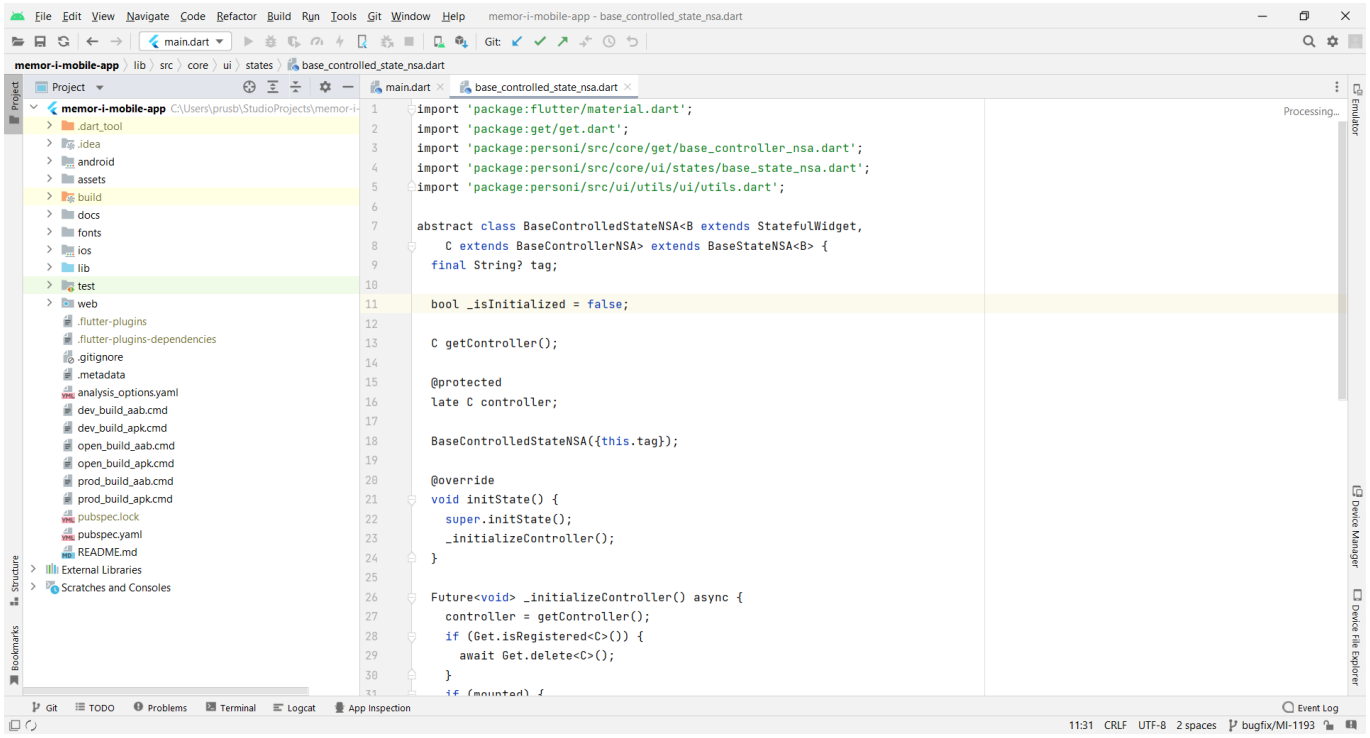


Рисунок 3.2 – Головне вікно середовища Android Studio

Оскільки розробка для операційної системи iOS дозволяється лише на пристроях, що працюють на операційній системі MacOS, то для розробки можна використовувати лише середовище Xcode [29].

Xcode — інтегроване середовище розробки (IDE) виробництва Apple. Дозволяє створювати програмне забезпечення з використанням таких технологій як GCC, GDB, Java та ін. На сьогодні є єдиним засобом написання «універсальних» (Universal Binary) прикладних програм для Mac OS X.

Xcode включає в себе більшу частину документації розробника від Apple та Interface Builder — застосунок, який використовується для створення графічних інтерфейсів. Інтерфейс інтерактивного середовища Xcode зображено на рисунку 3.3.

Таким чином для розробки програмного забезпечення було обрано середовище розробки Android Studio для роботи безпосередньо з кодовою базою програмного продукту на мові Dart та розробки застосунку для операційної системи Android, а

також для розробки застосунку для операційної системи iOS буде використано середовище розробки Xcode.

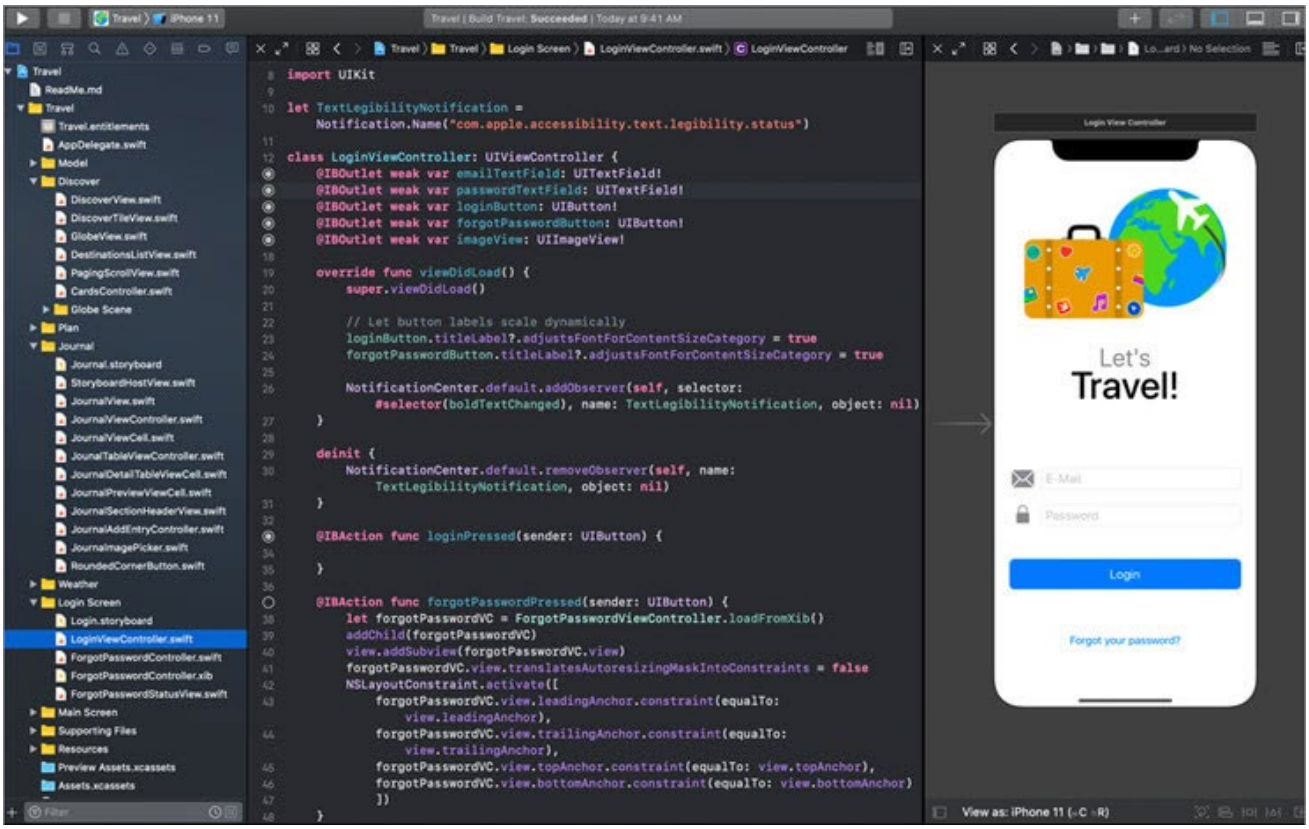


Рисунок 3.3 – Головне вікно середовища XCode

3.3 Програмна реалізація мобільного додатку

Програмну реалізацію необхідно розпочати з завантаження Flutter Framework з офіційного сайту, для розробки було обрано версію 3.0.5 як актуальну версію на момент початку розробки. Розпакувавши архів з даними фреймворку, та прописавши необхідні команди для встановлення фреймворку згідно інструкцій можна приступати до створення проекту в середовищі Android Studio. Для цього необхідно відкрити середовище розробки та перейти на вкладку New Project -> New Flutter Project, що зображено на рисунку 3.3, та обрати необхідні платформи, під які буде розроблятись даний проект. В нашому випадку необхідно обрати Android та iOS. Та натиснути далі, таким чином буде створено проект з програмним кодом.

Наступним етапом необхідно реалізувати підключення проекту до Firebase. Firebase — це Backend-as-a-Service (Baas). Він надає розробникам різноманітні

інструменти та послуги, які допомагають їм розробляти якісні програми, розширювати базу користувачів і отримувати прибуток. Він побудований на інфраструктурі Google. Firebase класифікується як програма бази даних NoSQL, яка зберігає дані в JSON-подібних документах [30]. Для цього створимо проект Firebase, та додамо додаток до створеного Firebase проєкту – рисунок 3.4.

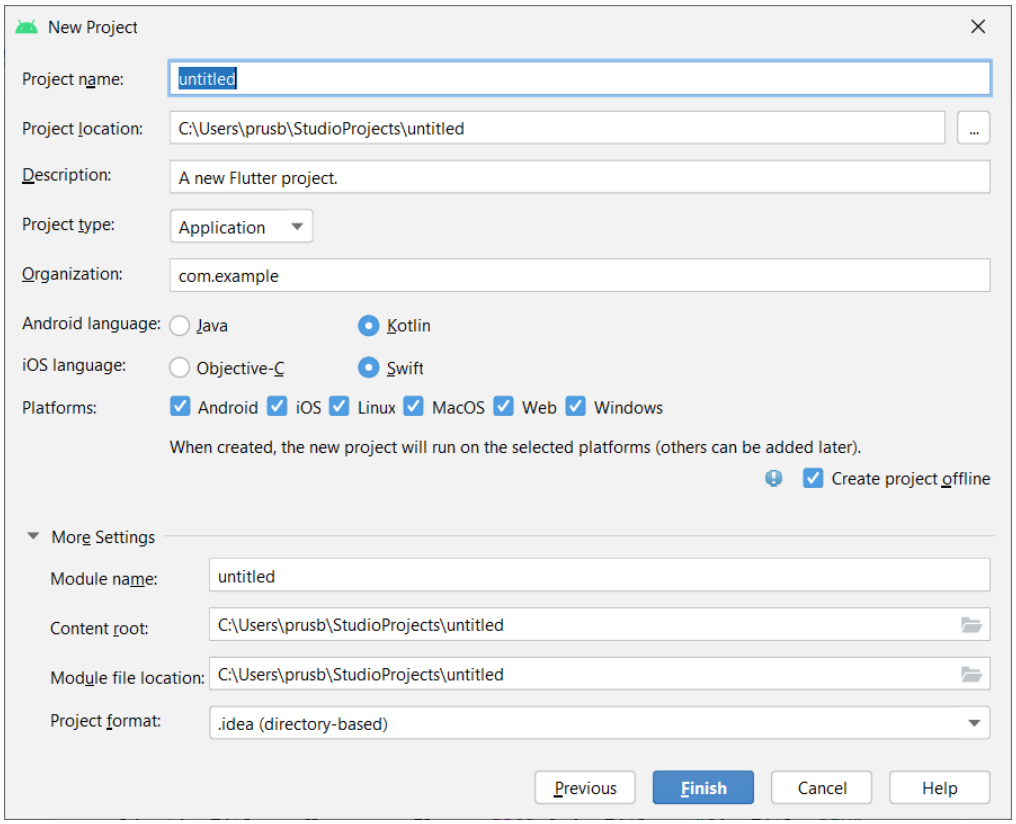


Рисунок 3.3 – Створення проекту в середовищі Android Studio

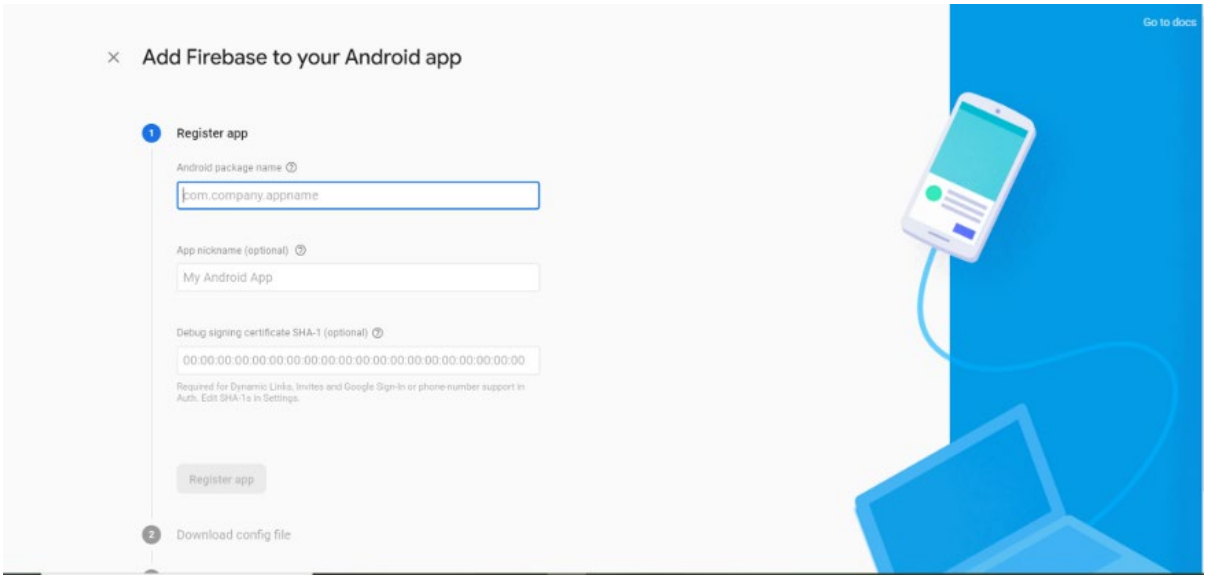


Рисунок 3.4 – Додавання Android додатку проекту Firebase

Далі, відповідно інструкцій Firebase, необхідно створити підключення програмного коду до серверів Firebase. Для цього, на сторінці pub.dev знаходимо бібліотеки firebase core та firebase auth, що буде мінімально необхідно для створення авторизації через firebase. Для їх підключення використовуємо файл проекту pubspec.yaml, в якому знаходяться налаштування проекту, а також всі його залежності. Додамо зазначені бібліотеки до списку залежностей – рисунок 3.5.

```
#firebase
firebase_storage: 11.0.6
firebase_analytics: 10.0.6
firebase_core: 2.3.0
firebase_crashlytics: 3.0.6
firebase_auth: 4.1.3
firebase_messaging: 14.1.1
firebase_dynamic_links: 5.0.6
```

Рисунок 3.5 – Підключення бібліотек firebase до залежностей проєкту

Наступним етапом створимо програмний модуль, для реалізації авторизації. Для цього створимо абстрактний клас BaseAuth в якому створимо логіку роботи з Firebase Auth та імплементацію для нього у класі Auth.

```
class Auth implements BaseAuth {
  final FirebaseAuth _firebaseAuth = FirebaseAuth.instance;
  Future<String> signIn(String email, String password) async {
    FirebaseUser user = await _firebaseAuth.signInWithEmailAndPassword(email: email, password:
password);
    return user.uid;
  }
  Future<String> signUp(String email, String password) async {
    FirebaseUser user = await _firebaseAuth.createUserWithEmailAndPassword(
        email: email, password: password);
    return user.uid;
  }
  Future<FirebaseUser> getCurrentUser() async {
```

```
    FirebaseUser user = await _firebaseAuth.currentUser();
    return user;
  }
  Future<void> signOut() async {
    return _firebaseAuth.signOut();
  }
  Future<void> sendEmailVerification() async {
    FirebaseUser user = await _firebaseAuth.currentUser();
    user.sendEmailVerification();
  }
  Future<bool> isEmailVerified() async {
    FirebaseUser user = await _firebaseAuth.currentUser();
    return user.isEmailVerified;
  }
  @override
  Future<void> changePassword(String password) async {
    FirebaseUser user = await _firebaseAuth.currentUser();
    user.updatePassword(password).then((_) {
      print("Succesfull changed password");
    }).catchError((error) {
      print("Password can't be changed" + error.toString());
    });
    return null;
  }
}
```

В даному методі реалізовані такі методи, як:

– signIn – дозволяє виконувати авторизацію за допомогою електронної пошти та паролю;

– signUp – дозволяє виконувати реєстрацію через Firebase Auth за допомогою електронної пошти та паролю;

– getCurrentUser – дозволяє отримати модель конкретного користувача, використовується для перевірки чи користувач авторизований у системі, або для отримання авторизаційних даних;

– signOut – дозволяє вийти з аккаунту;

– sendEmailVerification – виконується для підтвердження електронної пошти;

– isEmailVerified – виконується для перевірки чи була підтверджена електронна пошта користувача;

– changePassword – виконується для зміни паролю аккаунту.

Слід зазначити, що після стартового екрану, коли користувач вперше відкриває додаток і переходить на головний екран, виконується так звана анонімна реєстрація аккаунту. Це необхідно для того, щоб була можливість ідентифікувати користувача у системі, проте на даному етапі, поки користувач не здійснює жодних дій з програмою, в тому числі і створення резервної копії. Додавання електронної адреси до аккаунту буде виконуватись вже після дій користувача в додатку, таких як автоматична агрегація спогадів, або створення їх власноруч.

Для цього необхідно налаштувати проєкт Firebase у консолі. Перейдемо до вкладки «Authentication», та на вкладці «Settings» необхідно увімкнути можливість об’єднання аккаунтів – рисунок 3.6.

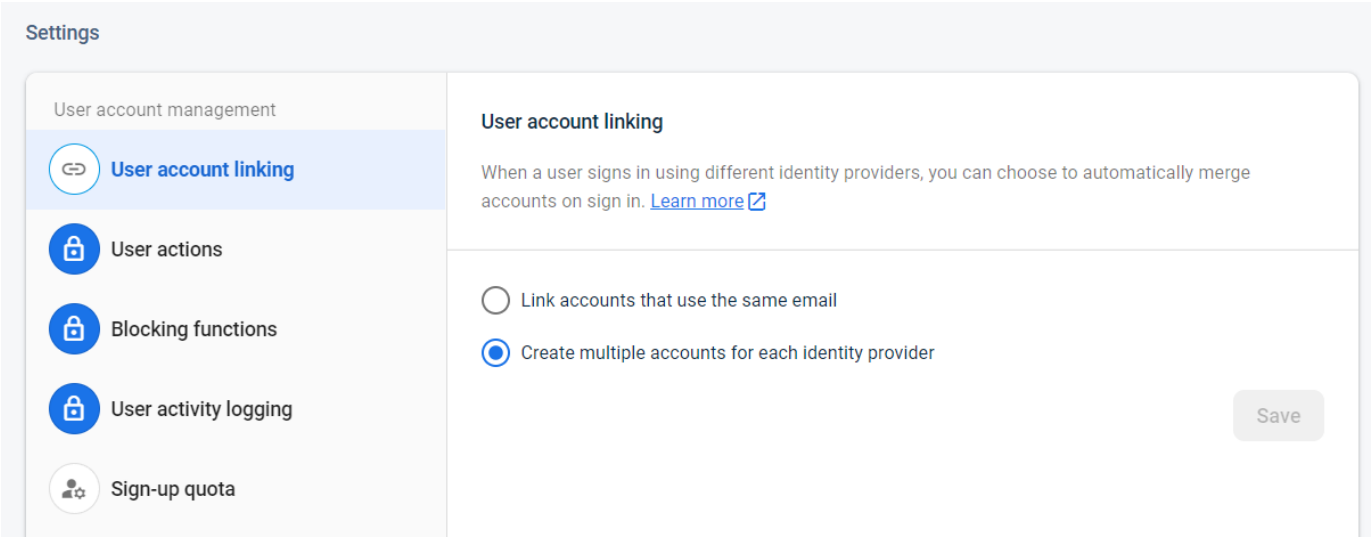


Рисунок 3.6 – Налаштування об’єднання аккаунтів у консолі Firebase

На даному етапі ми маємо підключену авторизацію через Firebase, що допоможе нам ідентифікувати користувача, та реалізувати наступні функції програмного забезпечення.

Далі можемо переходити до реалізації ядра системи. Для цього буде створено клас CoreApp, який буде викликатись при запуску додатку, та в ньому буде рендеретись інтерфейс та реалізована навігація.

Далі, згідно дизайну створимо та реалізуємо інтерфейс відповідних екранів, таких як:

– Main View;
– Story View;
– Story Editor View;
– Audio Editor View;
– Canvas Editor View;
– Tags View;
– Map View;
– Splash View;
– SignUp View.

Описані вище екрани будуть імплементацією базового екрану:

```
abstract class BaseControlledStateNSA<B extends StatefulWidget,
   C extends BaseControllerNSA> extends BaseStateNSA<B> {
 C getController();
 @protected
 late C controller;
 BaseControlledStateNSA({this.tag});
 Future<void> _initializeController() async {
  controller = getController();
  if (Get.isRegistered<C>()) {
   await Get.delete<C>();
  }
  if (mounted) {
   setState(() {
    Get.put<C>(controller, tag: tag);
    _isInitialized = true;
   });
  }
 }
 @override
 Widget build(BuildContext context) {
  return Scaffold(
        body: getProgress(background: false),
       );
 }
 @override
 void dispose() {
```

```
    controller.dispose();
    Get.delete<C>(tag: tag);
    super.dispose();
  }
}
```

Даний клас реалізує поведінку шаблону MVC, який було обрано в якості архітектурного шаблону програмного засобу. Метод getController повертає для класу представлення контролер, з якого представлення використовує дані, які необхідно показати на екран. За допомогою методу update() контролер може змінювати стан екрану.

Таким чином було реалізовано користувацький інтерфейс програмного продукту. Програмний код всіх вікон буде наведено у додатках. Реалізовані екрани зображено на рисунках 3.7 – 3.10.

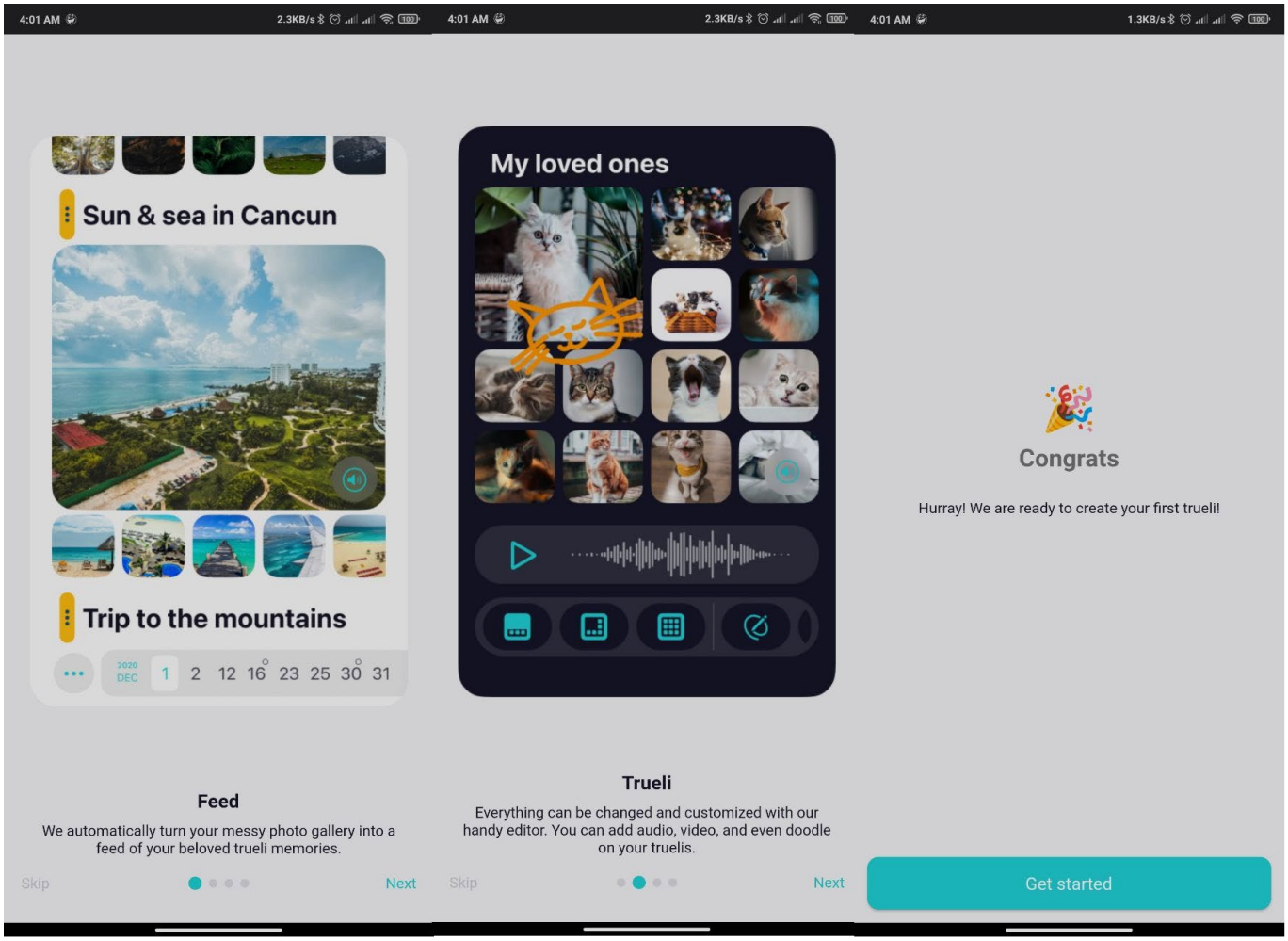


Рисунок 3.7 – Реалізація стартових екранів

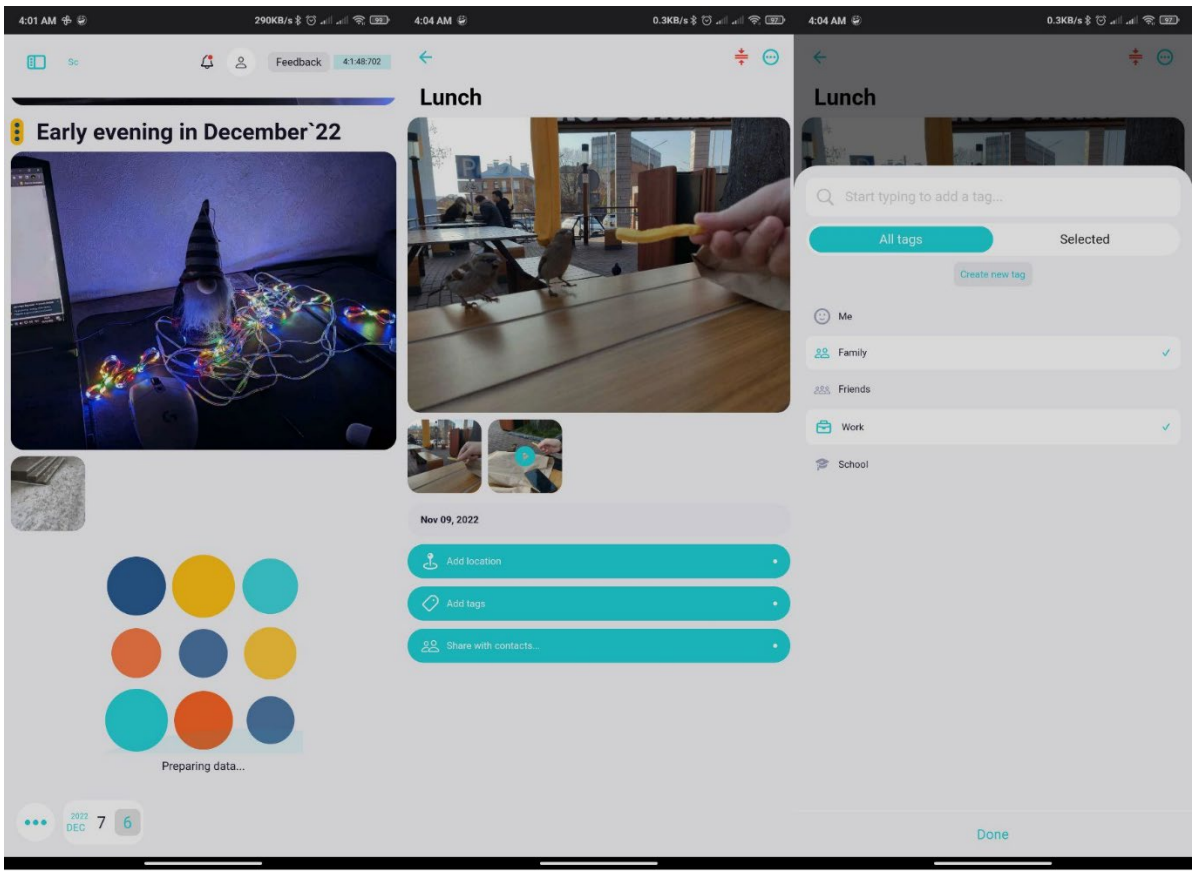


Рисунок 3.8 – Реалізація головного екрану та екрану перегляду спогаду

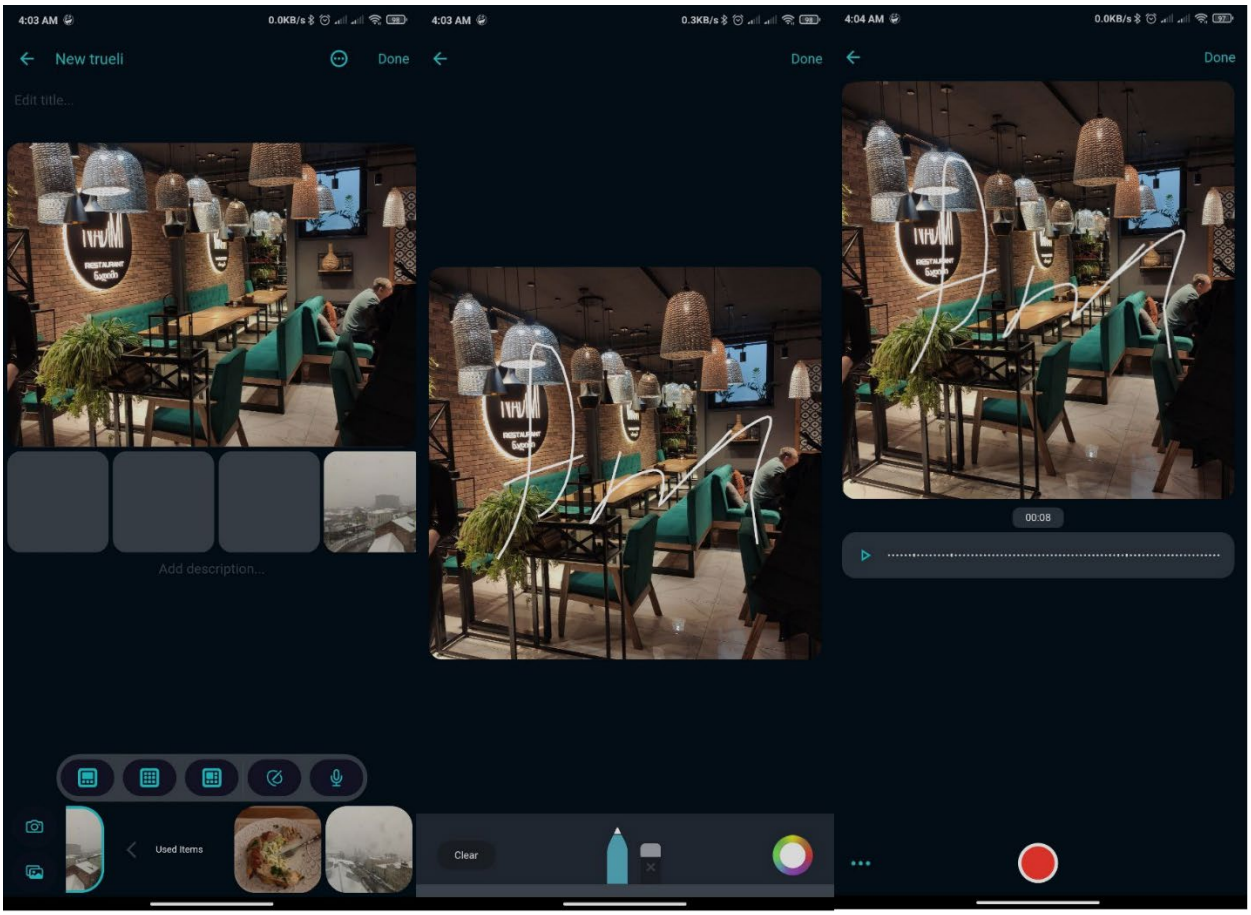


Рисунок 3.9 – Реалізація екрану редагування спогаду та суміжних з ним екранів

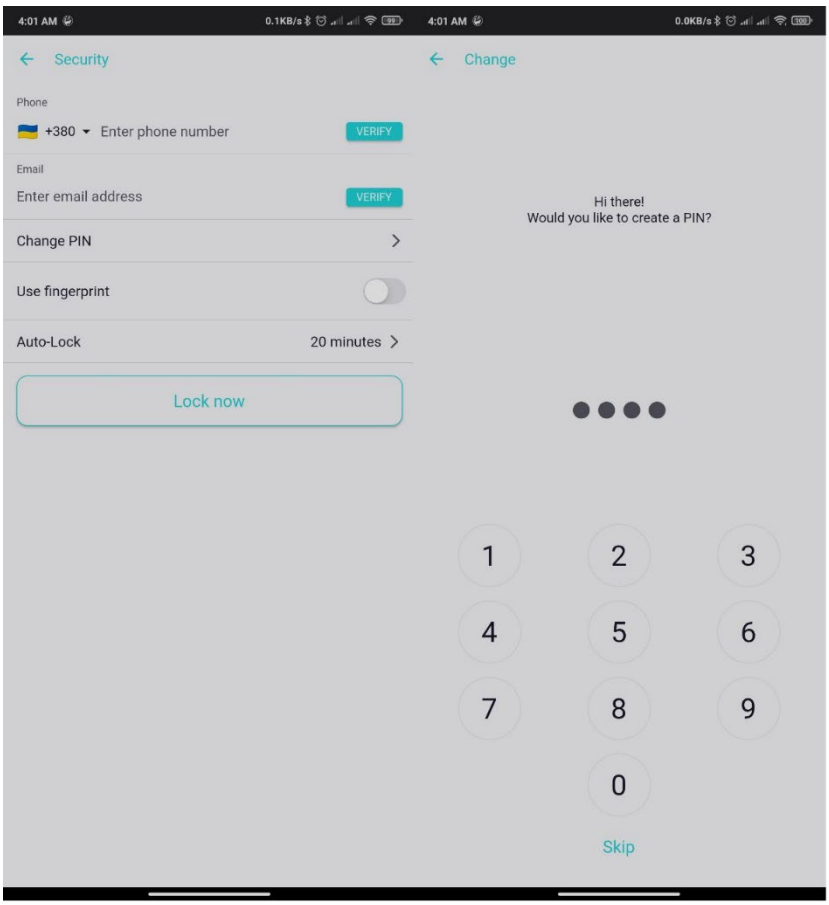


Рисунок 3.10 – Реалізація екранів верифікації електронної адреси та введення пін коду

3.4 Програмна реалізація алгоритму агрегування медіа контенту

Для агрегування медіа контенту відповідно алгоритму, який було описано у другому розділі створимо клас AutoCreateStoryManager. У ньому метод getAllAvailableContentAndStartCreate. У даному методі реалізуємо алгоритм роботи агрегування медіа контенту. Для початку дістаємо медіа контент з галереї пристрою за допомогою бібліотеки PhotoManager:

```
final List<AssetEntity> _content = await PhotoManagerService()
    .readAssetsFromStorage(startDate: startDate, endDate: endDate);
```

Визначивши список медіа контенту переходимо до його агрегування. Виконаємо фільтрування медіа за часом, відкинувши таким чином фотографії та відео які були створені серією, щоб уникнути можливого дубляжу:

```
contentModel =
    await ContentDateFilter().filterSameContent(contentModel);
```

Далі переходимо до створення моделі спогаду та обробки зображень за допомогою засобів розпізнавання зображень визначимо об’єкти на медіа контенті:

```
StoryModel? storyModel =
   await CreateStoryModelManager().createStoryModel(contentModel);
 storyModel = await ObjectsDetector().processImages(storyModel);
```

У методі processImages визначимо об’єкти та виконаємо фільтрування згідно таблиці 2.2.

```
final DetectedObject detectedObject = DetectedObject(
   recognitions[j]['detectedClass'],
   recognitions[j]['confidenceInClass']);
if (detectedObject.confidence! *
       ObjectThresholdMapper.thresholdCorrection >=
   ObjectThresholdMapper().objectThreshold[detectedObject.name]!) {
  detectedObjects.add(detectedObject);
}
```

Таким чином ми отримаємо список медіа файлів та список знайдених на них об’єктів. Далі за допомогою алгоритму генерації назви згенеруємо назву спогаду. Для цього використовуємо метадані медіа файлу, зокрема локацію та час. За допомогою geocoder визначаємо назву локації, а за допомогою часу створення медіа файлу та таблиці 2.1 визначаємо частину дня, яка буде використана для назви. Визначені об’єкти відносимо до категорій.

Реалізація створення назви спогаду:

```
Future<StoryModel> getTitle(StoryModel story) async {
  story.titleGeneratorDTO!.dateTime = story.mainContent!.dateCreateOriginal;
  final Locale locale = await PreferenceManager.getLocale();
  try {
    story.titleGeneratorDTO!.dateTime = story.mainContent!.dateCreateOriginal;
    if (story.locationEntity != null) {
      final Pair<String, String>? locationName =
          await _getTitleLocationName(story.locationEntity!);
      if (locationName != null) {
        story.titleGeneratorDTO!.locationName = locationName.right;
        story.titleGeneratorDTO!.preposition = locationName.left;
      }
    }
    final Pair<String, StoryTitleChunksEntity> titleResult =
        await StoryTitleGenerator(
      data: story.titleGeneratorDTO,
      langCode: locale.languageCode,
    ).getTitleForStory();
    story.name = titleResult.left;
```

```
    story.storyTitleChunksEntity = titleResult.right;
  } catch (err) {
    story.name = L.s.trueli;
  }
  return story;
}
```

Реалізація визначення назви локації для назви спогаду:

```
///Returns preposition and locationName
///['in', 'London']
///['at', 'home']
///[null, 'London']
///null if not found place name or city in location entity
Future<Pair<String, String>?> _getTitleLocationName(
    final LocationEntity location) async {
  String? locationName;
  if (location.placeName.isNotNullOrEmpty()) {
    locationName = location.placeName;
  } else if (location.city.isNotNullOrEmpty()) {
    locationName = location.city;
  }
  if (locationName == null) {
    return null;
  }
  final List<CustomLocationEntity> _locationsByName =
      await Get.find<LocationRepository>()
          .getCustomLocationsByAutogeneratedPlaceName(locationName);
  await LogService().logRealtime(
      "Locations: Searched ${_locationsByName.length} locations with autogenerated name
'$locationName'");

  ///Calculate distance between story location and location with same name
  for (final CustomLocationEntity element in _locationsByName) {
    if (element.calculateDistance(location.latitude!, location.longitude!)! <=
        Get.find<ConfigResponse>()
            .settings!
            .maxStoryTitleLocationRadiusMeters!) {
      return Pair<String, String>(element.preposition, element.locationName);
    }
  }
  return Pair<String, String>(null, locationName);
}
```

Під час створення спогаду у локальний файл logs.txt записуються усі дії алгоритму, для того щоб можна було відслідковувати роботу алгоритму, а також визначити можливі покращення його роботи. Процес створення спогаду за проміжком часу у вигляді файлу логу програми зображено на рисунку 3.11.

```
2022-09-21 21:27:19.948100 ================================2022-09-21 21:27:20.247006 ================================
2022-09-21 21:27:20.247009 Started story images processing
2022-09-21 21:27:20.247009 globalStoryId = 6DEC13EF-5009-42C2-B676-911F5302ED6A_b1c2597f-3f9b-4844-bfa5-e456363bc6af
2022-09-21 21:27:20.247010 story dates = 2022-03-19 19:03:31.000 -- 2022-03-19 18:28:01.000
2022-09-21 21:27:20.247013 story created at = 2022-09-21 21:27:20.246745
2022-09-21 21:27:20.247015 images count before processing = 4
2022-09-21 21:27:20.247016 -> started detection
2022-09-21 21:27:20.247018 -> -> got image = /var/mobile/Media/DCIM/108APPLE/IMG_8302.HEIC
2022-09-21 21:27:20.250717 -> -> -> image saved with milliseconds = 3
2022-09-21 21:27:20.264290 -> -> -> Tflite.detectObjectOnImage finished with milliseconds = 13 2022-09-21 21:27:20.264290 and returned 1 detected objects
2022-09-21 21:27:20.264313 -> -> -> -> added detected object = DetectedObject{name: tv, confidence: 0.5234375}
2022-09-21 21:27:20.264318 -> -> got image = /var/mobile/Media/DCIM/108APPLE/IMG_8301.HEIC
2022-09-21 21:27:20.268690 -> -> -> image saved with milliseconds = 4
2022-09-21 21:27:20.281197 -> -> -> Tflite.detectObjectOnImage finished with milliseconds = 12 2022-09-21 21:27:20.281197 and returned 1 detected objects
2022-09-21 21:27:20.281218 -> -> -> -> ignored DetectedObject{name: book, confidence: 0.57421875}
2022-09-21 21:27:20.281224 -> -> got image = /var/mobile/Media/DCIM/108APPLE/IMG_8300.HEIC
2022-09-21 21:27:20.284872 -> -> -> image saved with milliseconds = 3
2022-09-21 21:27:20.297353 -> -> -> Tflite.detectObjectOnImage finished with milliseconds = 12 2022-09-21 21:27:20.297353 and returned 2 detected objects
2022-09-21 21:27:20.297391 -> -> -> -> ignored DetectedObject{name: book, confidence: 0.66796875}
2022-09-21 21:27:20.297403 -> -> -> -> ignored DetectedObject{name: clock, confidence: 0.44921875}
2022-09-21 21:27:20.297405 -> -> got image = /var/mobile/Media/DCIM/108APPLE/IMG_8299.HEIC
2022-09-21 21:27:20.301164 -> -> -> image saved with milliseconds = 3
2022-09-21 21:27:20.313749 -> -> -> Tflite.detectObjectOnImage finished with milliseconds = 12 2022-09-21 21:27:20.313749 and returned 5 detected objects
2022-09-21 21:27:20.313773 -> -> -> -> ignored DetectedObject{name: refrigerator, confidence: 0.51171875}
2022-09-21 21:27:20.313785 -> -> -> -> added detected object = DetectedObject{name: dining table, confidence: 0.51171875}
2022-09-21 21:27:20.313794 -> -> -> -> ignored DetectedObject{name: chair, confidence: 0.4609375}
2022-09-21 21:27:20.313803 -> -> -> -> added detected object = DetectedObject{name: cup, confidence: 0.4140625}
2022-09-21 21:27:20.313811 -> -> -> -> ignored DetectedObject{name: oven, confidence: 0.3671875}
2022-09-21 21:27:20.313813 -> finished detection with millisecond = 66
2022-09-21 21:27:20.314265 -> excluded 2 images because of no detected objects on them
2022-09-21 21:27:20.314269 -> defined main image if could: ContentModel{assetId: E1490BBE-9F97-4692-94E0-0C482204C352/L0/001, , pathOrigin: /var/mobile/M
2022-09-21 21:27:20.314276 -> prepared TitleGeneratorDTO
2022-09-21 21:27:20.314277 Finished story images processing. Millisecond = 67
```

Рисунок 3.11 – Лог роботи алгоритму агрегування медіа контенту

Після створення моделі спогаду додаємо у неї посилання на медіа файли, та додаємо запис до таблиці StoryTable, а також на головний екран.

Далі переходимо у циклі до наступної вибірки зображень, і працюємо за алгоритмом, що описаний вище.

Програмний код класів для агрегування медіа контенту знаходиться у додатках.

3.5 Висновки

Таким чином у даному розділі було здійснено аналіз існуючих засобів реалізації програмного забезпечення. Розглянуто такі засоби, як Kotlin Multiplatform, React Native та Flutter. Після здійснення аналізу було обрано Flutter для реалізації мобільного додатку, оскільки він має низку критичних переваг, таких як швидкість розробки, поріг для входу у технологію, підтримка компіляції у нативний код та швидкість роботи програми.

Наступним кроком було здійснення аналізу середовищ розробки. Оскільки було обрано Flutter для реалізації програмного забезпечення, то вибір основного середовища відбувався між Visual Code та Android Studio. Після аналізу обох середових було здійснено висновок, що середовище Android Studio підходить для

здійснення розробки більше, оскільки має додаткові можливості для розробки, особливо для відлагодження програмного коду.

Для розробки програмного додатку для операційної системи iOS було обрано середовище Xcode, оскільки через політику компанії Apple дане середовище є безальтернативним, та не має аналогів.

Після здійснення аналізу існуючих засобів реалізації та середовищ розробки було розпочато роботу над програмною реалізацією проєкту. Для цього було спочатку створено користувацький інтерфейс та навігацію. Після чого була здійснена робота над створенням логіки кожного екрану.

Після створення користувацького інтерфейсу було здійснено програмну реалізацію алгоритму агрегування медіа контенту за допомогою засобів та методів розпізнавання зображень.

Таким чином було здійснено програмну реалізацію методів та засобів розпізнавання зображень для агрегування медіа контенту у вигляді мобільного додатку для платформ Android та iOS. Після завершення цього етапу необхідно приступити до тестування створеного додатку.

4 ТЕСТУВАННЯ РОБОТИ МОБІЛЬНОГО ДОДАТКУ

4.1 Опис методик, що використовуються для тестування системи

Тестування програмного забезпечення — процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, який здійснюють шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином [31].

При тестуванні можна оцінити:

– відповідність вимогам, якими керувалися проектувальники та розробники;

– правильність відповіді для всіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність із програмним забезпеченням та операційними системами;

– відповідність задачам замовника.

На сьогодні тестування програмного забезпечення – один з найбільш дорогих етапів життєвого циклу програмного забезпечення, на нього відводиться від 50% до 65% загальних витрат [31].

Розроблений застосунок було протестовано методом «чорної скриньки».

Тестування чорної скриньки - це підхід тестування, який використовується для перевірки функціональності на основі специфікацій без опирання на технології, що використовується для реалізації тестованої програми.

Під час тестування в чорній скриньці основне тестування стосуватиметься можливих входів та очікуваних результатів.

4.2 Проведення тестування мобільного додатку

Тестування екрану привітання:

1) Перегортання сторінок екрану привітання. Результат: програма дозволяє перегортувати сторінки привітання за допомогою жесту горизонтального руху.

2) Одноразове відображення екрану привітання. Результат: програма відображає екран привітання лише для незареєстрованого у системі користувача. При перезавантаженні додатку екран привітання не відображається.

Висновок: функціонал працює коректно.

Тестування автентифікації:

1) Введення коректного пін коду для входу. Результат: при введенні не коректного пін коду програма не дозволяє вхід у додаток – рисунок 4.1.

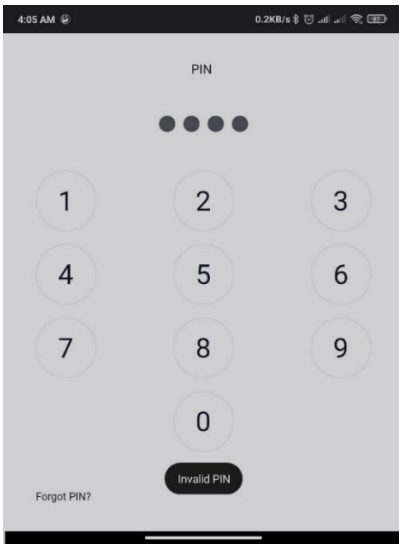


Рисунок 4.1 – Результат перевірки коректності пін коду

Тестування верифікації електронної адреси

1) Введення не коректної електронної адреси. Результат: програма видає користувачу помилку, якщо електронна адреса не валідна – рисунок 4.2.

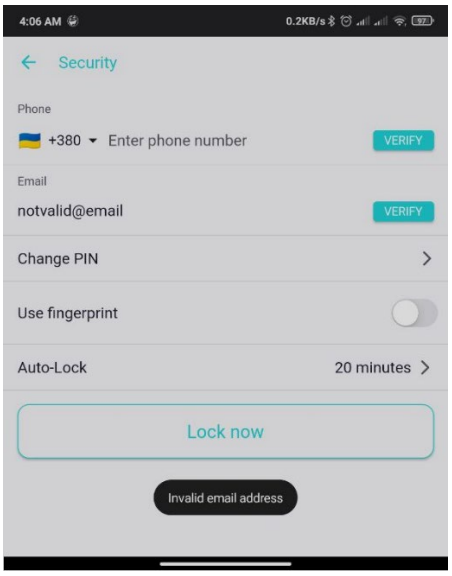


Рисунок 4.2 – Результат перевірки коректності електронної адреси

2) Введення електронної адреси, яка вже використовується. Результат: програма видає користувачу помилку, якщо електронна адреса використовується для іншого облікового записку – рисунок 4.3.

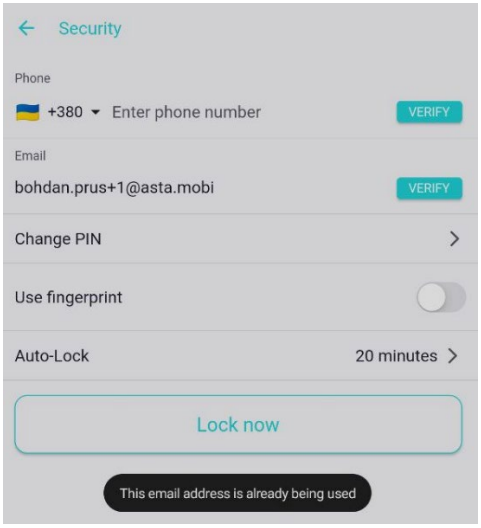


Рисунок 4.3 – Результат перевірки коректності введення коду верифікації

3) Введення не коректного коду підтвердження верифікації. Результат: при введення не коректного коду верифікації програма видає помилку – рисунок 4.4.

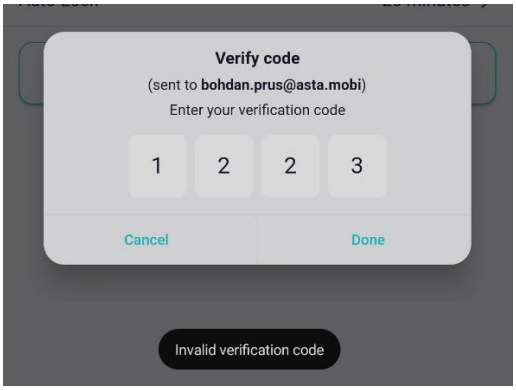


Рисунок 4.4 – Результат перевірки коректності введення коду верифікації

Висновок: функціонал працює коректно.

Тестування головного екрану

1) Агрегація медіа контенту. Результат: програма виконує сканування галереї мобільного пристрою, розбиває її на часові проміжки та виконує агрегацію медіа контенту у вигляді спогадів – рисунок 4.5.

2) Використання медіа контенту знятого на камеру мобільного телефону для агрегування. Результат: програма не використовує скріншоти, медіа файли з інших програм, записи екрану та завантажені фотографії під час агрегування медіа контенту.

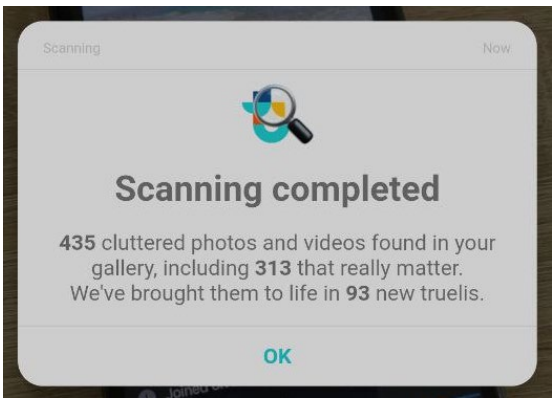


Рисунок 4.5 – Результат роботи алгоритму агрегування медіа контенту

3) Генерація назви спогаду. Результат: програма виконує аналіз часу зйомки, знайдених об’єктів та локації для генерації назви.

4) Повторна агрегація медіа контенту для нових фото та відео матеріалів. Результат: після закінчення першого проходу агрегування медіа контенту програма дозволяє виконати агрегацію для контенту, який ще не був використаний для агрегування.

5) Виділення дат у календарі. Результат: у календарі виділені дати, у яких є згенеровані або створені власноруч спогади.

6) Перехід до конкретної дати при натисканні. Результат: при натисканні на конкретну дату список спогадів прокручується до обраної дати.

7) Фільтрування спогадів по тегу. Результат: відображаються спогади, які відповідають обраним тегам.

8) Пошук спогадів по назві. Результат: при введенні пошукового запиту відображаються спогади, що відповідають йому.

9) Відображення спогадів на карті. Результат: на карті відображаються спогади, у яких є локація.

Висновок: функціонал працює коректно.

Тестування екрану перегляду спогадів

1) Відкривання екрану з обраним спогадом. Результат: при натисканні на спогад на головному екрані відбувається перехід на екран перегляду спогаду, де завантажується обраний спогад.

2) Додавання тегу. Результат: при натисканні на кнопку додавання тегу відкривається діалог з можливістю додавання або видалення тегу, де після

збереження теги додаються до спогаду і використовуються для фільтрування на головному екрані.

3) Відкривання медіа контенту на повний екран. Результат: при натисканні на віджет медіа контенту відбувається перехід на екран повного перегляду, у якому відбувається перегляд відео, збільшення фотографій, а також присутній слайдер для переходу до наступних фотографій.

Висновок: функціонал працює коректно.

Тестування екрану редагування спогадів

1) Редагування обраного спогаду. Результат: при натисканні на кнопку редагування спогаду на екрані перегляду спогаду відбувається перехід на екран редагування обраного спогаду.

2) Створення нового спогаду власноруч. Результат: при натисканні кнопки створення нового спогаду відбувається перехід на екран створення нового спогаду з пустим шаблоном.

3) Зміна типу шаблону. Результат: при натисканні на кнопку зміни шаблону відбувається його зміна на обраний тип – рисунок 4.6.

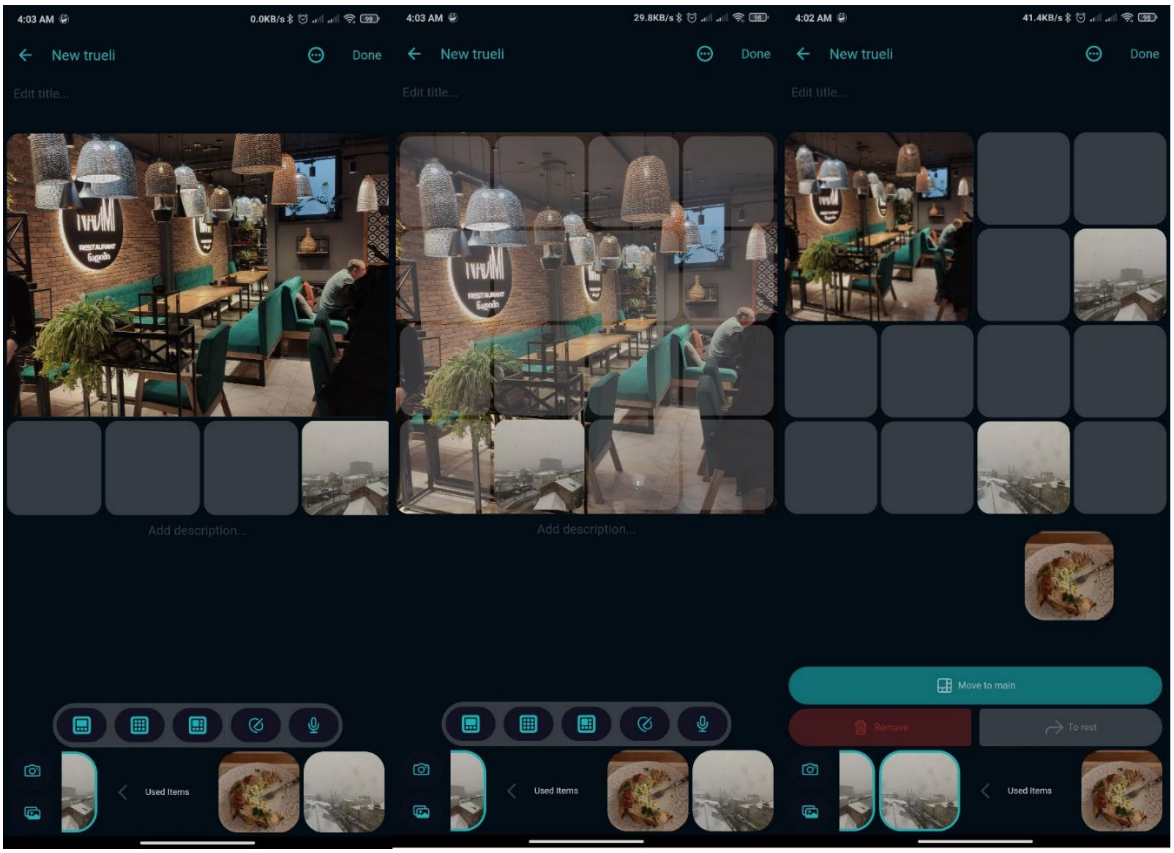


Рисунок 4.6 – Вибір типу підшаблону

4) Зміна типу підшаблону. Результат при довгому натисканні на підшаблон відкривається діалог з вибором типу підшаблону. При виборі типу відбувається його зміна – рисунок 4.7.

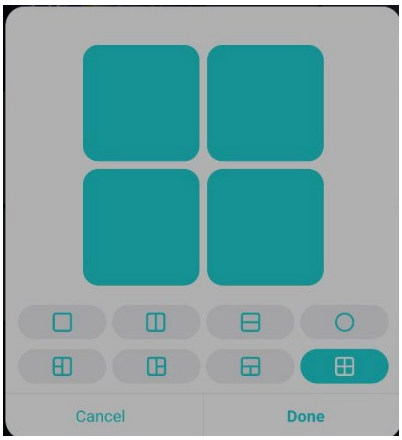


Рисунок 4.7 – Вибір типу підшаблону

5) Додавання нового медіа контенту до спогаду з галереї. Результат: при натисканні на кнопку галереї відбувається перехід на екран вибору медіа контенту з галереї мобільного пристрою. Після натискання кнопки підтвердження вибору медіа контент додається до горизонтального списку фото та відео контенту в низу екрану редагування.

6) Додавання нового медіа контенту до спогаду з камери. Результат: при натисканні на кнопку камери відбувається відкриття камери телефону, де користувач може зробити знімок. Після здійснення знімку він додається до горизонтального списку фото та відео контенту в низу екрану редагування.

7) Переміщення порядку фотографій. Результат: при довгому натисканні на картинку відбувається переміщення картинки на іншу позицію.

8) Підтвердження збереження змін. Результат: при натисканні кнопки назад або кнопки збереження відкривається діалог з підтвердженням на збереження – рисунок 4.8.

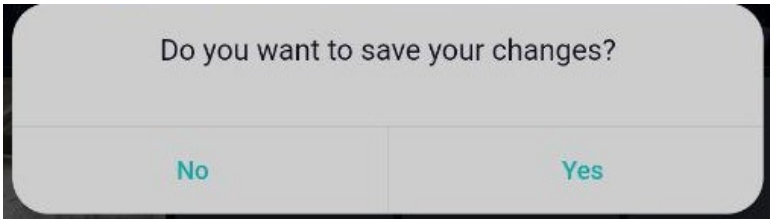


Рисунок 4.8 – Підтвердження збереження

9) Створення спогаду без медіа контенту. Результат: якщо на головному шаблоні відсутні фотографії, то при натисканні кнопки назад або кнопки збереження відкривається діалог з інформацією, що створити спогад без контенту неможливо – рисунок 4.9.

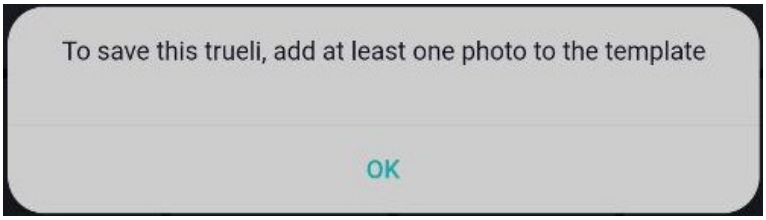


Рисунок 4.9 – Діалог з інформацією про створення спогаду без контенту

Висновок: функціонал працює коректно.

Тестування екрану створення малюнків

1) Створення малюнку для обраного спогаду. Результат: при натисканні на кнопку додавання малюнку на екрані перегляду або редагування спогаду відбувається перехід на екран створення малюнку для обраного спогаду.

2) Зміна кольору курсору. Результат: при натисканні на кнопку вибору палітри курсору відкривається діалог з вибором кольору курсору.

3) Стирання малюнку. Результат: при виборі інструменту «ластик» обирається необхідний інструмент яким можна здійснювати стирання малюнку.

4) Підтвердження збереження. Результат: при натисканні кнопки назад користувачу відображається діалог з підтвердженням змін.

Висновок: функціонал працює коректно.

Тестування екрану створення аудіо запису

1) Створення запису голосового повідомлення. Результат: при натисканні кнопки запису вмикається мікрофон мобільного пристрою та відбувається запис голосу користувача, після натискання кнопки стоп відбувається зупинка запису.

2) Завантаження аудіо файлу з пристрою. Результат: при натисканні кнопки додавання аудіо запису відбувається перехід на екран вибору медіа файлу з пам'яті пристрою. Після вибору аудіо файлу він завантажується у додаток та прикріплюється до спогаду.

3) Підтвердження збереження. Результат: при натисканні кнопки назад користувачу відображається діалог з підтвердженням змін.

Висновок: функціонал працює коректно.

4.3 Модульне тестування мобільного додатку

Чим більше виділень у вашій програмі, тим важче її перевірити вручну. Автоматичні тести допомагають гарантувати правильну роботу вашої програми, перш ніж ви її опублікуєте, зберігаючи швидкість виправлення функцій і помилок.

Модульне тестування — це тип тестування програмного забезпечення, при якому тестуються окремі модулі або компоненти програмного забезпечення. Мета полягає в тому, щоб підтвердити, що кожна одиниця програмного коду працює належним чином. Модульне тестування виконується розробниками під час розробки (фази кодування) програми. Модульні тести виділяють частину коду та перевіряють його правильність. Одиницею може бути окрема функція, метод, процедура, модуль або об’єкт.

Модульне тестування — це метод тестування WhiteBox, який зазвичай виконує розробник. Хоча в практичному світі через брак часу або небажання розробників здійснювати тестування інженери з якості також проводять модульне тестування.

Flutter одразу отримав популярність серед розробників завдяки створенню чудових програм для Android та iOS. Подібно до додатків, створених за допомогою інших інструментів розробки, автоматизоване тестування додатків Flutter є найкращим способом гарантувати якість додатків у найкоротші терміни.

Flutter-тестування поділяється на три категорії:

– Модульний тест;
– Тест віджетів;
– Інтеграційний тест.

Модульні тести зручні для перевірки поведінки окремої функції, методу чи класу. У Flutter пакет тестів надає основну структуру для написання модульних тестів, а пакет flutter_test надає додаткові утиліти для тестування віджетів.

Ми створюємо тестові випадки для нашої програми, оскільки нам потрібно, щоб наша програма була вільна від помилок і відповідала вимогам програми перед публікацією нашої роботи клієнту. Клієнту не потрібен жахливий предмет. Тому, щоб уникнути цього, ми тестуємо нашу програму, створюючи тестові випадки.

Визначення з тестових випадків Software Testing Fundamentals — це низка умов, за яких аналізатор вирішуватиме, чи програма відповідає вимогам або працює точно. Шлях до розробки тестових випадків також може допомогти виявити проблеми в потребах або дизайні програми.

Для розробленого додатку проведемо тестування віджетів. Тестування віджетів інакше називають тестуванням компонентів. Як випливає з назви, він використовується для тестування окремого віджета, і мета цього тесту — перевірити, чи працює віджет і чи виглядає він правильно.

Так само можна використовувати утиліту WidgetTester для різних речей під час тестування, наприклад, надсилання вхідних даних у віджет, пошук частини в дереві віджетів, підтвердження значень тощо.

Тестування віджетів виявляється корисним, коли ми пробуємо певні віджети. Якщо віджет «Текст» не містить тексту, у цей момент тестування віджета видає помилку, вказуючи, що віджет «Текст» не містить тексту. Ми також не розміщуємо для встановлення будь-яких сторонніх залежностей для тестування віджетів у Flutter.

Щоб підключити модульне тестування необхідно додати бібліотеку для тестування до залежностей проєкту у файл pubspec.yaml:

```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

Для перевірки правильності роботи віджету, який відображає назву спогаду створимо файл title_test.dart

```
import 'package:test/test.dart';
void main() {
 group('Title', () {
   test('value should start at Story name', () {
     expect(TitleWidget ().value, ‘Story name’);
   });
   test('value should be changed, () {
     final title = TitleWidget ();
```

```
    title.update(‘Story name new’);
    expect(title.value, ‘Story name new’);
   });
   test('value should be decremented', () {
    final title = TitleWidget();
    title.update(‘Story name old’);
    expect(title.value, ‘Story name old’);
   });
  });
}
```

Для запуску модульного тесту виконаємо наступні дії:

1) Відкриємо файл тесту title_test.dart.

2) Виберемо у меню «Run».

3) Оберемо опцію «tests in title_test.dart».

Результат роботи модульного тестування додатку зображено на рисунку 4.10.

Рисунок 4.10 – Результат роботи модульного тесту

4.4 Висновки

Після проведених тестів можна зробити висновок, що основна ціль проекту була досягнута. Розроблений мобільний додаток надає користувачеві можливість агрегувати медіа контент з галереї мобільного пристрою, переглядати результат роботи агрегування, фільтрувати спогади за назвою та тегами, створювати спогади власноруч, додавати до них аудіо та малюнки. Програмне забезпечення працює стабільно, ефективно та виконує функції згідно визначених вимог.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу розробка методу і засобів веб-системи для пошуку іменованих сутностей у тексті з використанням нейронних мереж.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету: Ракитянська Ганна Борисівна (к.т.н., доц. кафедри ПЗ ВНТУ), Катєльніков Денис Іванович (к.т.н., доц. кафедри ПЗ ВНТУ), Майданюк Володимир Павлович (к.т.н., доц. кафедри ПЗ ВНТУ). Для проведення технологічного аудиту було використано таблицю 5.1 в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	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
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
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

Продовження таблиці 5.1

Ринкові перспективи					
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
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
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 фінансуван ня є	Не потребує додаткового фінансування
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
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виро-бництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів 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

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Ракитянська Г.Б.	Катєльніков Д.І.	Майданюк В.П.
	Бали, виставлені експертами:		
1	4	4	4
2	3	3	3
3	4	4	4
4	3	4	3
5	4	4	4
6	4	3	3
7	3	3	4
8	4	4	4
9	3	3	3
10	3	3	3
11	4	4	4
12	4	4	4
Сума балів	$СБ_1$=43	$СБ_2$=43	$СБ_3$=43
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{43+43+43}{3} = 43$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 43 бали, що згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є високий.

Методи і програмі засоби розпізнавання зображень для агрегування медіа контенту, що розробляються в рамках магістерської кваліфікаційної роботи будуть

цікаві інвесторам як потенційно прибутковий стартап.

Порівняємо нову розробку, що розробляється в магістерській кваліфікаційній роботі з аналогом, який існує на ринку.

Аналогом і даному випадку було обрано додаток Google Photos. Основними недоліками аналога є відсутність приватності користувацьких даних, оскільки для обробки та збереження медіа контенту використовуються хмарні технології сервісу.

У розробці дана проблема вирішується за допомогою локального розпізнавання об’єктів на зображеннях, а також їх шифрування при відправці на хмарні сховища при створенні резервної копії.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 5.4 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
Швидкість генерації спогадів	5	10	2	50%
Точність визначення геолокації	5	5	1	10%
Використання ресурсів інтернет з’єднання, %	80	10	8	20%
Використання ресурсів пристрою, %	50	80	0.625	20%

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.5.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{5.1}$$

або

$$q_i = \frac{P_{Бi}}{P_{Нi}} \tag{5.2}$$

де $P_{Нi}$, $P_{Бi}$ – числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{10}{5} = 2;$$

$$q_2 = \frac{5}{5} = 1;$$

$$q_3 = \frac{80}{10} = 8;$$

$$q_4 = \frac{50}{80} = 0.625;$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \tag{5.3}$$

$$К_{я.в.} = 2 \cdot 0{,}5 + 1 \cdot 0{,}1 + 8 \cdot 0{,}2 + 0.625 \cdot 0{,}2 = 2{,}825$$

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ – індекс технічних параметрів; $I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Нei}}{\sum_{i=1}^{n} P_{Бei}}, \tag{5.5}$$

де P_{Hei}, $P_{Бei}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$I_{е.п.} = \frac{10}{9,6} = 0,96;$$

$$К = \frac{2,825}{0,96} = 2,94.$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде більш конкурентоспроможною, ніж конкурентний товар.

5.2 Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_О = \frac{М}{Т_р} \cdot t \text{ (грн)} \qquad (5.6)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

T_P – число робочих днів в місяці; приблизно $T_P \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

$$З_о = \frac{14000}{22} \cdot 5 = 3182 \text{ (грн)}$$

$$З_о = \frac{12000}{22} \cdot 66 = 36003 \text{ (грн)}$$

Зведемо сумарні розрахунки до таблиці 5.5.

Таблиця 5.5 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	14000	636,4	5	3182
Програміст	12000	545,5	66	36003
Всього				39185

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%} \quad (5.7)$$

$$З_д = 0{,}11 \cdot 39185 = 4310{,}35 \text{ (грн)}$$

3. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.3):

$$Н_{ЗП} = (З_о + З_д) \cdot \frac{\beta}{100} \text{ (грн)} \quad (5.8)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = (39185 + 4310{,}35) \cdot \frac{22}{100} = 9569 \text{ (грн)}$$

4. Витрати на матеріали М та комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$M = \sum_{1}^{n} H_i \cdot Ц_i \cdot K_i - \sum_{1}^{n} B_i \cdot Ц_в \text{ грн.}, \quad (5.9)$$

де H_i – витрати матеріалу *i*-го найменування, кг;

$Ц_i$ – вартість матеріалу *i*-го найменування, грн./кг.;

K_i – коефіцієнт транспортних витрат, K_i = (1,1…1,15);

B_i – маса відходів матеріалу *i*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *i*-го найменування, грн/кг;

n – кількість видів матеріалів.

Таблиця 5.6 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Папір	140	1	140
Ручка	20	1	20
CD-диск	20	1	20
Флешка	150	1	150
Всього			350
З врахуванням коефіцієнта транспортування			385

5. Програмне забезпечення для наукової роботи включає витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження. Для нової розробки використовувались безкоштовні програмні засоби.

6. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$A = \frac{Ц \cdot T}{Tкор \cdot 12} \ [грн], \quad (5.10)$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункту 137.3.3 Податкового кодексу амортизація нараховується на основні засоби вартістю понад 2500 грн. В нашому випадку для написання магістерської роботи використовувався персональний комп'ютер вартістю 25000 грн.

$$А = \frac{25000 \cdot 3}{3 \cdot 12} = 2083.3$$

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \qquad (5.11)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}05 \cdot 528 \cdot 1{,}44 \cdot 0{,}5}{0{,}8} = 23{,}76$$

Витрати на службові відрядження, витрати на роботи, які виконують сторонні підприємства, установи, організації та інші витрати в нашому досліджені не враховуються оскільки їх не було.

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення,

водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = 39185 \cdot \frac{100}{100\%} = 39185 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР:

$$В = 39185+4310,35+9569+385+2083,3+23,76+39185 = 94741,4$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (5.13)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт $\beta = 0,9$.

Звідси:

$$ЗВ = \frac{94741,4}{0,9} = 105268,2 \text{ грн.}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \quad (5.14)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х – ставка податку на прибуток. У 2022 році – 18%.

Припустимо, що при впроваджені результатів наукової розробки покращується якість програмного продукту. Припустимо, що дохід від одного користувача зросте на 5 грн. Кількість одиниць реалізованої продукції (в даному випадку кількість користувачів) також збільшиться: протягом першого року на 50 000, протягом другого року – на 100 000, протягом третього року на 250 000. Реалізація продукції до впровадження розробки складала 1 шт., а дохід від одного користувача складає 10 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [5 \cdot 1 + (10 + 5) \cdot 50000] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 184302{,}47 \text{ грн.}$$

$$\Delta П_2 = [5 \cdot 1 + (10 + 5) \cdot (50000 + 100000)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 552905 \text{ грн.}$$

$$\Delta П_3 = [5 \cdot 1 + (10 + 5) \cdot (50000 + 100000 + 250000)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 1474411{,}2 \text{ грн.}$$

Таким чином, розрахунки показують, що відповідно прогнозуванню комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \qquad (5.15)$$

$k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}}$ = 2…5).

$$PV = 2 \cdot 105268{,}2 = 210536{,}4$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \qquad (5.16)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.17)$$

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

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$\text{ПП} = \frac{184302,47}{(1+0,2)^1} + \frac{552905}{(1+0,2)^2} + \frac{1474411,2}{(1+0,2)^3} = 153585,4 + 383961,8 + 853247,2$$

$$= 1390794,4 \text{ грн.}$$

$$\text{Е}_{\text{абс}} = (1390794,4 - 210536,4) = 1180258 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (5.18)$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$\text{Е}_\text{в} = \sqrt[3]{1 + \frac{1180258}{210536,4}} - 1 = 1.817121 = 181\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \qquad (5.19)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0,18 + 0,05 = 0,23.$$

Так як $E_в > \tau_{\min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \tag{5.20}$$

$$\mathrm{T_{ок}} = \frac{1}{1,81} = 0,55 \text{ роки}$$

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки є доцільним.

5.5 Висновки

В даному розділі було здійснено оцінювання комерційного потенціалурозробки методів і програмних засобів розпізнавання зображень для агрегування медіа контенту у вигляді мобільного додатку для платформ iOS та Android.

Проведено технологічний аудит з залученням трьох експертів. Аналіз експертних даних показав, що рівень комерційного потенціалу розробки високий. Дослідження комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти і є перспективною розробкою. Він має кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складає 94741,1 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 105268,2 грн.

Вкладені інвестиції в даний проект окупляться через 5 з половиною місяців при прогнозованому прибутку 1390794,4 грн. за три роки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі проведено розробку методів та засобів розпізнавання зображень для агрегування медіа контенту. У результаті проведення роботи було створено мобільний застосунок за допомогою фреймворку Flutter, який доступний для платформ iOS та Android. Відмінністю розробленого застосунку є його реалізація на основні мобільні платформи, а також забезпечення приватності користувацьких даних за допомогою шифрування.

У результаті аналізу предметної області було виявлено, що в сучасній сфері мобільних застосунків існує потреба у реалізації мобільного додатку, який допоможе здійснити агрегацію медіа контенту з галереї мобільного пристрою для створення спогадів, які будуть включати в себе лише ті фото та відео матеріали, які мають певну естетичну цінність. Враховуючи ряд вимог до розробки мобільного застосунку було вирішено обрати архітектуру MVC та здійснити реалізацію додатку за допомогою мультиплатформенного фреймворку Flutter на мові Dart. Для організації менеджеру станів екранів та архітектури додатку було обрано менеджер станів GetX. Для реалізації алгоритму агрегування медіа контенту було обрано відкриту бібліотеку TensorFlow, яка допомагає здійснювати аналіз зображень на предмет виявлення на них об'єктів. Для здійснення локального збереження даних було обрано відкриту бібліотеку SqfLite, яка основана на мові запитів SQLite.

Проведено аналіз з такими аналогами на ринку, як Google Photos та Apple Photos. В процесі здійснення аналізу було визначено слабкі та сильні сторони аналогів та визначено відмінність розроблюваної системи від її конкурентів. Виконавши побудову таблиці порівняння аналогів було здійснено висновок, що система є конкурентоспроможною та має ряд переваг які роблять її кращою за аналогічні системи, зокрема головною перевагою є забезпечення приватності користувацьких даних, оскільки вся обробка зображень здійснюється локально на пристрої без використання сторонніх сервісів чи серверів.

В процесі проектування було розроблено діаграму використання мобільного додатку. Спроектовано логічну та фізичну структури мобільного додатку, які допомагають розробнику оптимізувати свою роботу, а також будуть зрозумілими

для інших фахівців, що працюють над проектом. Проведено розробку структури інтерфейсу, розроблений дизайн мобільного додатку спрямований на належне представлення користувачеві всіх візуальних складових та забезпечення правильної роботи усього функціоналу застосунку.

Проведено аналіз найвідоміших засобів реалізації програмного продукту, які мають можливість одночасної розробки під різні платформи. Було проаналізовано такі засоби, як Kotlin Multiplatform, React Native та Flutter. Після виконання аналізу засобів реалізації було здійснено вибір в користь розробки мобільного додатку за допомогою фреймворку Flutter.

Проведено аналіз середовищ розробки, таких як Visual Studio Code, Android Studio та Xcode. Після проведення аналізу середовищ було обрано Android Studio для здійснення основної розробки мобільного додатку для платформи Android, а також середовище Xcode для здійснення розробки для платформи iOS.

Здійснено розробку програмного забезпечення за допомогою фреймворку Flutter та мови Dart. Здійснено розробку візуальної та логічної частини усіх екранів. Проведено розробку модулів агрегування медіа контенту, створення спогадів, здійснення авторизації у системі.

Після здійснення розробки програмного забезпечення було проведено тестування чорної та білої скриньки. Під час проведення тестування було здійснено аналіз усіх можливих випадків, і створено відповідні шаблони їх тестування. Після виконання тестування було доведено, що система відповідає своїй головній задачі, а також виконує усі функції коректно.

Економічні розрахунки витрат та прибутків від впровадження розробки, термінів окупності та визначення її економічного ефекту показали, що в сучасній технічній ситуації розробка є конкурентоспроможною на IT-ринку.

Робота має практичну цінність, оскільки розроблені модулі агрегування медіа контенту можуть використовуватись для інших проєктів, а також має високу конкурентоспроможність на ринку, оскільки має мало прямих конкурентів при цьому маючи низку вагомих переваг.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Прус Б.В. Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту // Г.Б. Ракитянська, Б. В. Прус., – Збірник матеріалів Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» – Вінниця ВНТУ, Вінницька академія неперервної освіти, 2022 – С. 58-60.

2. Smartphones vs Cameras: Closing the gap on image quality [Електронний ресурс] – Режим доступу: https://www.dxomark.com/smartphones-vs-cameras-closing-the-gap-on-image-quality.

3. Photos for iOS and iPadOS [Електронний ресурс] – Режим доступу: https://www.apple.com/ios/photos.

4. Google Photos [Електронний ресурс] – Режим доступу: https://www.google.com/photos/about.

5. Google Photos. Growth [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Google_Photos#:~:text=In%20May%202017%2C%20Google%20announced%20that%20Google%20Photos%20has%20over,photos%20and%20videos%20are%20uploaded.

6. Google Drive [Електронний ресурс] – Режим доступу: https://www.google.com/drive.

7. Apple iCloud [Електронний ресурс] – Режим доступу: https://www.icloud.com.

8. Теорія розпізнавання образів [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Теорія_розпізнавання_образів.

9. Tensor Flow [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/TensorFlow.

10. Pramod Singh. Learn Tensor Flow 2.0. 1st edition / Apress, 2020. – 343 с.

11. UML. Короткий посібник по UML (UML Distilled: A Brief Guide to the Standard Object Modeling Language.) 1999.

12. Архітектура мобільних додатків [Електронний ресурс] – Режим доступу: https://koloro.ua/arkhitektura-prilozheniya.html.

13. Архітектура модель-вид-контролер [Електронний ресурс] – режим доступу: https://www.codecademy.com/articles/mvc.

14. Проектування мобільного додатку [Електронний ресурс] – Режим доступу: https://wezom.com.ua/ua/blog/proektirovanie-mobilnogo-prilozheniya.

15. Photo Manager package [Електронний ресурс] – Режим доступу: https://pub.dev/packages/photo_manager.

16. Sqflite package [Електронний ресурс] – Режим доступу: https://pub.dev/packages/sqflite.

17. Розробка дизайну інтерфейсу мобільних додатків [Електронний ресурс] – Режим доступу: https://web24.pro/android-rozrobka/rozrobka-dyzajnu-interfejsu-mobilnyh-dodatkiv.

18. Theo Mendel. Golden Rules to User Interface Design / DMK 2001 – 54 с.

19. Figma [Електронний ресурс] – Режим доступу: https://www.figma.com.

20. Kotlin Multiplatform [Електронний ресурс] – Режим доступу: https://kotlinlang.org/docs/multiplatform.html.

21. React Native [Електронний ресурс] – Режим доступу: https://reactnative.dev.

22. Flutter [Електронний ресурс] – Режим доступу: https://flutter.dev.

23. Dart – programming language [Електронний ресурс] – Режим доступу: https://dart.dev.

24. Flutter vs Native vs React-Native: Examining performance [Електронний ресурс] – Режим доступу: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980.

25. Visual Studio Code [Електронний ресурс] – Режим доступу: https://code.visualstudio.com.

26. GitHub [Електронний ресурс] – Режим доступу: https://github.com.

27. Android Studio [Електронний ресурс] – Режим доступу: https://developer.android.com/studio.

28. Google I/O 2013 [Електронний ресурс] – Режим доступу: https://developers.google.com/events/io/2013.

29. Xcode [Електронний ресурс] – Режим доступу: https://developer.apple.com/xcode.

30. Firebase [Електронний ресурс] – Режим доступу: https://firebase.google.com.

31. Тестування мобільних додатків [Електронний ресурс] – Режим доступу: https://qagroup.com.ua/publications/testuvannia-mobilnykh-dodatkiv-vid-a-do-ia.

ДОДАТОК А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ
д.т.н., професор Романюк О.Н.

"16 " вересня 2022 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

к.т.н., доц. Г.Б. Ракитянська

"16 " 09 2022 р.

Виконав:

студент гр.2ПІ-21м Б.В. Прус

"16 " 09 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту».

Галузь застосування – мобільні додатки.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є розробка алгоритму для агрегування зображень у події за рахунок застосування програмних засобів розпізнавання зображень.

Призначення роботи – розробка методів та програмних засобів розпізнавання зображень для агрегування медіа контенту.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Прус Б.В. Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту // Г.Б. Ракитянська, Б. В. Прус., – Збірник матеріалів Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» – Вінниця ВНТУ, Вінницька академія неперервної освіти, 2022 – С. 58-60.

4. Технічні вимоги

– Android 7.0 (API level 24) і новіше;
– iOS 12.1 і новіше;
– мінімум 1024 мб вільного простору на диску.

5. Конструктивні вимоги.

Мобільний додаток повинен бути зручним у використанні та мати інтуїтивно зрозумілий інтерфейс, а також бути доступним для платформ iOS та Android.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз, вибір та обґрунтування актуальності розробки	17.09.2022 – 24.09.2022
2	Аналіз існуючих аналогів	25.09.2022-30.09.2022
3	Розробка структури мобільного додатку та алгоритму агрегування медіа контенту	01.10.2022-08.10.2022
4	Проектування бази даних	09.10.2022-10.10.2022
5	Розробка структури та інтерфейсу мобільного додатку	11.10.2022-21.10.2022
6	Розробка програмного забезпечення	22.10.2022-22.11.2022
7	Тестування програмного забезпечення	23.11.2022-30.11.2022
8	Економічна частина	01.12.2022-08.12.2022

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б

Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ – 21м

Науковий керівник: к.т.н. доц. Ракитянська Г.Б.

Memori	
Оригінальність	**89,2 %**
Схожість	10,8 %

Аналіз звіту подібності

■ **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 доопрацювання.

□ 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.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методів і програмних засобів розпізнавання зображень для агрегування медіа контенту».

Автор _____ Прус Богдан Вікторович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ Ракитянська Г.Б., к.т.н., доц. каф. ПЗ
(за потреби) (підпис) (прізвище, ініціали, посада)

ДОДАТОК В

Лістинг вихідного коду

```
GlobalBinding? globalBinding;
///For dev change to $AppTitle-DEV
late String applicationTitle;
final FirebaseAnalytics analytics = FirebaseAnalytics.instance;
final FirebaseAnalyticsObserver analyticsObserver =
    FirebaseAnalyticsObserver(analytics: analytics);
final Alice alice = Alice();
bool isRunning = false;
late String isolateId;
PackageInfo? packageInfo;
late String initTime;
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  final PackageInfo packageInfo = await PackageInfo.fromPlatform();
  final Flavor flavor =
      (packageInfo.packageName == bundleIdProd) ? Flavor.PROD : Flavor.DEV;
  isolateId = await IdGenerator().generateId();
  globalBinding = GlobalBinding(flavor);
  applicationTitle = flavor == Flavor.PROD ? AppTitle : "$AppTitle-DEV";
  await Firebase.initializeApp();
  HttpOverrides.global =
      CustomHttpOverrides(await PreferenceManager.getProxyAddress());
  await StatusBarColorizer().updateStatusBar(colorPrimary);
  WidgetsBinding.instance
      .addObserver(LifecycleEventHandler(resumeCallBack: () async {
    await StatusBarColorizer().updateStatusBar(StatusBarColorizer().lastColor);
  }));
  await GetStorage.init();
  SystemChrome.setPreferredOrientations(const [
    DeviceOrientation.portraitUp,
  ]);
  Future<void> run() async {
    final CoreApp app = CoreApp();
```

```
    isRunning = true;
    initTime = DateFormat('MM.dd kk:mm:ss').format(DateTime.now());
    runApp(app);
  }
  if (kReleaseMode) {
    FlutterError.onError = FirebaseCrashlytics.instance.recordFlutterError;
    runZonedGuarded<Future<void>>(
        run, FirebaseCrashlytics.instance.recordError);
  } else {
    runZoned(run);
  }
}
```

Лістинг файлу CoreApp

```
/// widget of app and its configuration
class CoreApp extends BaseStatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return CoreAppState();
  }
}
class CoreAppState extends BaseControlledState<CoreApp, CoreAppController> {
  @override
  CoreAppController getController() => CoreAppController();
  @override
  Widget getLayout() {
    return GetMaterialApp(
      builder: (BuildContext context, Widget? child) {
        Get.locale = Localizations.localeOf(context);
        return MediaQuery(
          data: MediaQuery.of(context).copyWith(
              textScaleFactor: 1, devicePixelRatio: window.devicePixelRatio),
          child: Stack(children: [
            child!,
            if (controller.absorbPointerEnabled)
```

```
        Container(
          color: colorPrimary,
          child: getProgress(background: false),
        )
    ]),
  );
},
transitionDuration: controller.absorbPointerEnabled
    ? Duration.zero
    : AppPages.transitionDuration,
debugShowCheckedModeBanner: false,
localizationsDelegates: const [
  LocalizationDelegate(),
  CountryLocalizations.delegate,
  GlobalMaterialLocalizations.delegate,
  GlobalWidgetsLocalizations.delegate,
  GlobalCupertinoLocalizations.delegate
],
supportedLocales: Localization.getSupportedLocales(),
localeResolutionCallback: (locale, supportedLocales) {
  for (final Locale supportedLocaleLanguage in supportedLocales) {
    if (supportedLocaleLanguage.languageCode == locale!.languageCode) {
      return supportedLocaleLanguage;
    }
  }
  // If device not support with locale to get language code then default get first on from the list
  return supportedLocales.first;
},
navigatorKey: alice.getNavigatorKey(),
title: applicationTitle,
theme: buildThemeData(),
initialRoute: AppPages.INITIAL,
getPages: AppPages.routes,
initialBinding: globalBinding,
routingCallback: controller.routingCallback,
```

```
      navigatorObservers: [
        NavigationHistoryObserver(),
        analyticsObserver,
      ],
    );
  }
}
```

Лістинг файлу CoreAppController

```
class CoreAppController extends BaseController {
  bool? _isFirstStart;
  bool? get isFirstStart => _isFirstStart;
  bool _absorbPointerEnabled = false;
  bool get absorbPointerEnabled => _absorbPointerEnabled;
  set absorbPointerEnabled(bool value) {
    if (absorbPointerEnabled != value) {
      _absorbPointerEnabled = value;
      update();
    }
  }
  DateTime? _pauseDate;
  UserEntity? _user;
  UserEntity? get user => _user;
  set user(UserEntity? user) {
    _user = user;
    update();
  }
  final RouteObserver<PageRoute> routeObserver = RouteObserver<PageRoute>();
  final NavigatorObserver observer = NavigatorObserver();
  StreamSubscription? _currentUserUpdatedNotifierSubscription;
  StreamSubscription? _appLifestyleNotifierSubscription;
  StreamSubscription? _noMemoryNotifierSubscription;
  StreamSubscription? _storySharingNotifierSubscription;
  bool noMemoryDialogOpen = false;
  bool isFlutterBottomSheetOpen = false;
```

```
Future<void> initApp() async {
  await LogService()
      .logRealtime('Initialize MainIsolate CoreApp by $isolateId');
  _isFirstStart = await PreferenceManager.getFirstStartFlag();
  packageInfo = await PackageInfo.fromPlatform();
  await PreferenceManager.setBuildNumber(
      packageInfo!.buildNumber.toIntOrNull() ?? -1);
  if (_isFirstStart!) {
    await LogService().logRealtime("IS FIRST START");
    await SecurePreferenceManager.clearAllData();
    await PreferenceManager.clear();
    if (Platform.isAndroid) await FileService().clearDirectories(true);
    await GetStorage().erase();
    await PreferenceManager.setIsNeedToRescanGallery(false);
  } else {
    final bool initDb = await DatabaseManager().initDB();
    if (initDb) {
      await LogService().logRealtime("IS NOT FIRST START");
      _user = await Get.find<AuthRepository>().getCurrentUserDb();
      await LogService().logRealtime('CurrentUserGlobalId: ${user?.globalUserId}');
      if (_user != null) {
        Analytics().setAnalyticsUser(_user!);
      } else if (!_isFirstStart!) {
        throw DatabaseInitializationException("Cannot find user in db");
      }
      await RequestTable().resetProcessorForRequests();
    } else {
      final String dbPath = await DatabaseManager().getDbPath();
      final String logString =
          "Database ($dbPath) exists: ${File(dbPath).existsSync()}";
      await LogService().logRealtime(logString);
      throw DatabaseInitializationException("Cannot init db | $logString");
    }
    BackupService().backupAllowed =
        await SecurePreferenceManager.getBackupAllowed();
```

```
  }
  _initListeners();
}
void _initListeners() {
  _currentUserUpdatedNotifierSubscription =
      currentUserUpdatedNotifier.listen((value) {
    user = value;
  });
  _noMemoryNotifierSubscription = noMemoryNotifier.listen((value) async {
    await LogService()
        .logRealtime("CoreAppController::noMemoryNotifier called");
    await displayNoMemoryDialog(value);
  });
  _storySharingNotifierSubscription =
      uploadStoryNotifier.listen((value) async {
    if (RequestService().hasSharingUploadRequests()) {
      await WakelockService().enable(WakelockDto(sharingPending: true));
    } else {
      await WakelockService().enable(WakelockDto(sharingPending: false));
    }
  });
  _appLifestyleNotifierSubscription =
      appLifecycleStateNotifier.listen((value) async {
    if (value == AppLifecycleState.paused) {
      _pauseDate = DateTime.now();
      await LogService().logRealtime('AppLifecycleState is paused');
    } else if (value == AppLifecycleState.resumed) {
      if (_pauseDate != null) {
        if (Get.currentRoute == rootRoute || Get.currentRoute == loginRoute) {
          _pauseDate = null;
          return;
        }
        final int lockTime = await PreferenceManager.getLockTime();
        await LogService().logRealtime(
            "AutoLock: \nLockTime: $lockTime\nPause date: $_pauseDate");
```

```
      if (DateTime.now().difference(_pauseDate!).inMinutes.abs() >=
            lockTime &&
          ScanningService().storyGenerateNotifier.valueOrNull != true &&
          !RequestService().hasSharingUploadRequests()) {
        final String? pinCode = await SecurePreferenceManager.getPinCode();
        if (pinCode.isNotNullOrEmpty() ||
            await PreferenceManager.getBiometricEnable()) {
          await LogService().logRealtime("Autolock send to login");
          Get.offAllNamed(loginRoute);
        }
      }
    }
    _pauseDate = null;
  }
 });
}
Future<void> displayNoMemoryDialog(NoMemoryType type) async {
 if (noMemoryDialogOpen) return;
 Map<String, String> textKey = unexpectedError;
 if (type == NoMemoryType.RESTORATION) {
  textKey =
      Platform.isIOS ? restorationNoMemoryIosMsg : restorationNoMemoryMsg;
 } else if (type == NoMemoryType.DOWNLOADING_MEMORY) {
  textKey = Platform.isIOS ? getStoryNoMemoryIosMsg : getStoryNoMemoryMsg;
 } else if (type == NoMemoryType.BACKUP) {
  textKey = Platform.isAndroid
      ? backupNoMemoryCloudAndroidMsg
      : backupNoMemoryCloudIOSMsg;
 }
 final bool isBlocking = type == NoMemoryType.DOWNLOADING_MEMORY;
 noMemoryDialogOpen = true;
 await Get.dialog(
    NoMemoryDialog(
      isBlocking: isBlocking,
      headerKey: attention,
```

```
descriptionKey: textKey,
mainIcon: Image.asset(
  AppIcons.attention,
  height: 48,
  width: 48,
),
buttonTextKey: Platform.isIOS
    ? goToStorageButtonIosTitle
    : goToStorageButtonTitle,
onButtonPressed: () async {
  if (type == NoMemoryType.RESTORATION ||
      type == NoMemoryType.BACKUP) {
    Get.back();
  }
  if (type == NoMemoryType.BACKUP) {
    if (Platform.isAndroid) {
      await launchUrl(
          Get.find<ConfigResponse>().settings!.storageConfigsGoogle!);
    } else if (Platform.isIOS) {
      await SystemSettings.system();
    }
  } else {
    await SystemSettings.internalStorage();
  }
},
retryButtonTextKey:
    type == NoMemoryType.RESTORATION ? null : tryAgainButton,
retryButtonPressed: () {
  if (type == NoMemoryType.RESTORATION) return;
  if (type == NoMemoryType.BACKUP) {
    BackupService().startBackup();
    return;
  }
  RequestService().startOrResumeRequests();
},
```

```
      ),
      barrierDismissible: !isBlocking);
  noMemoryDialogOpen = false;
}
Future<void> goToMainOrVerification() async {
  if (!user!.isPhoneVerified! &&
      !user!.isEmailVerified! &&
      await PreferenceManager.getIsNeedToShowVerificationScreen()) {
    Get.offAllNamed(verificationRoute);
  } else {
    Get.offAllNamed(mainScreenRoute);
  }
}
void wipeControllerData() {
  _isFirstStart = null;
  _absorbPointerEnabled = false;
  _pauseDate = null;
  _user = null;
  _currentUserUpdatedNotifierSubscription?.cancel();
  _appLifestyleNotifierSubscription?.cancel();
  _storySharingNotifierSubscription?.cancel();
  _noMemoryNotifierSubscription?.cancel();
  noMemoryDialogOpen = false;
  isFlutterBottomSheetOpen = false;
}
@override
void dispose() {
  _currentUserUpdatedNotifierSubscription?.cancel();
  _appLifestyleNotifierSubscription?.cancel();
  _storySharingNotifierSubscription?.cancel();
  _noMemoryNotifierSubscription?.cancel();
  super.dispose();
}
Future<void> routingCallback(Routing? value) async {
  LogService().logRealtime(
```

```
      "Current route: previous: ${value?.previous} current: ${value?.current}");
    if (value != null &&
        value.isDialog == false &&
        value.isBottomSheet == false) {
      if (value.current == mainScreenRoute &&
          await PreferenceManager.getShowUpdateDialog()) {
        await ConfigService().checkConfig();
      }
    }
  }
}
```

Лістинг файлу AutoCreateStoryManager

```
class AutoCreateStoryManager {
  final BehaviorSubject<dynamic> storyGenerateErrorNotifier =
      BehaviorSubject<dynamic>();
  final PublishSubject<CreateStoryStep> createStoryStepNotifier =
      PublishSubject();
  static fmpMediaFile.MediaType? getAssetMediaType(AssetType type) {
    if (type == AssetType.image) return fmpMediaFile.MediaType.IMAGE;
    if (type == AssetType.video) return fmpMediaFile.MediaType.VIDEO;
    return null;
  }
  Future<bool> _checkScreenRecordsIOS(fmpMediaFile.MediaFile mediaFile) async {
    final FlutterFFprobe flutterFFprobe = FlutterFFprobe();
    bool isNeedToSkip = true;
    final MediaInformation mediaInformation =
        await flutterFFprobe.getMediaInformation(mediaFile.path!);
    final Map<dynamic, dynamic> mp = mediaInformation.getAllProperties();
    ///Finding the right parameter(color_range) to define a screen recording
    ///if color_range == 'pc' - screenRecord
    ///if color_range == 'tv' - camera video
    entities:
    for (final f in mp.entries) {
      if (f.key == 'streams') {
```

```
      for (final r in f.value) {
        if (r['color_range'] != null && r['color_range'] != 'pc') {
          isNeedToSkip = false;
          break entities;
        }
      }
    }
  }
  return isNeedToSkip;
}
Future<void> _getAssetLocations(Map<String, IfdTag>? mediaFileExifData,
    fmpMediaFile.MediaFile mediaFile, ContentModel model) async {
  Pair<double, double>? location;
  if (mediaFile.type == fmpMediaFile.MediaType.IMAGE) {
    location = await PhotoManagerService()
        .locationFromFile(exifData: mediaFileExifData);
  } else if (mediaFile.type == fmpMediaFile.MediaType.VIDEO) {
    var tempLocation =
        (await FlutterVideoInfo().getVideoInfo(mediaFile.path!))!.location;
    if (tempLocation != null) {
      double latitude, longitude;
      latitude = double.parse(
          tempLocation.substring(tempLocation.startsWith('+') ? 1 : 0, 8));
      tempLocation = tempLocation.replaceRange(0, 8, '');
      longitude = double.parse(tempLocation.substring(
          tempLocation.startsWith('+') ? 1 : 0, tempLocation.length - 1));
      location = Pair<double, double>(latitude, longitude);
    }
  }
  model.latitude = location?.left ?? model.latitude;
  model.longitude = location?.right ?? model.longitude;
}
Future<List<ContentModel>> _convertAssetEntityToMediaContent(
    List<AssetEntity> assets) async {
  final List<ContentModel> result = [];
```

```
  for (final AssetEntity asset in assets) {
    final mediaFile = await FlutterMultiMediaPicker.getMediaFile(
        fileId: asset.id, type: getAssetMediaType(asset.type)!);
    if (!(mediaFile != null &&
        mediaFile.path != null &&
        mediaFile.path!.isNotEmpty &&
        File(mediaFile.path!).existsSync())) {
      continue;
    }
    if (asset.type == AssetType.video && Platform.isIOS) {
      final bool isNeedToSkip = await _checkScreenRecordsIOS(mediaFile);
      if (isNeedToSkip) {
        continue;
      }
    }
    final Map<String, IfdTag>? mediaFileExifData =
        await PhotoManagerService().getExifDataFromFile(mediaFile.path);
    final ContentModel model = ContentModel(
      dateCreateOriginal: (await PhotoManagerService()
              .getImageOriginalDateTime(exifData: mediaFileExifData)) ??
          asset.createDateTime,
      dateCreatePhotoManager: asset.createDateTime,
      type: asset.type,
      pathOrigin: mediaFile.path,
      assetId: asset.id,
    );
    await _getAssetLocations(mediaFileExifData, mediaFile, model);
    result.add(model);
  }
  return result;
}
List<Pair<List<AssetEntity>, List<AssetEntity>>> _filterAssets(
    List<AssetEntity> content) {
  List<Pair<List<AssetEntity>, List<AssetEntity>>> filteredAssets = [];
  filteredAssets = ContentDateFilter().filterContentByDate(content);
```

```
  return filteredAssets;
}
Future<void> _updateScanData(
    int lastReadContentDate, List<String> usedAssetsId) async {
  ScanDataEntity? scanDataEntity = await ScanDataTable().getDataById(1);
  if (scanDataEntity == null) {
    scanDataEntity = ScanDataEntity(
        lastScannedImageDate: lastReadContentDate,
        usedAssetsId: usedAssetsId,
        lastScanDate: DateTime.now().millisecondsSinceEpoch);
    await ScanDataTable().addData(scanDataEntity);
  } else {
    if (scanDataEntity.lastScannedImageDate == 0 ||
        scanDataEntity.lastScannedImageDate > lastReadContentDate) {
      scanDataEntity.lastScannedImageDate = lastReadContentDate;
    }
    scanDataEntity.usedAssetsId.addAll(usedAssetsId);
    await ScanDataTable().updateData(scanDataEntity, 1);
  }
}
Future<bool> checkUnscannedAssets() async {
  final ScanDataEntity? scanDataEntity = await ScanDataTable().getDataById(1);
  await LogService().logRealtime('checkUnscannedAssets 1 - $scanDataEntity');
  if (scanDataEntity == null) {
    final bool isEmpty =
        (await PhotoManagerService().readAssetsFromStorage()).isEmpty;
    if (await PreferenceManager.getIsScanAfterChangedPermission() &&
        isEmpty) {
      return false;
    } else if (await PermissionService.isStoragePermissionPermanentlyDenied() || Platform.isIOS &&
        await PermissionService.iOSisLimitedPhotosPermission() &&
        isEmpty) {
      return false;
    }
    return true;
```

```
  } else {
    int? _startDate;
    if (!await PreferenceManager.getIsCheckNewPhotoAfterBackup()) {
      _startDate = scanDataEntity.lastScanDate;
    }
    final List<AssetEntity> content = await PhotoManagerService()
        .readAssetsFromStorage(startDate: _startDate, endDate: 0);
    await LogService()
        .logRealtime('checkUnscannedAssets 2 - ${content.length}');
    if (content.isNotEmpty) {
      return true;
    }
    return false;
  }
}
Future<bool> showRescanDialog() async {
  final List<StoryEntity> stories = await StoryTable().getAllData();
  if (!await PreferenceManager.getIsNeedToRescanGallery()) {
    return true;
  }
  if (stories.isNotEmpty) {
    final bool? result = await Get.dialog(AskDialog(
      titleKey: rescanGalleryDialogTitle,
      descriptionKey: rescanGalleryDialogDescription,
    ));
    if (result == true) {
      final StoryRepository _storyRepository = Get.find<StoryRepository>();
      final MainScreenController _mainScreenController =
          Get.find<MainScreenController>();
      await PhotoManagerService().reCreateScanDataEntity();
      await _storyRepository.deleteStoriesNotApproved();
      await _mainScreenController.getStories();
      return true;
    }
  }
```

```
  if ((await PhotoManagerService().readAssetsFromStorage()).isEmpty) {
    return true;
  }
  return false;
}
Future<int> getNewPhotosCount() async {
  final ScanDataEntity? scanDataEntity = await ScanDataTable().getDataById(1);
  if (scanDataEntity != null) {
    final List<AssetEntity> content = await PhotoManagerService()
        .readAssetsFromStorage(
            startDate: DateTime.now().millisecondsSinceEpoch,
            endDate: scanDataEntity.lastScannedImageDate);
    return content.length;
  }
  return 0;
}
Future<void> getAllAvailableContentAndStartCreate(
    {required int startDate, int? endDate}) async {
  try {
    await LogService().logRealtime(
        'getAllAvailableContentModel: start readAssetsFromStorage');
    final List<AssetEntity> content = await PhotoManagerService()
        .readAssetsFromStorage(startDate: startDate, endDate: endDate);
    if (content.isNotEmpty) {
      final int lastReadContentDate =
          content.first.createDateTime.millisecondsSinceEpoch;
      await LogService()
          .logRealtime('getAllAvailableContentModel: start _filterAssets');
      ///Left - all assets, Right - filtered assets
      final List<Pair<List<AssetEntity>, List<AssetEntity>>> filteredContent =
          _filterAssets(content);
      for (final Pair<List<AssetEntity>, List<AssetEntity>> content
          in filteredContent) {
        if (ScanningService().stopFlagForScan) break;
        final ScanResult scanResult = await PreferenceManager.getScanResult();
```

```
final List<String> usedAssetsId =
    content.left!.map((e) => e.id).toList();
scanResult.allPhotosCount += usedAssetsId.length;
await LogService().logRealtime(
    'getAllAvailableContentModel: start _convertAssetEntityToMediaContent');
List<ContentModel> contentModel =
    await _convertAssetEntityToMediaContent(content.right!);
await LogService().logRealtime(
    'getAllAvailableContentModel: start filterSameContent');
contentModel =
    await ContentDateFilter().filterSameContent(contentModel);
await LogService().logRealtime(
    'getAllAvailableContentModel: start createStoryModel');
StoryModel? storyModel =
    await CreateStoryModelManager().createStoryModel(contentModel);
if (storyModel == null) {
  await _updateScanData(lastReadContentDate, usedAssetsId);
  continue;
}
createStoryStepNotifier
    .add(DetectingObjectsStep(storyModel.content!.length));
await LogService()
    .logRealtime('getAllAvailableContentModel: start processImages');
storyModel = await ObjectsDetector().processImages(storyModel);
if (storyModel.content?.isEmpty == true &&
    storyModel.mainContent == null) {
  await LogService().logRealtime(
      'getAllAvailableContentModel: processImages return null');
  await _updateScanData(lastReadContentDate, usedAssetsId);
  await PreferenceManager.setScanResult(scanResult);
  continue;
}
await LogService()
    .logRealtime('getAllAvailableContentModel: start getLocation');
createStoryStepNotifier.add(FindingPlaceMarkStep());
```

```
        storyModel = await LocationsManager().getLocation(storyModel);
        if (storyModel == null) {
          await LogService().logRealtime(
              'getAllAvailableContentModel: getLocation return null');
          break;
        }
        await LogService()
            .logRealtime('getAllAvailableContentModel: start getTitle');
        createStoryStepNotifier.add(BuildingStoryStep());
        storyModel = await TitleManager().getTitle(storyModel);
        await LogService()
            .logRealtime('getAllAvailableContentModel: start saveContent');
        storyModel = await ContentManager().saveContent(storyModel);
        final int storyModelContent = storyModel?.content?.length ?? 0;
        await LogService().logRealtime(
            'getAllAvailableContentModel: start createStoryEntity');
        final StoryEntity storyEntity = await CreateStoryModelManager()
            .createStoryEntity(StoryModel.copy(storyModel!));
        storyUpdatedNotifier.add(storyEntity.globalStoryId!);
        await _updateScanData(lastReadContentDate, usedAssetsId);
        scanResult.skippedPhotosCount +=
            usedAssetsId.length - (storyModelContent + 1);
        ++scanResult.totalTrueliesCount;
        await PreferenceManager.setScanResult(scanResult);
      }
    }
  } catch (err, stackTrace) {
    await LogService().logRealtime(
        'Error getAllAvailableContentModel: Error - $err, StackTrace - $stackTrace');
  }
 }
}
```

Лістинг файлу PhotoManager

```
class PhotoManagerService {
```

```
Future<List<AssetEntity>> readAssetsFromStorage(
    {int? startDate, int? endDate}) async {
  List<AssetEntity> allPhotos = [];
  if (endDate == null) {
    endDate = 0;
  }
  if (startDate == null) {
    startDate = DateTime.now().millisecondsSinceEpoch;
  }
  final DateTimeCond? createDtCond = DateTimeCond(
    min: DateTime.fromMillisecondsSinceEpoch(endDate),
    max: DateTime.fromMillisecondsSinceEpoch(startDate),
  );
  PhotoManager.setIgnorePermissionCheck(true);
  final List<AssetPathEntity> paths = await PhotoManager.getAssetPathList(
    filterOption: FilterOptionGroup(
      createTimeCond: createDtCond,
      videoOption: const FilterOption(
        durationConstraint: DurationConstraint(
          min: Duration(seconds: 1),
          max: Duration(minutes: 10),
        ),
      ),
    ),
  );
  if(paths.isEmpty) return allPhotos;
  if (Platform.isIOS) {
    late AssetPathEntity cameraFolder;
    AssetPathEntity? screenshots;
    for (final element in paths) {
      final folderName = element.name.replaceAll(" ", "").toLowerCase();
      if (folderName.contains("screenshot")) screenshots = element;
      if (_checkIosCameraFolder(element)) cameraFolder = element;
    }
    allPhotos = await cameraFolder.getAssetListRange(
```

```
        start: 0, end: cameraFolder.assetCount);
    final List<AssetEntity> screenshotsList = screenshots != null
        ? await screenshots.getAssetListRange(
            start: 0, end: screenshots.assetCount)
        : [];
    screenshotsList.forEach((element) {
      allPhotos.removeWhere((element2) => element.id == element2.id);
    });
  } else if (Platform.isAndroid) {
    for (final element in paths) {
      if (element.name == 'Camera') {
        allPhotos = await element.getAssetListRange(
            start: 0, end: element.assetCount);
      }
    }
  }
  final ScanDataEntity? scanDataEntity = await ScanDataTable().getDataById(1);
  if (scanDataEntity != null) {
    allPhotos.removeWhere(
        (element) => scanDataEntity.usedAssetsId.contains(element.id));
  }
  allPhotos.sort((s1, s2) {
    return s2.createDateTime.compareTo(s1.createDateTime);
  });
  return allPhotos;
}
Future<Pair<double, double>?> locationFromFile(
    {String? filePath, Map<String, IfdTag>? exifData}) async {
  try {
    if (exifData == null && filePath != null) {
      exifData = await getExifDataFromFile(filePath);
    }
    if (exifData != null) {
      if (exifData["GPS GPSLatitude"] != null &&
          exifData["GPS GPSLongitude"] != null) {
```

```
      final IfdTag longitudeIfdTag = exifData["GPS GPSLongitude"]!;
      final IfdTag longitudeRefIfdTag = exifData["GPS GPSLongitudeRef"]!;
      final IfdTag latitudeIfdTag = exifData["GPS GPSLatitude"]!;
      final IfdTag latitudeRefIfdTag = exifData["GPS GPSLatitudeRef"]!;
      final LatLongConverter converter = LatLongConverter();
      double longitude = converter.getDecimalFromDegree(
        (longitudeIfdTag.values as IfdRatios).ratios[0].numerator /
            (longitudeIfdTag.values as IfdRatios).ratios[0].denominator,
        (longitudeIfdTag.values as IfdRatios).ratios[1].numerator /
            (longitudeIfdTag.values as IfdRatios).ratios[1].denominator,
        (longitudeIfdTag.values as IfdRatios).ratios[2].numerator /
            (longitudeIfdTag.values as IfdRatios).ratios[2].denominator,
      );
      double latitude = converter.getDecimalFromDegree(
        (latitudeIfdTag.values as IfdRatios).ratios[0].numerator /
            (latitudeIfdTag.values as IfdRatios).ratios[0].denominator,
        (latitudeIfdTag.values as IfdRatios).ratios[1].numerator /
            (latitudeIfdTag.values as IfdRatios).ratios[1].denominator,
        (latitudeIfdTag.values as IfdRatios).ratios[2].numerator /
            (latitudeIfdTag.values as IfdRatios).ratios[2].denominator,
      );
      if (longitudeRefIfdTag.printable.toUpperCase() == "W") {
        longitude = -longitude;
      }
      if (latitudeRefIfdTag.printable.toUpperCase() == "S") {
        latitude = -latitude;
      }
      return Pair(latitude, longitude);
    }
  }
  return null;
} catch (e, st) {
  LogService().log(e, st);
  return null;
}
```

```
}
Future<Map<String, IfdTag>?> getExifDataFromFile(String? filePath) async {
  try {
    if (filePath != null) {
      final File file = File(filePath);
      if (file.existsSync()) return await readExifFromFile(file);
    }
    return null;
  } catch (e, st) {
    LogService().log(e, st);
    return null;
  }
}
Future<DateTime?> getImageOriginalDateTime(
    {String? filePath, Map<String, IfdTag>? exifData}) async {
  if (exifData == null && filePath != null) {
    exifData = await getExifDataFromFile(filePath);
  }
  if (exifData != null) {
    final IfdTag? dateTimeOriginalTag = exifData["EXIF DateTimeOriginal"];
    final IfdTag? dateTimeDigitizedTag = exifData["EXIF DateTimeDigitized"];
    final IfdTag? dateTimeTag = exifData["Image DateTime"];
    final List<DateTime?> results = [
      parseExifDateTime(dateTimeOriginalTag.toString()),
      parseExifDateTime(dateTimeDigitizedTag.toString()),
      parseExifDateTime(dateTimeTag.toString()),
    ];
    return results.firstWhereOrNull((element) => element != null);
  }
  return null;
}
bool _checkIosCameraFolder(AssetPathEntity element) {
  final folderName = element.name.replaceAll(" ", "").toLowerCase();
  final List<String> allPhotosIOSFolder = ["Recent", "AllPhoto"];
  for (int i = 0; i < allPhotosIOSFolder.length; ++i) {
```

```
    if (folderName
        .contains(allPhotosIOSFolder[i].replaceAll(" ", "").toLowerCase())) {
      return true;
    }
  }
  return false;
}
Future<Uint8List?> getThumbByAssetId(String assetId, {int size = 300}) async {
  return (await AssetEntity.fromId(assetId))!
      .thumbnailDataWithSize(ThumbnailSize(size, size));
}
Future<void> reCreateScanDataEntity() async {
  final ScanDataEntity? scanData = await ScanDataTable().getDataById(1);
  if(scanData == null) {
    await PreferenceManager.setIsNeedToRescanGallery(false);
    return;
  }
  final List<StoryEntity?> allStories = await StoryTable().getAllData();
  final UserEntity? currentUser = await UserTable().getCurrentUser();
  scanData.usedAssetsId = [];
  for (final StoryEntity? story in allStories) {
    if (story!.globalCreatorId != currentUser!.globalUserId) continue;
    if (!story.approved! && !story.deleted!) continue;
    Set<String> storyAssets = {};
    final List<Media?> storyMedias = story.getAllMedia();
    storyAssets = (story.extraData!['usedAssets'] as List<dynamic>?)
            ?.cast<String>()
            .toSet() ??
        {};
    for (final Media? media in storyMedias) {
      if (media == null) continue;
      for (final MediaContent mediaContent in media.mediaContent!) {
        if (mediaContent.assetId == null || mediaContent.assetId!.isEmpty) {
          continue;
        }
```

```
          storyAssets.add(mediaContent.assetId!);
        }
      }
      scanData.usedAssetsId.addAll(storyAssets.toList());
      story.extraData!.addAll({"usedAssets": storyAssets.toList()});
      await StoryTable().updateData(story, story.id);
    }
    scanData.lastScannedImageDate = 0;
    scanData.lastScanDate = 0;
    await ScanDataTable().updateData(scanData, scanData.id);
    await PreferenceManager.setIsNeedToRescanGallery(false);
  }
  Future<void> clearCache2() {
    return PhotoManager.clearFileCache();
  }
}
```

Лістинг файлу CreateStoryManual

```
class CreateStoryManual {
  CreateStoryManual({
    required this.mainBlockTemplateType,
    this.entity,
    this.globalStoryId,
    this.mainTemplateMedias,
    this.createdAt,
    this.historyDate,
    this.description,
    this.title,
    this.canvas,
    this.audio,
    this.templateQuoterType,
    this.medias = const [],
    this.mediasToDelete,
  });
  final List<Media?> medias; //Other story files
```

```
final int mainBlockTemplateType;
final List<int>? templateQuoterType;
final List<Media?>? mainTemplateMedias; //Files from main template
final List<Media>? mediasToDelete;
final String? description;
final DateTime? createdAt;
final DateTime? historyDate;
final String? title;
final Media? audio;
final Media? canvas;
final String? globalStoryId;
final StoryEntity? entity;
static const int _maxBlocks = 32;
Future<StoryEntity?> createStory() async {
 try {
  final AuthRepository _authRepository = Get.find<AuthRepository>();
  final StoryRepository _storyRepository = Get.find<StoryRepository>();
  StoryEntity? storyEntity = StoryEntity();
  final UserEntity currentUser =
     (await _authRepository.getCurrentUserDb())!;
  if (entity != null) {
   storyEntity = entity;
   if (entity!.isAutogenerated! && entity!.title != title) {
    await _storyRepository.saveStoryTitleChangeHistory(
     StoryTitleUpdateHistoryEntity(
      storyId: entity!.id,
      newTitle: title,
      oldTitle: entity!.title,
      updatedTime: DateTime.now(),
     ),
    );
   }
  }
  storyEntity!.title = title;
  if (entity != null &&
```

```
    entity!.isAutogenerated! &&
    title != entity!.autogeneratedTitle) {
  try {
    Analytics().analyticsForChangeTitle(entity!.autogeneratedTitle, title);
  } catch (e, str) {
    LogService().log(e, str);
  }
}
storyEntity.createdAt = createdAt ?? DateTime.now();
storyEntity.approved = true;
storyEntity.updatedAt = DateTime.now();
storyEntity.historyDateGeneral = historyDate ?? DateTime.now();
storyEntity.approved = true;
storyEntity.globalCreatorId = currentUser.globalUserId;
storyEntity.creatorId = currentUser.id;
storyEntity.globalStoryId = globalStoryId;
storyEntity.data = await generateStoryData();
storyEntity.langCode = entity?.langCode ??
    await Get.find<ConfigRepository>().getLanguageCode();
StoryEntity? result;
storyEntity.synchronized = false;
if (entity == null) {
  storyEntity.compressed = false;
  result = await _storyRepository.addStory(storyEntity);
} else {
  storyEntity.compressed = storyEntity.fullyCompressed();
  if (storyEntity.isShared!) {
    storyEntity.isNeedToUpdate = true;
  }
  result = await _storyRepository.updateStory(storyEntity);
}
if (result == null) return entity;
result.locations = entity?.locations;
if (entity == null &&
    (result.locations == null || result.locations!.isEmpty)) {
```

```
  final LocationEntity? location = await getLocationFromFiles();
  if (location != null) {
    location.storyId = result.id;
    if (result.locations == null) {
      result.locations = [
        await _storyRepository.createOrUpdateLocation(location)
      ];
    } else {
      result.locations!
          .add(await _storyRepository.createOrUpdateLocation(location));
    }
  }
}
if (mediasToDelete != null && mediasToDelete!.isNotEmpty) {
  Set<String> storyAssets = {};
  storyAssets = (storyEntity.extraData!['usedAssets'] as List<dynamic>?)
          ?.cast<String>()
          .toSet() ??
      {};
  if (storyAssets.isNotEmpty) {
    for (final Media media in mediasToDelete!) {
      for (final MediaContent mediaContent in media.mediaContent!) {
        if (mediaContent.assetId == null ||
            mediaContent.assetId!.isEmpty) {
          continue;
        }
        storyAssets
            .removeWhere((element) => element == mediaContent.assetId);
      }
    }
    storyEntity.extraData!["usedAssets"] = storyAssets.toList();
  }
}
await LogService().logRealtime(
```

```
        "Editor::StartBackup after edit for ${result.id} BackupAllowed:
${BackupService().backupAllowed}");
      if (!BackupService().backupAllowed) {
        BackupService().displayBackupIntroDialog();
      } else {
        BackupService().startBackup();
      }
      return result;
    } catch (e, str) {
      LogService().log(e, str);
      rethrow;
    }
  }
  Future<StoryData> generateStoryData() async {
    final List<BaseStoryBlock> listBaseStoryBlock = <BaseStoryBlock>[
      await generateStoryBlockMain(),
      TextStoryBlock(text: description),
    ];
    if (medias.isNotEmpty) {
      listBaseStoryBlock.addAll(_generateStoryRandomBlocks());
    }
    final StoryData storyData = StoryData();
    storyData.blocks = listBaseStoryBlock;
    return storyData;
  }
  Future<LocationEntity?> getLocationFromFiles() async {
    Pair<double, double>? location;
    for (int i = 0; i < mainTemplateMedias!.length; ++i) {
      if (mainTemplateMedias![i] != null) {
        location = await PhotoManagerService().locationFromFile(
            filePath: mainTemplateMedias![i]!.getMediaContent()!.getUrl());
        if (location != null) break;
      }
    }
    if (location == null) {
```

```
    for (int i = 0; i < medias.length; ++i) {
      location = await PhotoManagerService()
          .locationFromFile(filePath: medias[i]!.getMediaContent()!.getUrl());
      if (location != null) {
        break;
      }
    }
  }
  if (location != null) {
    final LocationEntity? locationEntity =
        (await LocationService().makeSearch(location.left, location.right))
            .locationEntity;
    if (locationEntity != null) {
      return locationEntity;
    }
  }
  return null;
}
List<BaseStoryBlock> _generateStoryRandomBlocks() {
  final List<BaseStoryBlock> generatedStoryBlocks = [];
  int totalLength = 0;
  final List<Media?> firstThirtyTwo = [];
  final Random rnd = Random();
  removeFirstItemsCount(int count) {
    totalLength -= count;
    firstThirtyTwo.removeRange(0, count);
  }
  MediaBlockType getRandomBlockType(int max) {
    final int res = rnd.nextInt(1000) % max + 1;
    return getMediaBlockTypeByIndex(res, withOutCircle: true);
  }
  MediaDataStoryBlock getStoryBlock(MediaBlockType type) {
    final List<Media?> temp = [];
    final int countAvailableItems =
        MediaDataStoryBlock.getCountOfMedias(type);
```

```
    for (int i = 0; i < countAvailableItems; ++i) {
      temp.add(firstThirtyTwo[i]);
    }
    removeFirstItemsCount(countAvailableItems);
    final List<Media?> listOfMedias = [];
    for (final Media? element in temp) {
      listOfMedias.add(element);
    }
    return MediaDataStoryBlock(mediaBlockType: type, data: listOfMedias);
  }
  if (medias.isNotEmpty) {
    if (medias.length > _maxBlocks) {
      for (int i = 0; i < _maxBlocks; ++i) {
        firstThirtyTwo.add(medias[i]);
      }
    } else {
      firstThirtyTwo.addAll(medias);
    }
    totalLength = firstThirtyTwo.length;
    while (totalLength > 0) {
      int res = 0;
      if (totalLength >= 4) res = rnd.nextInt(1000) % 4 + 1;
      if (totalLength == 3) res = rnd.nextInt(1000) % 3 + 1;
      if (totalLength == 2) res = rnd.nextInt(1000) % 2 + 1;
      if (totalLength == 1) res = 1;
      final MediaBlockType mediaBlockType = getRandomBlockType(res);
      generatedStoryBlocks.add(getStoryBlock(mediaBlockType));
    }
    if (medias.length > _maxBlocks) {
      generatedStoryBlocks.add(generateMorePhotosBlock());
    }
  }
  return generatedStoryBlocks;
}
MorePhotosDataStoryBlock generateMorePhotosBlock() {
```

```
  final List<Media?> tempMedias = [];
  for (final element in medias.getRange(_maxBlocks, medias.length)) {
    tempMedias.add(element);
  }
  return MorePhotosDataStoryBlock(tempMedias);
}
Future<MainDataStoryBlock> generateStoryBlockMain() async {
  final MainDataStoryBlock mainDataStoryBlock = MainDataStoryBlock();
  mainDataStoryBlock.template = getMainBlockTemplateType();
  mainDataStoryBlock.canvasData = canvas;
  mainDataStoryBlock.audioData = audio;
  mainDataStoryBlock.media = mainTemplateMedias!.first;
  mainDataStoryBlock.blocks = generateMainDataStoryBlocks();
  return mainDataStoryBlock;
}
List<MediaDataStoryBlock> generateMainDataStoryBlocks() {
  if (mainBlockTemplateType == 1) return generateMainDataStoryBlocksStream();
  if (mainBlockTemplateType == 2) return generateMainDataStoryBlocksBadge();
  if (mainBlockTemplateType == 3) return generateMainDataStoryBlocksMosaic();
  return [];
}
List<MediaDataStoryBlock> generateMainDataStoryBlocksStream() {
  final List<Media?> listOfMedia = createListOfMediaStream();
  final List<MediaDataStoryBlock> blocks = [];
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: MediaBlockType.LIST,
    data: isListOfNulls(listOfMedia) || mainTemplateMedias.isNullOrEmpty()
        ? []
        : listOfMedia,
  ));
  return blocks;
}
List<MediaDataStoryBlock> generateMainDataStoryBlocksMosaic() {
  final List<MediaDataStoryBlock> blocks = [];
  blocks.add(MediaDataStoryBlock(
```

```
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![0]),
    data: [null],
  ));
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![1]),
    data: createListOfMedia(quoterIndex: 1),
  ));
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![2]),
    data: createListOfMedia(quoterIndex: 2),
  ));
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![3]),
    data: createListOfMedia(quoterIndex: 3),
  ));
  return blocks;
}
List<MediaDataStoryBlock> generateMainDataStoryBlocksBadge() {
  final List<MediaDataStoryBlock> blocks = [];
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![0]),
    data: createListOfMedia(quoterIndex: 0),
  ));
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![1]),
    data: createListOfMedia(quoterIndex: 1),
  ));
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![2]),
    data: createListOfMedia(quoterIndex: 2),
  ));
  blocks.add(MediaDataStoryBlock(
    mediaBlockType: getMediaBlockTypeByIndex(templateQuoterType![3]),
    data: createListOfMedia(quoterIndex: 3),
  ));
```

```
  return blocks;

List<Media?> createListOfMediaStream() {
  final List<Media?> media = [];
  for (int i = 1; i < mainTemplateMedias!.length; ++i) {
    if (mainTemplateMedias![i] != null) media.add(mainTemplateMedias![i]);
  }
  if (media.isEmpty) return mainTemplateMedias!.sublist(1, 9);
  return media;}
List<Media?> createListOfMedia({required int quoterIndex}) {
  final List<Media?> media = [];
  int _start = 0;
  if (mainBlockTemplateType == 2) _start = (quoterIndex * 4) + 1;
  if (mainBlockTemplateType == 3 && quoterIndex > 0) {
    _start = ((quoterIndex - 1) * 4) + 1;
  }
  for (int i = 0;
      i <
          MediaDataStoryBlock.getCountOfMediasByInt(
              templateQuoterType![quoterIndex]);
      ++i) {
    media.add(mainTemplateMedias![_start + i]);
  }
  return media;
}
MainBlockTemplate getMainBlockTemplateType() {
  switch (mainBlockTemplateType) {
    case 1:
      return MainBlockTemplate.ONE_BIG_OTHER_SMALL;
    case 2:
      return MainBlockTemplate.ONE_BIG_FEW_UNDER;
    case 3:
      return MainBlockTemplate.MOSAIC;
    default:
      return MainBlockTemplate.UNKNOWN;}}}
```

ДОДАТОК Г ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІНОЇ РОБОТИ

Завідувач кафедри ПЗ, д. т. н., професор _____ О.Н. Романюк

Науковий керівник, к.т.н. доцент кафедри ПЗ _____ Г.Б. Ракитянська

Опонент, к.т.н. доцент кафедри ОТ_____ В.В. Богомолов

Нормоконтроль, к. т. н., доцент кафедри ПЗ _____ Г.Б. Ракитянська

Виконавець, студент групи 2ПІ-21м _____ Б.В. Прус

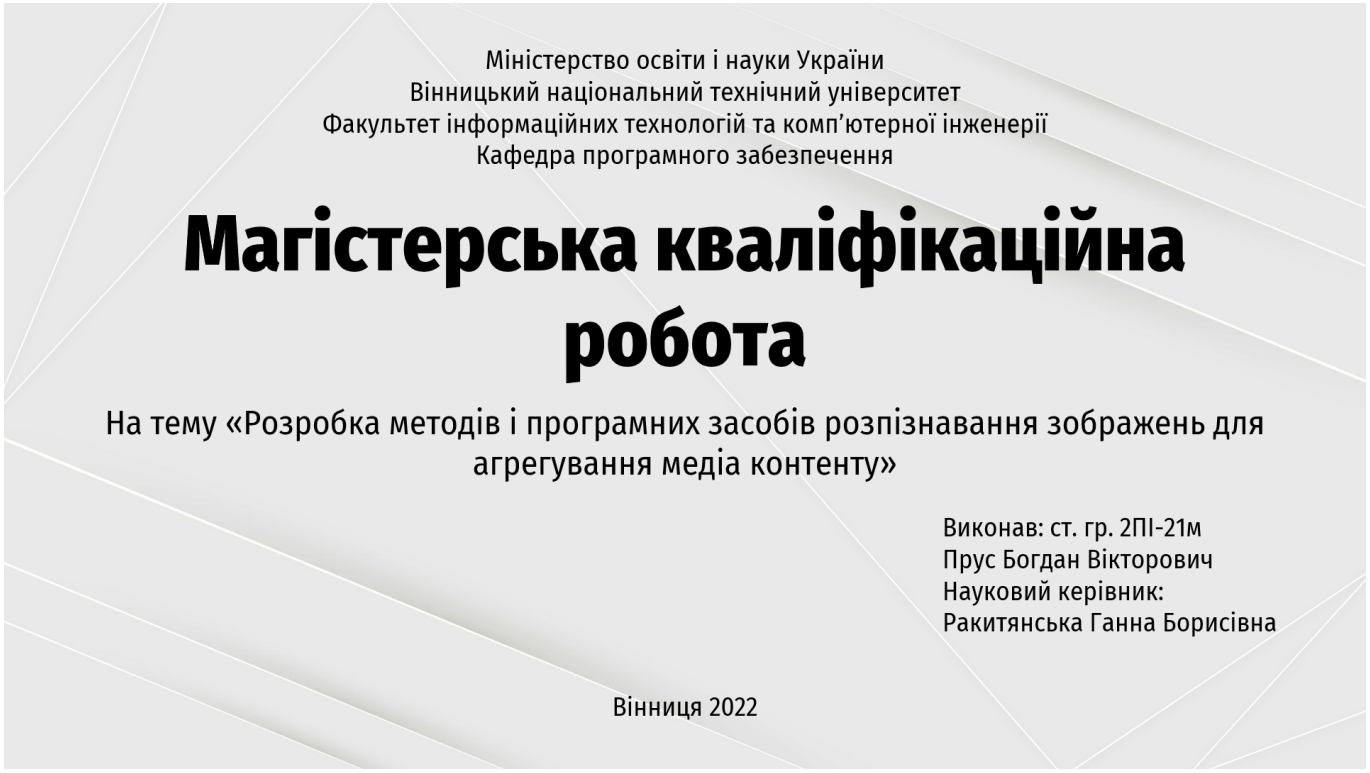


Рисунок Г.1 – Назва роботи

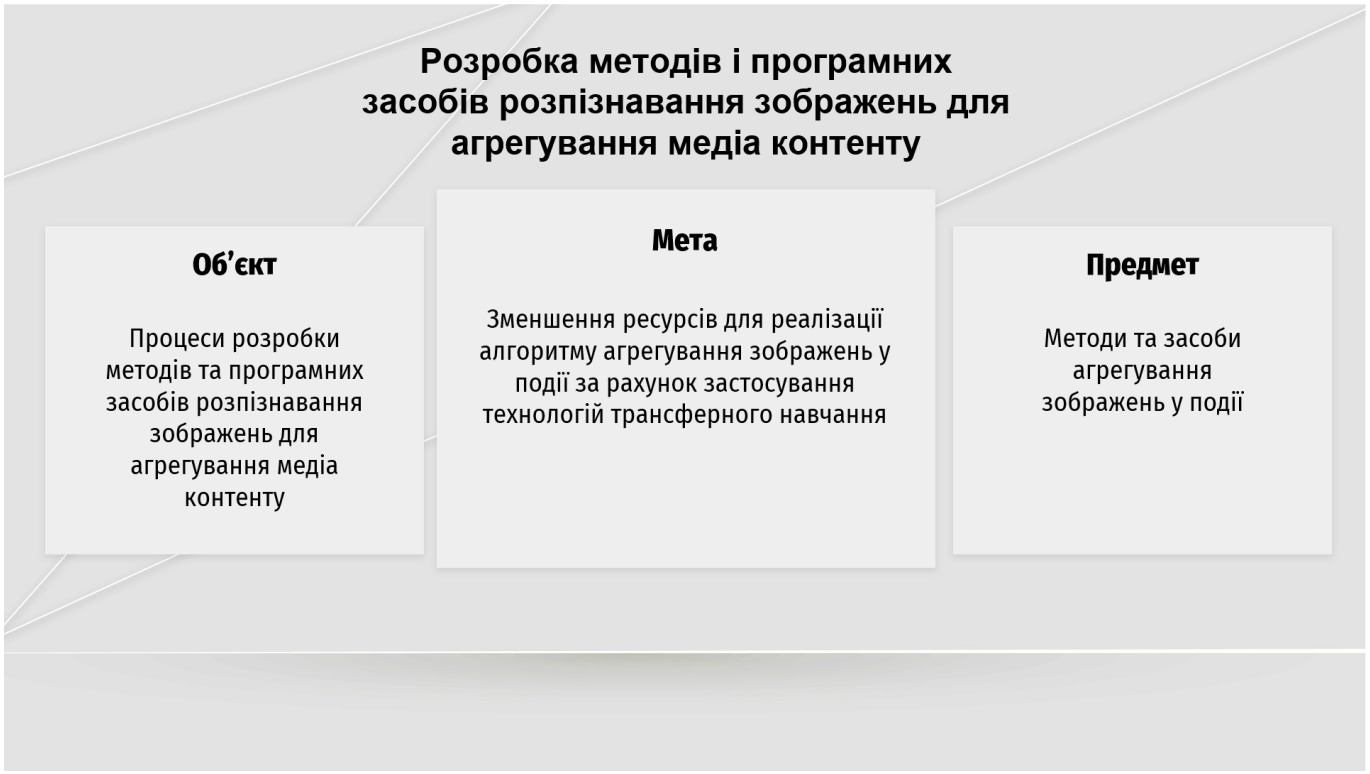


Рисунок Г.2 – Мета, об'єкт і предмет дослідження

Рисунок Г.3 – Етапи розробки

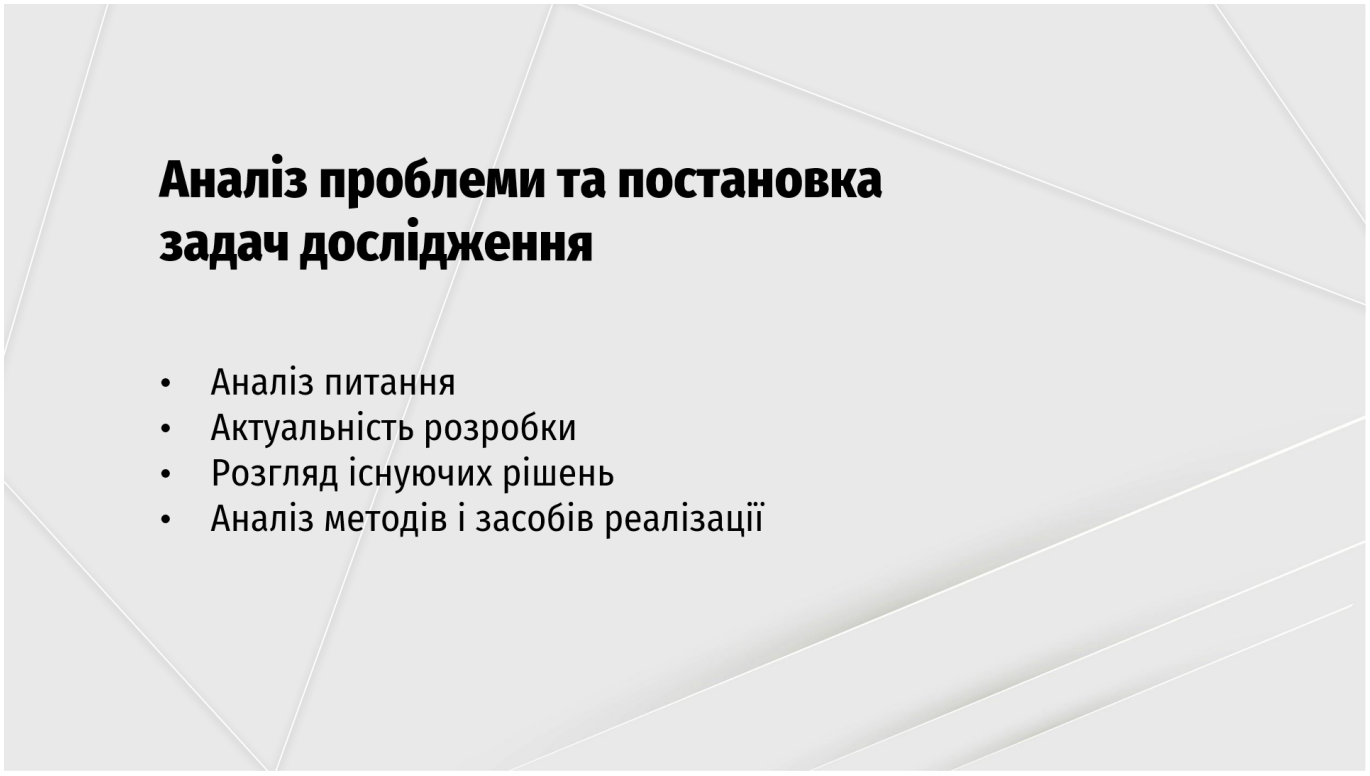


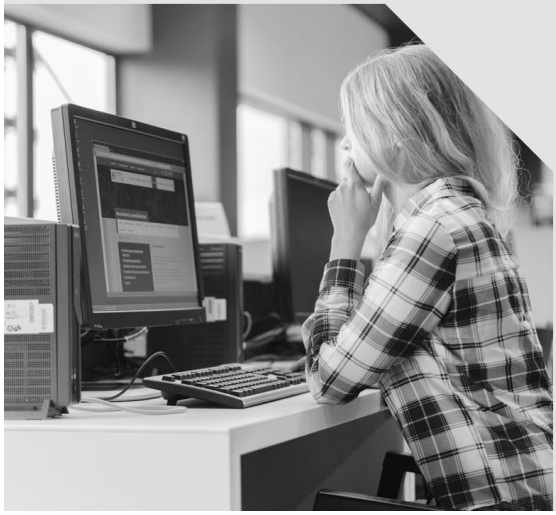
Рисунок Г.4 – Аналіз проблеми та постановка задач дослідження

Розробка структури мобільного додатку та алгоритму агрегування медіа контенту

- Спроектовано діаграму варіантів використання
- Розроблено архітектуру мобільного додатку
- Розроблено логічну та фізичну структури мобільного додатку
- Спроектовано алгоритм агрегування медіа контенту
- Спроектовано базу даних
- Розроблено інтерфейс та дизайн мобільного додатку

Рисунок Г.5 – Розробка структури мобільного додатку та алгоритму агрегування медіа контенту

Розробка програмного забезпечення



- Аналіз засобів реалізації програмного забезпечення
- Аналіз середовища розробки
- Програмна реалізація мобільного додатку
- Програмна реалізація алгоритму агрегування медіа контенту

Рисунок Г.6 – Розробка програмного забезпечення

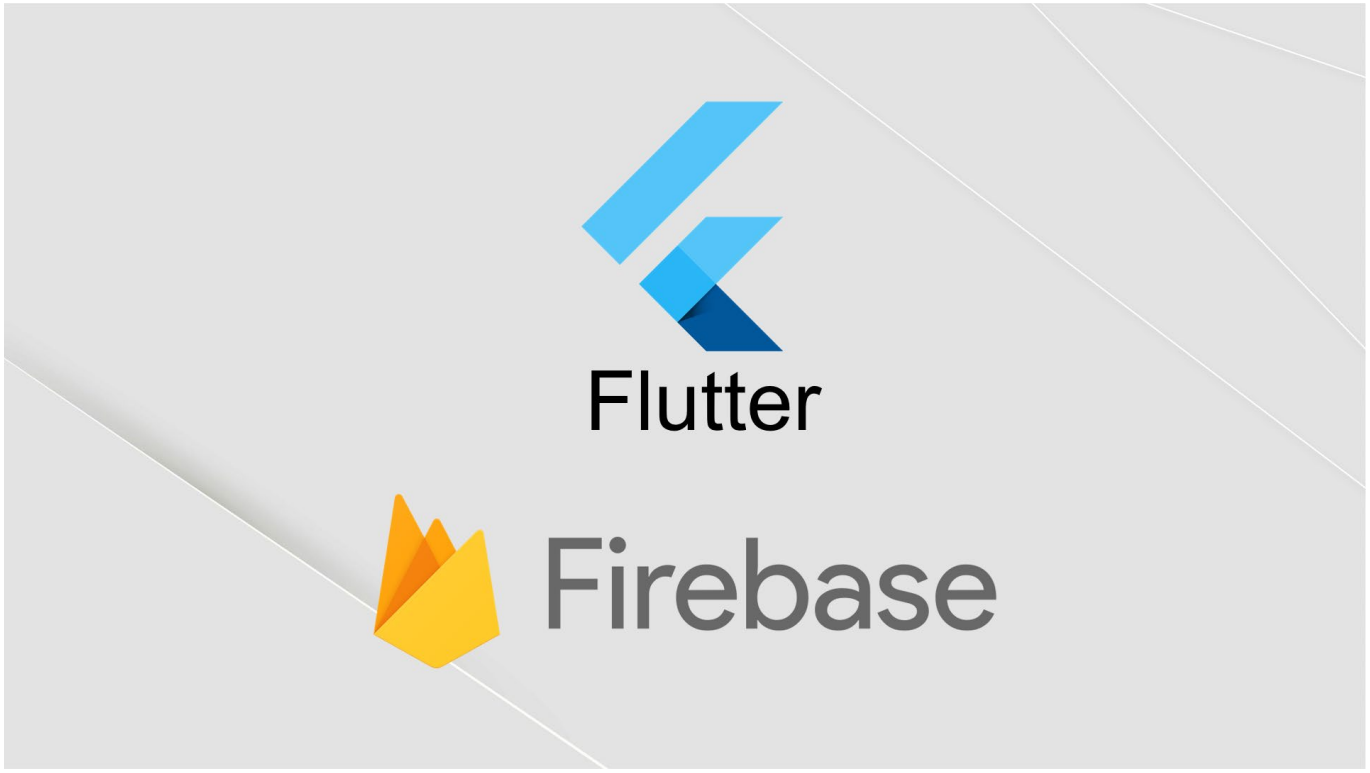


Рисунок Г.7 – Використані технології Flutter та Firebase

Рисунок Г.8 – Середовища розробки

Тестування

Проведено тестування ПЗ методами білої та чорної скриньок.
Проведене тестування довело, що програмне забезпечення працює стабільно, ефективно та виконує функції згідно визначених вимог. Основна ціль проекту була досягнута.

Рисунок Г.9 –Тестування

Економічна частина

1. Оцінено комерційний потенціал розробки.
2. Здійснено прогноз витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи.
3. Здійснено прогноз комерційних ефектів від реалізації результатів розробки.
4. Проведено розрахунок ефективності вкладених інвестицій та періоду їх окупності

Рисунок Г.10 – Економічна частина

Висновки

1. Проведено аналіз предметної області
2. Здійснено огляд аналогів на ринку
3. Спроектовано аріхтектуру мобільного додатку
4. Здійснено розробку мобільного додатку
5. Проведено тестування чорної та білої скриньки
6. Розраховано економічну доцільність реалізації проекту

Рисунок Г.11 – Висновки

Публікації

Результати роботи доповідалися на Міжнародній науково-практичній Інтернет-конференції 2022 року «Електронні інформаційні ресурси: створення, використання, доступ».’

РАКИТЯНСЬКА, Г.; Прус, Б.. РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ ДЛЯ АГРЕГУВАННЯ МЕДІА КОНТЕНТУ. НТКП ВНТУ. Факультет інформаційних технологій та комп'ютерної інженерії, Україна, 2022.

Рисунок Г.12 – Публікації