Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення мобільного додатку творчого арт-простору за допомогою фреймворку Flutter”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-1
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп'ютерна інженерія»
_____ Стрюк В. Є.
«____» _____ 2025 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Рівень вищої освіти *бакалавр*
Галузь знань *12 "Інформаційні технології"*
Спеціальність *123 Комп'ютерна інженерія*
Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Стрюк Владиславу Євгенійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення мобільного додатку творчого арт-простору за допомогою фреймворку Flutter*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№48-02 від 17.01.2025 року*

2. Строк подання студентом роботи до захисту *. .2025 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення мобільного додатку творчого арт-простору за допомогою фреймворку Flutter*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « » 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	24.05.2025 р.	

Студент _____ **Стрюк В. Є.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Мелешко Є. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Стрюк В. Є. Програмне забезпечення мобільного додатку творчого арт-простору за допомогою фреймворку Flutter. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення мобільного додатку творчого арт-простору з використанням фреймворку Flutter.

Метою роботи є створення кросс-платформного мобільного застосунку, який дозволяє отримати доступ до арт-простору, використовуючи фреймворк Flutter.

Результат роботи – програмна реалізація мобільного додатку творчого арт-простору за допомогою фреймворку Flutter.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено адаптивний інтерфейс, який відповідає вимогам UI/UX та має інтуїтивний та простий у використанні, для користувача, алгоритм роботи. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на пристроях з операційними системами Android та iOS.

Програму розроблено з фреймворком Flutter з використанням мови програмування Dart.

Ключові слова: комп'ютерна інженерія, мобільний додаток, Flutter, Dart, арт-простір, креативність, зображення, API.

ABSTRACT

Striuk V.E. Software of the mobile application of creative art space using the Flutter framework. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

This qualification bachelor's thesis has developed software of a mobile application for a creative art space using the Flutter framework was developed.

The purpose of the work is to create a cross-platform mobile application that allows you to access the art space using the Flutter framework.

The result of the work is a software implementation of a mobile application for a creative art space using the Flutter framework.

In the process of working on the implementation of the system, a study of existing methods, algorithms and software tools was carried out. The software itself was developed and implemented, and all its components were described.

An adaptive interface has been developed that meets UI/UX requirements and has an intuitive and easy-to-use algorithm for the user.

The program can be used on devices with Android and iOS operating systems.

The program is developed with the Flutter framework using the Dart programming language.

Keywords: computer engineering, mobile application, Flutter, Dart, art space, creativity, images, API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

API – Application Programming Interface

BLoC – Business Logic Component

UI – User Interface

UX – User Experience

ВСТУП

Актуальність теми дипломної роботи полягає в необхідності розробки мобільного додатку, здатного надати доступ до творчого арт-простору. В сучасному перенасиченому цифровому середовищі для більшості користувачів необхідно забезпечувати більш зручний, швидкий та простий доступ до візуального контенту.

Існують багато різноманітних сервісів що надають можливість до взаємодії з творчою візуальною складовою інтернет-простору, проте більшість з них мають або складний інтерфейс та відсутність адаптації під мобільні девайси, або містять перенасиченість функціоналу та не безкоштовний контент. Звідси можна винести доступність різноманітних API-сервісів, які дозволяють реалізувати необхідний функціонал для задоволення аудиторії творчим арт-простором.

Тому розробка програмного забезпечення, яке забезпечить зручну архітектуру зв'язку мобільного додатку з API-сервісом, є важливим напрямком досліджень. Така система, в зв'язку зі своєю простотою, зручністю та відкритим вільним доступом до відкритою бази даних візуального контенту сприятиме популяризації цифрового мистецтва, спрощуватиме роботу дизайнерів, митців, контент-креаторів, а також задовольняє потреби пересічних користувачів у якісному візуальному контенті.

Мета й завдання дослідження. Метою роботи є програмне забезпечення мобільного додатку творчого арт-простору за допомогою фреймворку Flutter.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Дослідження існуючих методів реалізації мобільного додатку творчого арт-простору.

– Розробка мобільного додатку творчого арт-простору.

– Програмна реалізація мобільного додатку творчого арт-простору.

Практична цінність отриманих результатів полягає в тому, що розроблений мобільний додаток за допомогою фреймворку Flutter дозволяє отримати доступ до творчого арт-простору.

Отже, розробка та впровадження мобільного додатку творчого арт-простору за допомогою фреймворку Flutter є актуальною задачею, яка вимагає постійного вдосконалення і розробки нових рішень та вирішувалася у цій кваліфікаційній роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Програмне забезпечення призначене для забезпечення зручного доступу користувача до творчого арт-простору візуального контенту за допомогою мобільного застосунку. Система реалізована з можливостями фреймворку та бібліотек, що дозволяють переглядати зображення з відкритої платформи Pixabay, здійснювати пошук за категоріями та переглядати обрані зображення у зручному інтерфейсі.

Застосунок працює на основі взаємодії користувача з API сервісом Pixabay через інтерфейс програми, що надає доступ до великого обсягу зображень з відкритою ліцензією. Користувач може обрати категорію, переглянути сітку зображень, що їй відповідають, ознайомитись із детальною інформацією про кожне з них та завантажити на пристрій. Такий підхід дозволяє швидко знаходити релевантний візуальний контент для творчих, навчальних чи інформаційних цілей.

Основна мета системи – забезпечити простий, швидкий та інтуїтивно зрозумілий доступ до якісних зображень на мобільному пристрої. Завдяки використанню фреймворку Flutter, застосунок є кросплатформеним та може працювати на пристроях з різними операційними системами.

Важливою перевагою такого рішення є можливість використання зображень без необхідності ручного пошуку у браузері, що значно економить час користувача. Крім того, інтерфейс програми оптимізований для мобільних пристроїв, що забезпечує комфорт під час використання. Це особливо актуально для дизайнерів, ілюстраторів, студентів та всіх, хто шукає візуальне натхнення у зручному форматі.

1.2 Область застосування

Програмне забезпечення мобільного додатку творчого арт-простору, може застосовуватися у різних сферах, де є потреба в оперативному доступі до якісного візуального контенту.

Однією з ключових сфер використання є графічний та веб-дизайн. Завдяки можливості швидко знаходити та переглядати зображення з відкритої платформи Pixabay, користувачі можуть надихатися ідеями для створення контенту, макетів, банерів або ілюстрацій без потреби у складному пошуку через браузери або платні ресурси.

Ще однією важливою сферою застосування є освітній процес. Застосунок може бути корисним для студентів, викладачів та учнів, які шукають тематичні зображення для презентацій, навчальних матеріалів або візуального супроводу проєктів. Простий та інтуїтивний інтерфейс робить його зручним для використання навіть тими, хто не має досвіду роботи з професійними графічними ресурсами.

Також система може застосовуватись у сфері соціальних мереж та створенню контенту. Блогери, SMM-спеціалісти та автори можуть оперативно знаходити ілюстрації для дописів, історій або візуального оформлення своїх публікацій, використовуючи лише мобільний пристрій.

У сфері творчості та цифрового мистецтва застосунок може бути джерелом візуального натхнення для художників, фотографів та ілюстраторів, які шукають референси або просто переглядають візуальні ідеї відповідно до заданих категорій.

Загалом, використання такого мобільного застосунку дозволяє спростити доступ до ліцензованих зображень, підвищити ефективність роботи з візуальним контентом та мінімізувати витрати часу на пошук графічних матеріалів у повсякденній діяльності.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Pinterest – є візуально орієнтованою соціальною платформою, основною функцією якої є організація, збереження та пошук зображень, ідей та іншого медіаконтенту у вигляді так званих «пінів». Користувачі можуть створювати тематичні віртуальні дошки (boards), на які додають піни відповідно до власних інтересів чи проєктів. Платформа забезпечує персоналізовані рекомендації на основі попередньої активності користувача, що реалізується завдяки сучасним системам рекомендацій і кластеризації контенту.

Рисунок 2.1 – Логотип застосунку Pinterest

Pinterest активно використовується як інструмент для пошуку натхнення у сферах мистецтва, дизайну, моди, інтер’єру, кулінарії тощо. Платформа також підтримує бізнес-функції, включаючи створення портфоліо, просування товарів

та послуг, аналітику взаємодії з контентом і можливість співпраці над спільними дошками. Користувачі можуть обмінюватися повідомленнями, коментувати піни, а також інтегрувати Pinterest із зовнішніми сайтами та інтернет-магазинами.

Завдяки інструментам візуального пошуку, системі тегування та аналітичним можливостям, Pinterest є ефективною платформою для систематизації ідей, вивчення трендів та розвитку особистого бренду або бізнесу у візуальних індустріях. Варто зазначити, що функціонал створення зображень або малювання безпосередньо у додатку відсутній, що відрізняє Pinterest від спеціалізованих графічних редакторів.

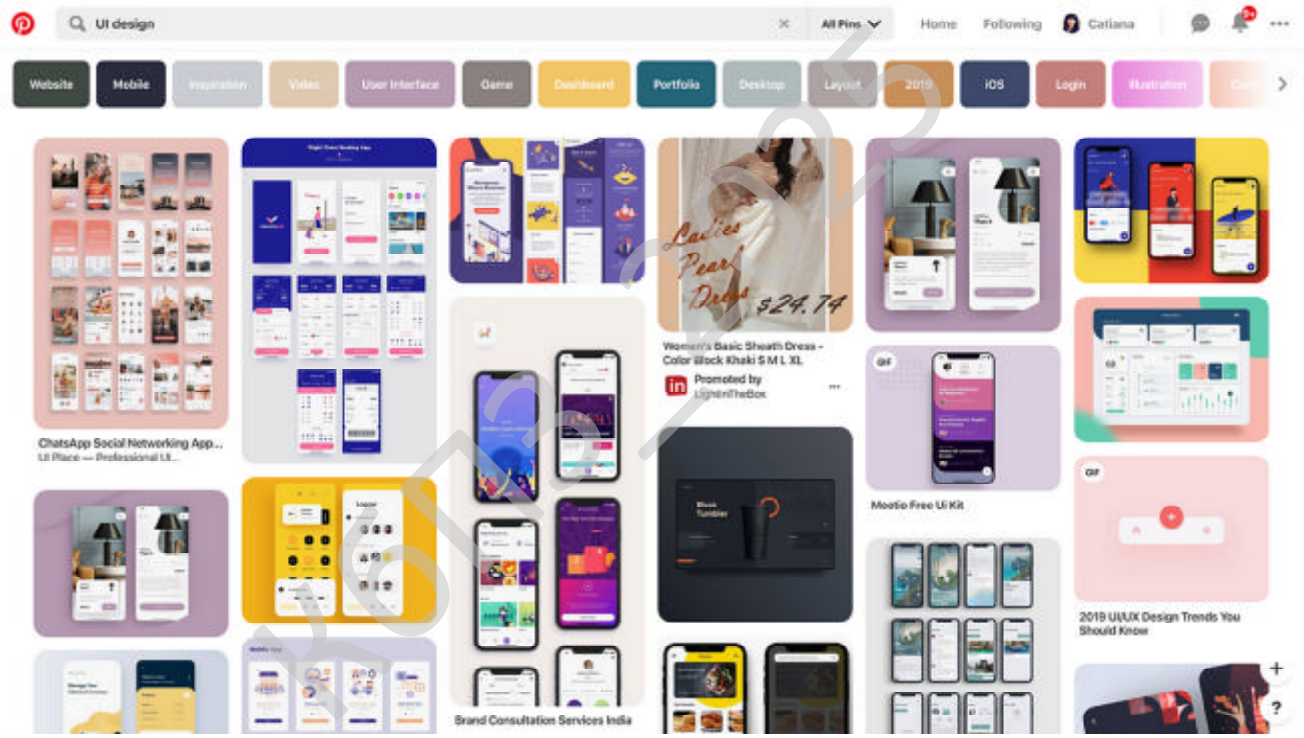

Рисунок 2.2 – Вигляд інтерфейсу застосунку Pinterest

DeviantArt – це одна з найбільших у світі онлайн-платформ для художників і поціновувачів мистецтва, яка функціонує як соціальна мережа та цифрова галерея. Платформа була заснована у 2000 році й на сьогодні об'єднує понад 75 мільйонів зареєстрованих користувачів і містить понад 550 мільйонів завантажених зображень. DeviantArt підтримує широкий спектр творчих

напрямів: цифровий живопис, традиційне малювання, фотографію, літературу, скульптуру, анімацію, поезію тощо.

Рисунок 2.3 – Логотип платформи DeviantArt

Основні можливості DeviantArt:

– Завантаження та організація робіт: користувачі можуть завантажувати свої твори у різних форматах, організовувати їх у персональні галереї та тематичні папки.

– Категоризація і теги: кожен твір можна віднести до певної категорії та позначити тегами, що полегшує пошук і систематизацію контенту.

– Взаємодія з аудиторією: користувачі можуть залишати коментарі, оцінки, додавати роботи до улюбленого, створювати власні колекції, а також отримувати зворотний зв’язок і критику від інших учасників спільноти.

– Групи та спільноти: існує можливість створювати та приєднуватися до тематичних груп за інтересами або напрямами мистецтва для обміну досвідом та спільних проєктів.

– Інструменти для монетизації: DeviantArt дозволяє художникам продавати свої роботи, пропонувати комісії, а також використовувати сучасні інструменти для демонстрації й захисту авторських прав.

– Підтримка спільноти: платформа активно просуває нових і маловідомих авторів, організовує конкурси, надає освітні ресурси та підтримує інклюзивне середовище для творчого розвитку.

DeviantArt є важливим майданчиком для презентації творчості, професійного зростання та налагодження контактів у міжнародній арт-спільноті. Варто зазначити, що платформа не містить вбудованих інструментів для створення малюнків (за винятком базового браузерного редактора Muro), а основний функціонал зосереджений на демонстрації, обговоренні й поширенні готових робіт.

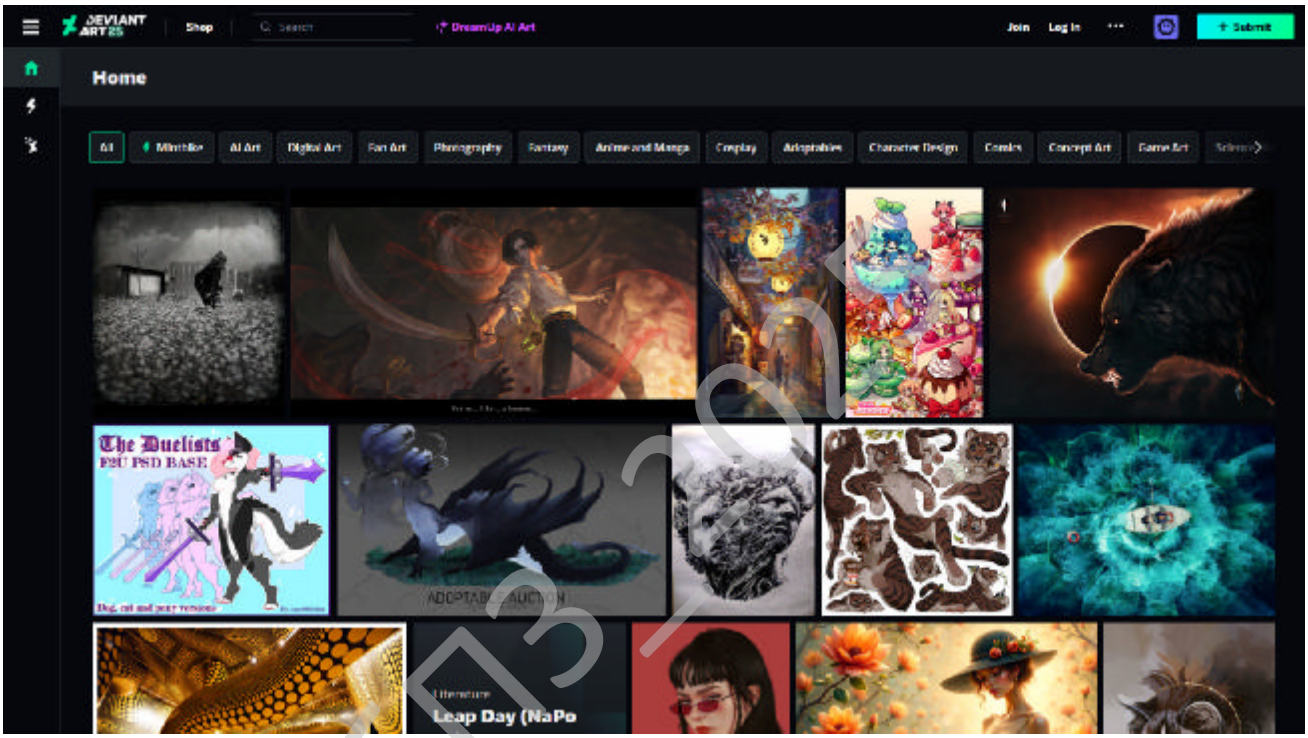

Рисунок 2.4 – Вигляд інтерфейсу платформи DeviantArt

ArtStation – це провідна онлайн-платформа для демонстрації портфоліо та просування творчих робіт, орієнтована переважно на професійних художників, ілюстраторів, 3D-моделерів, дизайнерів і розробників ігор у сферах ігор, кіно, медіа та розваг.

Рисунок 2.5 – Логотип програми ArtStation

Основні функції та можливості ArtStation:

– Портфоліо та презентація робіт: користувачі можуть розміщувати високоякісні зображення, відео, 3D-сцени, панорами 360°, а також додавати інформацію про свій досвід і проєкти. Платформа підтримує різноманітні формати контенту, що дозволяє комплексно представити творчість.

– Власний вебсайт: ArtStation автоматично генерує персональний портфоліо-сайт із власним доменом і захищеним HTTPS, який синхронізується з профілем користувача, що спрощує підтримку актуальності контенту.

– Аналітика: надає статистику переглядів портфоліо, що допомагає оцінити інтерес аудиторії і потенційних роботодавців.

– Продаж цифрових продуктів: платформа дозволяє продавати цифрові активи (3D-моделі, кисті, навчальні матеріали, референси) та друковані копії робіт через сервіс ArtStation Prints із високою якістю друку і логістикою під ключ. Автори отримують до 95% доходу.

– Спільнота та конкурси: ArtStation організовує тематичні конкурси (Challenges), що сприяють професійному розвитку, обміну досвідом і створенню нових проєктів.

– Навчальні ресурси: через ArtStation Learning користувачі отримують доступ до великої бібліотеки навчальних матеріалів від провідних фахівців індустрії.

– Пошук і рекрутинг: платформа має потужний пошуковий механізм для знаходження художників за навичками, локацією, доступністю, що широко

використовується рекрутерами та замовниками для пошуку професіоналів.

ArtStation виступає не лише як платформа для демонстрації творчості, але й як комплексна екосистема для розвитку кар'єри художників у цифрових індустріях. Вона забезпечує ефективний зв'язок між авторами і роботодавцями, сприяє монетизації творчості та професійному зростанню. Завдяки фокусу на якість контенту і професійній аудиторії, ArtStation є стандартом у галузях ігрової розробки, анімації та цифрового мистецтва.

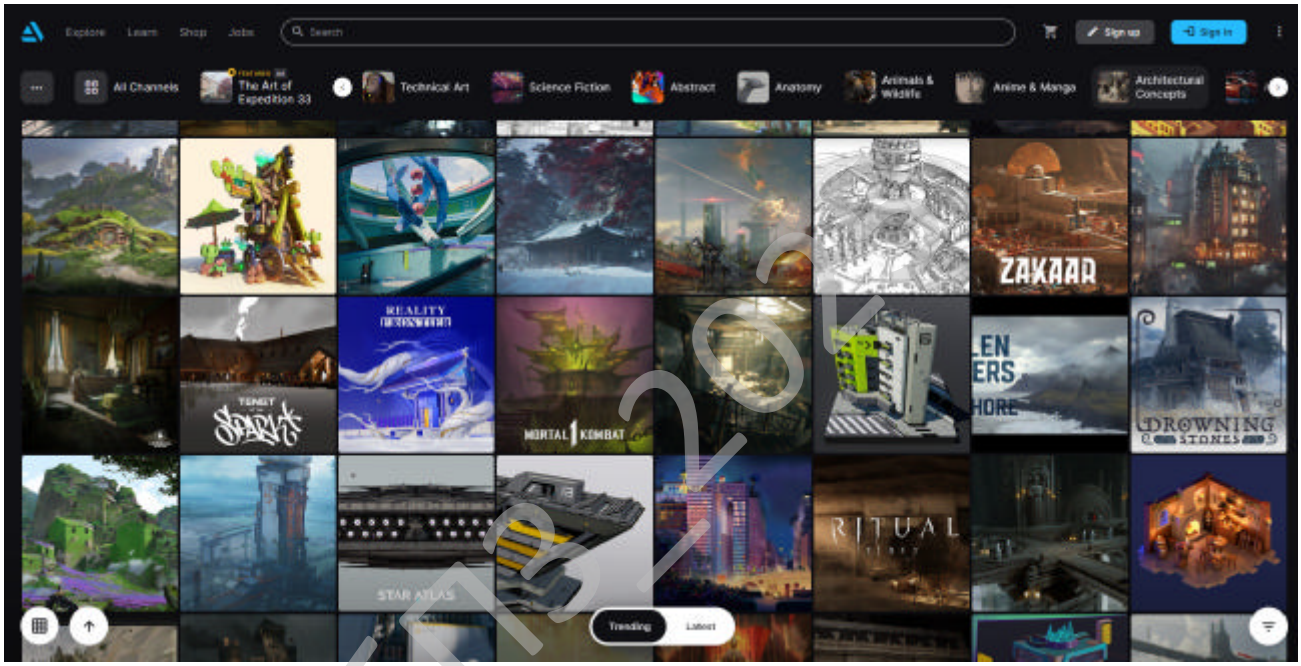

Рисунок 2.6 – Вигляд інтерфейсу програми ArtStation

Dribbble – це онлайн-платформа і спільнота для дизайнерів, ілюстраторів та інших креативних професіоналів, яка дозволяє демонструвати свої роботи, отримувати зворотний зв'язок і знаходити натхнення. Заснована у 2009 році, Dribbble швидко стала одним із провідних майданчиків для публікації коротких фрагментів проєктів, так званих «шотів» (shot), що представляють собою зображення, GIF-анімації або короткі відео, які ілюструють дизайнерські навички та ідеї.

Рисунок 2.7 – Логотип платформи Dribbble

Основні функції Dribbble включають:

– Публікацію і демонстрацію робіт: користувачі завантажують зображення високої якості (до 10 Мб), організують їх у портфоліо та отримують відгуки від спільноти.

– Пошук натхнення: платформа служить величезною галереєю ідей, де можна вивчати сучасні тренди, аналізувати техніки та розвивати власні навички.

– Соціальна взаємодія: можливість коментувати, оцінювати роботи, підписуватися на інших дизайнерів, обмінюватися досвідом та розширювати професійні контакти.

– Пошук роботи та замовлень: Dribbble використовується як ресурс для пошуку вакансій, фриланс-проектів і нових клієнтів, особливо для досвідчених дизайнерів з якісним портфоліо.

– Інструменти для створення портфоліо: платформа дозволяє створювати персональні сайти-портфоліо на базі профілю Dribbble з можливістю налаштування дизайну.

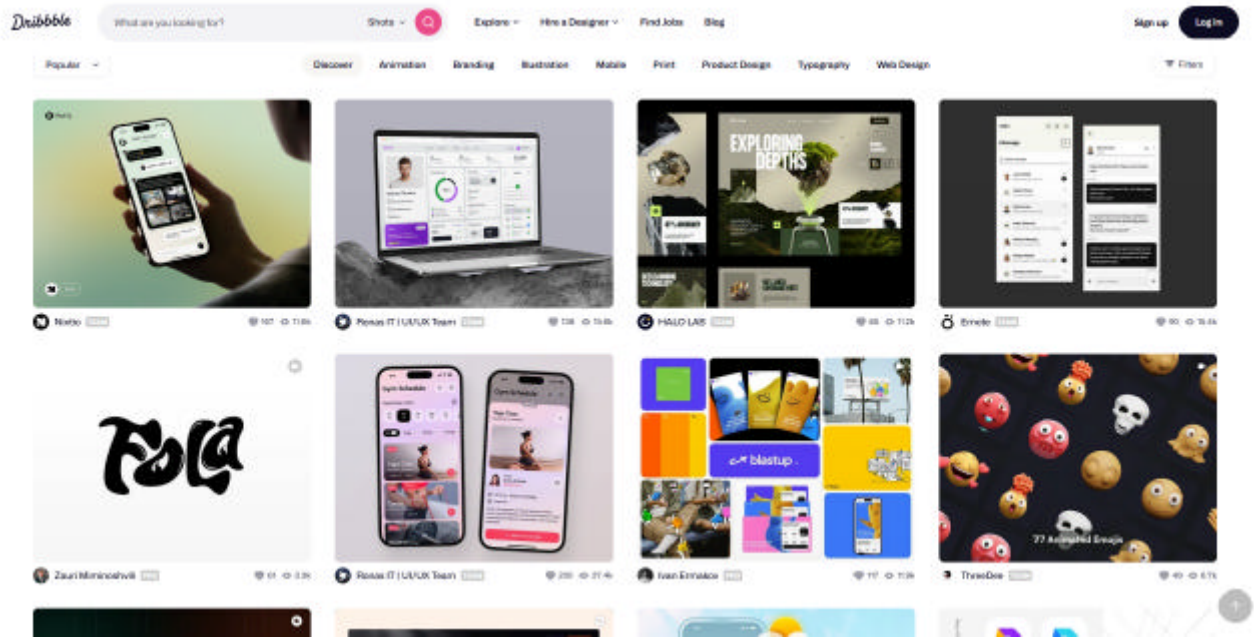

Рисунок 2.8 – Вигляд інтерфейсу платформи Dribbble

Dribbble відомий як місце, де народжуються дизайн-тренди, а також як професійне середовище для навчання та обміну досвідом. Платформа орієнтована на візуальну презентацію робіт і не містить вбудованих інструментів для створення зображень або малювання. Завдяки своїй активній спільноті та широкому охопленню Dribbble є важливим ресурсом для дизайнерів, які прагнуть розвивати кар'єру, знаходити нові можливості та підтримувати зв'язок із колегами по цеху.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

При розробці програмного забезпечення мобільного додатку творчого арт-простору було обрано кілька ключових технологій: середовище розробки Android Studio, мова програмування Dart, фреймворк даної мови – Flutter, сервіс надання ілюстрацій з безкоштовною ліцензію – Pixabay API, паттерн для оформлення архітектури та розділення логіки від інтерфейсу – BLoC, бібліотека для

оформлення запитів на API сервіс – dio, бібліотеки для збереження зображень на девайс – permission_handler, path_provider.

Середовищем для розробки проєкту обрано **Android Studio** - це офіційне інтегроване середовище розробки (IDE) для створення мобільних додатків. Програма базується на IntelliJ IDEA і надає розробникам всі необхідні інструменти для написання, тестування та налагодження мобільних додатків. Серед впливових можливостей додатку для реалізації проєкту, можна виділити:

– Інтелектуальний набір підказок в редакторі коду: доступна підтримка автодоповнення, підсвічування синтаксису та помилок, рефакторинг (можливість перейменовувати / переміщувати блоки коду / файли з автоматичною заміною всіх посилань / згадувань в проєкті), швидкий пошук коду в проєкті.

– Вбудовані емулятори Android: реалізує можливість компілювати, запускати та тестувати додатки на різних віртуальних пристроях з різними версіями Android, розмірами екранів і конфігураціями.

– Можливість інтеграції з системою контролю версій Git та вбудованими зручними адаптаціями консольних команд git bash у вигляді меню взаємодії.

– Тека плагінів, що дозволяє розширювати функціонал IDE за допомогою додаткових модулів, для покращення користуванням редактору.

Основною мовою, котрою написаний застосунок є **Dart**, яка орієнтована на створення клієнтських додатків. Вона використовується як основна мова для фреймворку Flutter, що дозволяє писати кросплатформені додатки. Мова надає наступні переваги у використанні:

– Простий та сучасний синтаксис: мова Dart схожа на C-подібні мови (Java, JavaScript, C#), що робить її легким для використання, в зв’язку з достатнім рівнем знань у галузі використання раніше вищеперерахованих мов.

– Підтримка асинхронних методів програмування, для зручної роботи з операціями котрі потребують багато-потокового завантаження, що особливо важливо при роботі з API та мережею.

– Компіляція Dart-коду в нативний код для різних операційних систем,

що забезпечує можливість реалізації єдиного формату додатку для мульти-платформ, що значно полегшує підтримку.

– Підтримка можливостей Hot Reload та Hot Restart, що дозволяються без повного перезапуску процесу компіляції або перезапустити додаток з початкової точки, витративши невелику кількість часу на очистку стану, або оновити поточний стан в процесі роботи в дебаг-режимі коду.

Замість написання дизайну нативним способом, використовується фреймворк **Flutter**, розроблений компанією Google для створення кросплатформених додатків з єдиною кодовою базою. Він дозволяє розробляти додатки для Android, iOS, Widnows, Linux, Web. Можна виділити ряд можливостей, корисних при розробці на Flutter:

– Flutter не використовує нативні компоненти платформи, а малює UI самостійно з бібліотек Material Components від Android та Cupertino від iOS, що забезпечує платформо-орієнтовану адаптивність та однаковий вигляд на різних операційних системах.

– Велика бібліотека шаблонів-віджетів для створення сучасних інтерфейсів.

– Hot Reload та Hot Restart: миттєве оновлення інтерфейсу без перезапуску додатку, що значно прискорює процес розробки та налагодження.

– Гнучкість в розробці, що з легкістю надає можливість створювати власні віджети та анімації.

– Активна спільнота що надає велику кількість пакетів, плагінів і документації, що постійно оновлюються.

Ресурс, що надає зображення з вільною ліцензію використання – **Pixabay API** – це безкоштовний сервіс, який надає доступ до величезної бібліотеки зображень, ілюстрацій, векторів і відео з відкритою ліцензією. API використовується для інтеграції великого обсягу ілюстрацій у додаток творчого арт-простору. Основні можливості:

– За допомогою query-параметру пошуку можна отримувати зображення,

що відповідають певній тематиці.

– Фільтрація за категоріями: фото, ілюстрації, вектори, відео.

– Доступ до метаданих: інформація про назву, автора, кількість взаємодій, набору тегів, тощо.

– Відповіді у форматі JSON, що мають структуру, зручну для обробки у мобільних додатках.

– Pixabay надає контент безкоштовно, що ідеально підходить для стартапів і невеликих проектів.

– Кількість запитів є обмеженою до 100 зображень на хвилину, чого достатньо для більшості невеликих додатків.

Для налагодженої архітектури проєкту була підключена зовнішня бібліотекаflutter_bloc. **BLoC** – це архітектурний патерн, який допомагає розділити бізнес-логіку додатку від інтерфейсу користувача. Особливо популярний у Flutter-розробці, та має відгалуження паттерну Cubit. Переваги використання:

– Модель реактивного програмування, що виконується бізнес-логікою реалізується через потоки, що дозволяє подіям на UI складовій реагувати на зміни стану.

– В даній архітектурі відбувається чітке розділення «обов'язків», де UI відповідає лише за відображення, а BLoC - за обробку даних і логіки.

– Логіку можна легко тестувати окремо від UI, як і UI може працювати без наявної логіки.

– Архітектурна впорядкованість, завдяки розділенню проєкту на незалежні частини, не дає проєкту втратити свою структуру при великій масштабованості.

Для зв'язку з API використовується **dio** - це популярна бібліотека HTTP-клієнта для Dart/Flutter, що забезпечує зручний і потужний інтерфейс для роботи з мережею. Основні можливості:

– Підтримка REST-запитів: GET, POST, PUT, DELETE, PATCH.

– Конфігуровані заголовки та параметри: легко додавати авторизацію,

токени, кукі.

– Перехоплювачі (Interceptors) дозволяють обробляти запити та відповіді глобально (наприклад, для логування чи обробки помилок).

– Завантаження та відправка файлів: підтримка multipart/form-data.

– Гнучке та зручне налаштування обробки різного виду помилок.

– Підтримка зв’язку з файловою системою девайсу, що дозволяє завантажувати та зберігати контент з мережі.

Аби отримати шлях на збереження зображень, без ручного прописування шляхів для всіх операційних систем, використовується **path_provider** – плагін для Flutter, який надає доступ до стандартних директорій файлової системи пристрою, де можна зберігати файли.

– Доступ до директорії завантажень та папки галереї.

– Завдяки кросплатформенності, працює як на Android, так і на iOS, так і на інших доступних операційних системах.

– Інтеграція з Dart I/O та dio дозволяє легко читати та записувати файли.

permission_handler – це бібліотека для Flutter, яка дозволяє запитувати у користувача дозволи на доступ до різних функцій пристрою (камера, сховище, локація тощо). Ряд функціоналу що слід передбачати при розробці:

– Необхідні дозволи можна динамічно запитувати під час роботи програми.

– В реальному часі відбувається визначення, чи надано дозвіл, чи відхилено або заблоковано.

– Користувач може змінити надані / заборонені дозволи в налаштуваннях телефону. Зміни відразу активуються в мобільному додатку.

– Відбуваєтсья врахування особливостей обох мобільних платформ: Android та iOS.

– Обробка різних типів дозволів: камера, мікрофон, геолокація, зберігання, сповіщення тощо.

2.3 Розгорнута постановка завдання

Основним призначенням і завданням програмного забезпечення, що розробляється, є мобільний додаток з наданням доступу до творчого арт-простору. Це передбачає, що користувач, увійшовши в додаток, отримує можливість вільного пошуку, перегляду та завантаження цифрових зображень високої якості. Для забезпечення зручності користування та ефективного доступу до контенту реалізовано інтуїтивно зрозумілий інтерфейс з адаптивним дизайном, який коректно відображається на різних розмірах екранів мобільних пристроїв.

Виведення інтерфейсу та взаємодія з ним реалізована за допомогою дерева вкладених віджетів у Flutter, а також використанням інших додаткових користувацьких бібліотек для покращення продуктивності, анімацій та UX. Отримання ілюстрацій відбувається завдяки надсиланню асинхронних запитів на отримання відповідей від безкоштовного API сервісу Pixabay, що дозволяє динамічно підвантажувати контент без затримок і перезавантажень.

Було обрано мову для розробки програмного забезпечення: Dart, з залежним від нього фреймворком – Flutter, та низки додаткових бібліотек. В якості ресурсу для отримання арт-ілюстрацій використовується Pixabay API. Архітектура зв’язку логіки з інтерфейсом та взаємодією з користувачем реалізована патерном BLoC. Для запитів на API в дію вступає бібліотека dio.

Користувацький інтерфейс передбачає собою навігаційну панель, головний екран категорій, екран зображень та екран деталей про ілюстрацію. Навігація реалізована таким чином, щоб користувач міг швидко перемикатися між основними розділами додатку, не втрачаючи контексту.

Головний екран категорій та екран зображень мають майже ідентичну структуру:

– Заголовок вікна з кнопкою повернення назад;

– Основне вікно з плитками для вибору категорій/зображень;

– Нижня навігаційна панель з кнопками вибору екрану категорій, екрану зі списком картинок та логотип додатку;

Екран з деталями про обрану ілюстрацію має інакшу структуру:

– Плаваючі кнопки для повернення назад та для завантаження картинки;

– Полотно перегляду зображення в повноекранному форматі та в оригінальній якості;

– Кнопка для відкриття нижньої панелі, що складається з: назви ілюстрації, автора публікації, типу жанру, кількості лайків, кількості завантажень, кількості збережень, переліку тегів;

Важливим аспектом функціонування всієї системи є швидкодія обробки API-запитів та їх динамічне виведення у візуально-адаптованому форматі для користувача. Для цього застосовуються методи кешування відповідей, оптимізації запитів та асинхронної обробки даних, що забезпечує плавний та безперебійний користувацький досвід навіть при повільному інтернет-з’єднанні. Крім того, реалізовано механізми обробки помилок і повідомлень про стан завантаження, що підвищує надійність та інформативність додатку.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Програмне забезпечення мобільного додатку творчого арт-простору працює за принципом надання доступу до бази даних різноманітних ілюстрацій від API-сервісу, через зручний та адаптивний інтерфейс.

Для створення ефективного для роботи мобільного додатку творчого арт-простору необхідно дотримуватись низки важливих принципів. Необхідним є визначення основних вимог до системи, її функціональність, швидкість обробки даних, незалежність від операційної системи та динамічність оновлень. Крім того, важливо зазначити, що обробка великого обсягу ілюстрацій потребує чи немалого інтернет-трафіку, тому оптимізація перегляду та завантажень ілюстрацій має пріоритетний вплив.

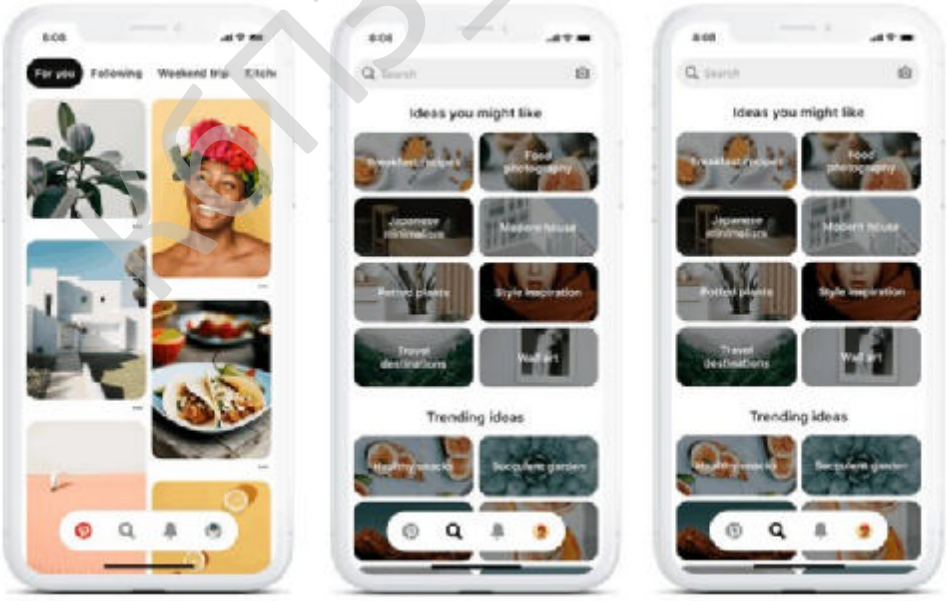

Рисунок 3.1 – Джерело прикладу зручного мобільного інтерфейсу онлайн-платформи Pinterest

Виходячи з того, що мобільний додаток націлений на надання доступу до творчого арт-простору (тобто програма є онлайн-галереєю з різноманітних образотворчих творінь), головним аспектом вважається візуальна складова. Надихнувшись існуючими проєктними рішенням та популярними сервісами, в дизайні програми застосовано найзручніші елементи інтерфейсу з попередньо вивчених рішень, котрі чітко розглядаються на рисунку 3.1.

З врахування найвпливовішого ефекту на користувацьку зручність, були застосовані наступні рішення для UI/UX додатку:

– Навігаційна панель: позиціонується внизу та містить мінімальну кількість кнопок, що зменшує шанс «промахнутись» при виборі пунктів;

– Екран категорій: замість простих інтерактивних карток з назвами категорій, створено віджет, що для наочного розуміння, яка категорія міститься під даним посиланням – відображає три демонстраційні прев'ю-ілюстрації. Для застосування категорії використовується інтуїтивно помітна кнопка, що міститься на кожній з карток;

– Екран списку зображень: натомість лінійному вертикальному списку, або сітки з односторонніми квадратами (як сервісі ArtStation) застосовано алгоритм побудови сітки з випадковими значеннями висоти та однаковими значеннями ширини та відступами. Даний спосіб позиціонування зображень чітко спостерігається на платформах DeviantArt, Pinterest;

– Екран деталей про зображення: ідея для дизайну полягала в реалізації повноекранного перегляду за замовчуванням, тому незалежно від горизонтальної або вертикальної орієнтації обраного зображення, алгоритм повноекранного перегляду розтягує зображення по висоті та центрує його. Кнопки взаємодій мають напівпрозору текстуру, тому не заважають перегляду ілюстрації. Нижня панель може відкриватись як при натисканні, так і при свайпі, і містить в собі простий для розуміння список детальної інформації про ілюстрацію;

До реалізованого адаптивного дизайну необхідним впровадженням є налаштування функціоналу. Модулі роботи програми виконується в декілька етапів:

1) Побудова інтерфейсу: реалізується за допомогою Flutter, що є кросплатформеним фреймворком для розробки інтерфейсів користувача, що дозволяє створювати адаптивні, продуктивні та візуально привабливі мобільні додатки з єдиною кодовою базою. У розробленому мобільному додатку Flutter відповідає за реалізацію всіх елементів інтерфейсу, зокрема навігаційної панелі, екрану категорій з віджетами, що відображають прев'ю-ілюстрації, сітки зображень із варіативною висотою елементів, а також повноекранного перегляду зображень.

Завдяки широкому набору готових віджетів, підтримці жестів і анімацій, Flutter забезпечує плавний та інтуїтивний зрозумілий для користувача UX. Крім того, він дозволяє легко адаптувати інтерфейс під різні розміри екранів, що важливо для мобільних пристроїв з різною роздільною здатністю;

2) Налаштування архітектури: BLoC – це архітектурний патерн, який забезпечує чітке розділення бізнес-логіки додатку від його інтерфейсу. Його суть полягає в тому, що всі події, ініційовані користувачем або системою, передаються у BLoC, де відбувається обробка логіки, а потім формується новий стан, який передається UI для відображення.

При побудові розробленого ПЗ, BLoC відповідає за керування станом екранів, обробку запитів до API, оновлення списків категорій та зображень, а також за логіку взаємодії з повноекранним переглядом. Використання BLoC дозволяє зручно орієнтуватись в архітектурі, а також зберігати завантажені фотографії та категорії, кешуючи їх на час використання додатку. Цей підхід особливо важливий для реалізації динамічного інтерфейсу, де зміни даних повинні миттєво відображатися на екрані;

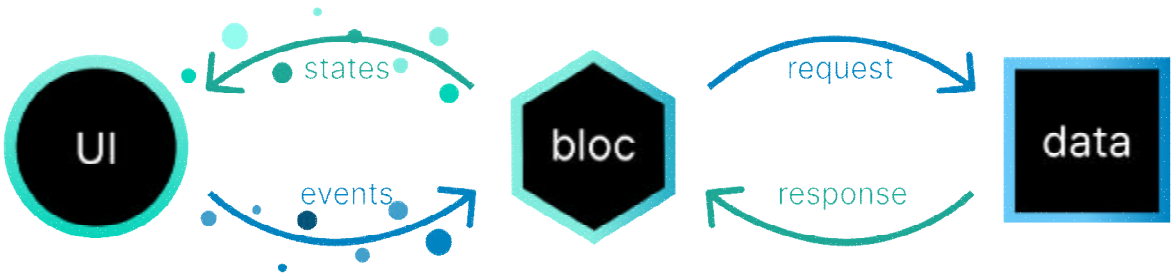

Рисунок 3.2 – Візуальна демонстрація взаємодії інтерфейсу з даними та архітектурою бізнес-логіки

3) Надсилання та отримання API-запитів: Використано бібліотеку Dio – потужну HTTP-клієнтська бібліотеку для Dart/Flutter, що забезпечує зручну та надійну роботу з мережею. Вона використовується для формування та відправки запитів до Pixabay API, отримання списку категорій, зображень та детальної інформації про них. Dio підтримує обробку помилок, таймаути, а також можливість додавання заголовків і параметрів запитів. Dio забезпечує ефективне завантаження даних, підтримуючи пагінацію та оптимізацію трафіку, що є критично важливим для роботи з великим обсягом ілюстрацій. Крім того, Dio дозволяє реалізувати кешування і завантаження файлів, що покращує швидкість роботи та зменшує навантаження на мережу;

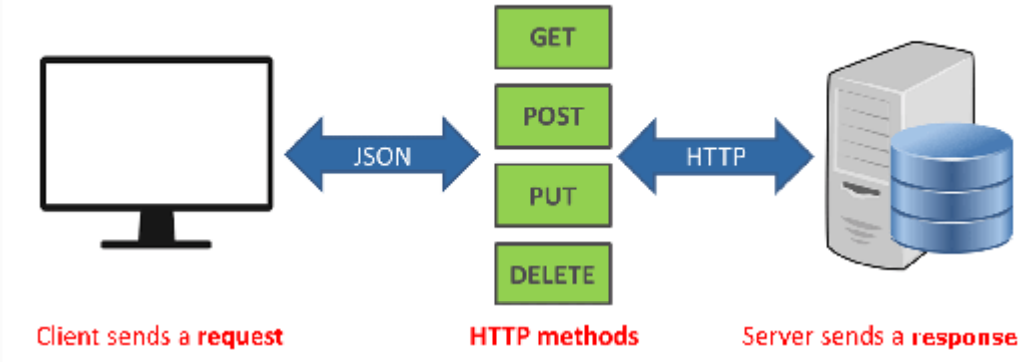

Рисунок 3.3 – Схема зв’язку клієнта та сервера через API

4) Зв’язок з провайдером бази даних зображень: Pixabay API є джерелом медіаконтенту для додатку. Це безкоштовний сервіс, який надає доступ до великої бази зображень з відкритою ліцензією. Через API здійснюється пошук і отримання ілюстрацій за категоріями, ключовими словами, а також отримання метаданих (автор, розмір, ліцензія). Pixabay API використовується для формування прев’ю категорій, наповнення сітки зображень та показу деталей обраного зображення. Інтеграція з цим API дозволяє забезпечити постійне оновлення контенту без необхідності зберігати великі обсяги даних локально;


```
{
"total": 4692,
"totalHits": 500,
"hits": [
    {
        "id": 195893,
        "pageURL": "https://pixabay.com/en/blossom-bloom-flower-195893/",
        "type": "photo",
        "tags": "blossom, bloom, flower",
        "previewURL": "https://cdn.pixabay.com/photo/2013/10/15/09/12/flower-195893_150.jpg",
        "previewWidth": 150,
        "previewHeight": 84,
        "webformatURL": "https://pixabay.com/get/35bbf209e13e9d2_640.jpg",
        "webformatWidth": 640,
        "webformatHeight": 360,
        "largeImageURL": "https://pixabay.com/get/ed6a99fd0a76647_1280.jpg",
        "fullHDURL": "https://pixabay.com/get/ed6a9369fd0a76647_1920.jpg",
        "imageURL": "https://pixabay.com/get/ed6a9361a9fd0a76647.jpg",
        "imageWidth": 4000,
        "imageHeight": 2250,
        "imageSize": 4731420,
        "views": 7671,
        "downloads": 6439,
        "likes": 5,
        "comments": 2,
        "user_id": 48777,
        "user": "Josch13",
        "userImageURL": "https://cdn.pixabay.com/user/2013/11/05/02-10-23-764_250x250.jpg",
    },
    {
        "id": 73424,
        ...
    },
    ...
]
}
```


Рисунок 3.4 – Приклад запиту з Pixabay API

3.2 Розробка структурної схеми

Структурною називають схему, яка відображає склад і взаємодію з управління частин розроблюваного програмного забезпечення. Для програмного забезпечення Flutter-застосунку мобільного додатку творчого арт-простору, структурна схема має вигляд як на рисунку 3.5;

Рендеринг візуальної частини інтерфейсу виконується фреймворком Flutter, що напряму залежить від backend-частини застосунку, котра реалізована через ініціалізацію запитів на API-сервіс. Тобто: інтерфейс динамічно підлаштовується до змін, викликаних запитами на API, котрі поділяються на три основних структурних частин.

Отримання масиву категорій виконується на самому початку програми, для надання вибору користувачу, котру з категорій ілюстрацій він бажає розглянути детальніше. Категорії завантажуються списком із плиток, що складаються з 3х демонстраційних зображень для попереднього перегляду вмісту категорії, назви категорії та кнопки переходу. Дана кнопка активує скрипт, що зберігає обрану категорію для подальшого перегляду (знаходячись в застосунку) та виконує операцію навігації користувача на екран з зображеннями даної категорії.

Якщо користувач не обрав жодної тематичної категорії, але перейшов на сторінку зі списком зображень – відбудеться API-запит на отримання масиву зображень за замовчуванням.

Якщо користувач обрав певну категорію з основного екрану – запускається скрипт з API-запитом на отримання масиву зображень даної категорії. В обох випадках, безумовно, після виконаного запиту користувач отримує список зображень, котрий буде динамічно завантажуватись по мірі просування списком нижче, доти – доки це можливо. При переході за посиланням на будь-яке з зображень запускається наступний скрипт.

Після вибору зображення, додаток отримує доступ до локально

збереженого масиву даних про зображення, та відповідно до відкритого посилання – виконується API-запит на отримання зображення за даним ID.

Завантаження зображення доступне на екрані деталей про зображення. Процес завантаження на девайс відбувається з дозволу користувача. Зображення буде збережено в теці завантажень, що визначається системою за замовчуванням.

Завдяки тому що структура Flutter-додатку реалізована за допомогою парадигми BLoC, користувач має можливість повторювати дії функціоналу, описані вище, в довільному порядку (передбаченому програмну) та отримувати комфортну динамічну зміну процесів на екрані, відповідно обраних дій.

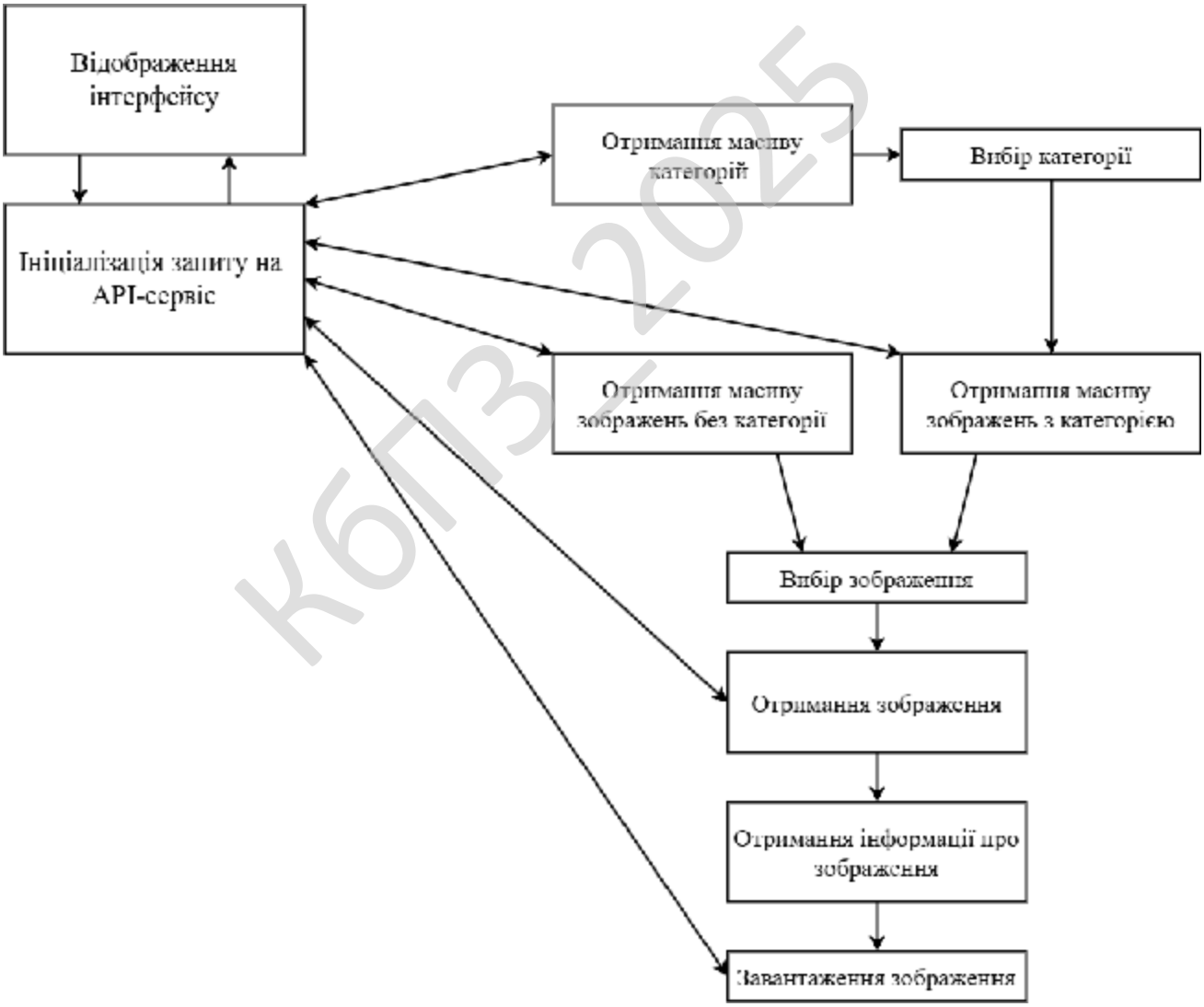

Рисунок 3.5 – Структурна схема

3.3 Розробка функціональної схеми

Функціональна схема або схема даних - схема взаємодії компонентів програмного забезпечення з описом інформаційних потоків, складу даних у потоках і вказівкою використовуваних файлів і пристроїв. На рисунку 3.6 функціональна схема системи демонструє логіку роботи мобільного додатку творчого арт-простору.

Функціонально вся система розбита на 3 логічні блоки: інтерфейс програми, API-запити та обробка зображень.

Блок «Інтерфейс програми» – це графічний інтерфейс, котрий динамічно відображає для користувача поточний стан програми, відносно подій які були ним викликані.

«Інтерфейс програми» включає в себе декілька модулів:

– Модуль Flutter-програма – основний рушій рендерингу віджетів та компонування програмних функцій у єдину систему, впорядковану архітектурою ПЗ. Складає основу для мобільного додатку.

– Модуль відображення отриманих категорій – відповідає за графічний функціонал сторінки вибору категорій, а саме: відображення категорій плитками що містять заголовок з кнопкою вибору та попереднім переглядом демонстраційних варіантів із трьох зображень даної категорії, виведення даних плиток у вигляді списку.

– Модуль відображення списку зображень – відповідає за адаптивний перегляд сітки зображень різних розмірів, що були попередньо завантажені.

– Модуль відображення конкретного зображення – виводить сторінку з повноекранною версією обраного зображення з перемикаючою нижньою панеллю мета-даних про іллюстрацію.

Блок «API-запити» – це функціональний набір інструкцій що дозволяє за заданим параметрами (категорія, ID, тощо) виконувати запит на віддалений сервіс на отримання масиву зображень або конкретного зображення.

«API-запити» включає в себе декілька модулів:

– Модуль API-запиту на отримання категорій – багато-потоковий запит на декілька категорій зображень, що полягає в отриманні масиву заголовків та по три варіанти ілюстрацій на кожну з категорій.

– Модуль API-запиту на отримання списку зображень – багато-потоковий запит на масив з 10-ти ілюстрацій обраної категорії та прописаною логікою, що передбачає отримання наступних 10-ти зображень доти, доки не в масиві не залишиться ілюстрацій.

– Модуль API-запиту на отримання одного зображення – одно-потоковий запит, що отримує обрану ілюстрацію в якіснішому форматі аніж для перед перегляду.

– Модуль API-запиту на отримання інформації про зображення – одно-потоковий запит, що отримує мета-дані про зображення з обраним ID.

– Модуль API-запиту на завантаження зображення – дозволяє стягти картинку з віддаленого сервісу у теку завантажень девайсу.

Блок «Обробка зображень» – набір модулів, націлених на логічний та коректний зв’язок блоку інтерфейсу та блоку API-запитів.

«Обробка зображень» включає в себе декілька модулів:

– Модуль оптимізованого завантаження категорій – виконує покрокове та оптимізоване відображення завантажених категорій, для швидкодії та зручності.

– Модуль оптимізованого завантаження списку зображень – виконує покрокове та оптимізоване відображення ілюстрацій по 10 одиниць при прокручуванні списку з плиток.

– Модуль адаптивного виведення зображення та інформації – реалізований для коректної взаємодії виведеного зображення разом з відповідними мета-даними.

– Модуль збереження зображення на девайс – виконує повторний запит на поточне зображення, для обробки посилання, що завантажиться як картинка.

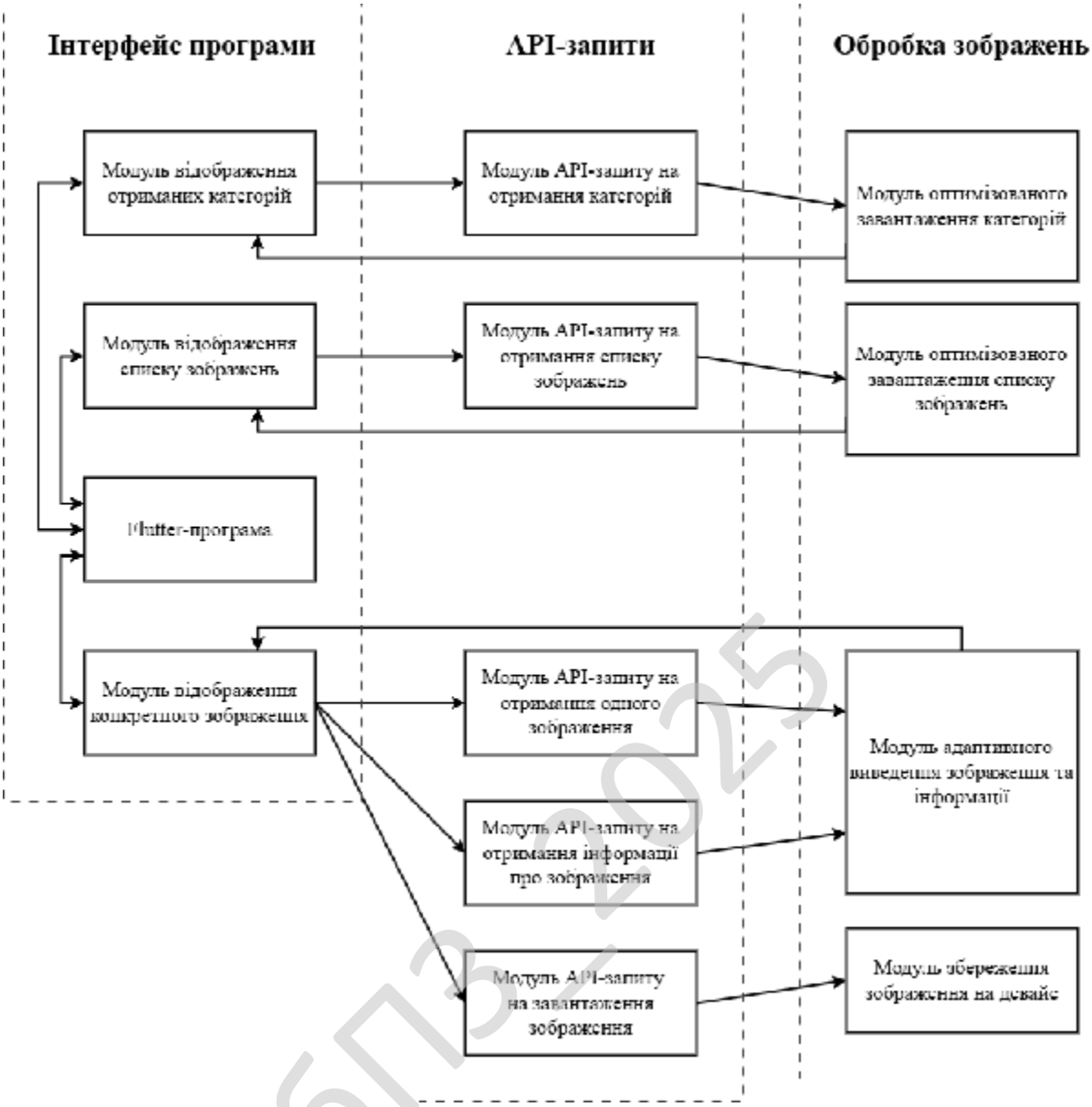

Рисунок 3.6 – Функціональна схема системи

3.4 Розробка діаграми процесів

Діаграма процесів у ПЗ — це графічне представлення послідовності дій, які виконуються в програмі для досягнення певного результату. Вона показує, як обробляються дані, які етапи проходять, та як взаємодіють різні компоненти системи.

Діаграма процесів, яка показує логіку системи та потік даних в ній, зображена на рисунку 3.7.

Після запуску програмного забезпечення з’являється головне вікно мобільного додатку творчого арт-простору. Головний екран має навігаційну панель для переходу між поточним екраном з можливістю обрати категорію та екраном вибору зображень. Без вибору, або з вибором категорії користувач має можливість перейти до сторінки з списком зображень. Після вибору будь-якого з зображень в списку – відкривається вікно з деталями про дану ілюстрацію. Даний екран дозволяє як переглянути саме зображення, так і деталі про нього, а також завантажити

Рисунок 3.7 – Діаграма процесів

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з таких пунктів:

– Завантаження інтерфейсу застосунку;

– API-запит: завантаження категорій зображень та завантаження зображень без категорії;

– Вибір пункту меню в навігаційній панелі;

– Перевірка: який пункт меню обрано? Якщо екран категорій – запустити підпрограму №1, якщо екран зображень – запустити підпрограму №2, якщо жодного – нічого не робити;

– Підпрограма 1: Екран списку категорій без вхідних даних;

– Підпрограма 2: Екран списку зображень з даними: КАТЕГОРІЯ;

– Виведення та збереження результатів;

На рисунку 4.2 наведено блок-схему підпрограми №1 – екран списку категорій. Її робота складається з таких пунктів:

– API-запит: завантаження категорій зображень;

– Оптимізоване, поступове завантаження категорій в інтерфейс;

– Перевірка, чи обрано категорію. Якщо так – запустити підпрограму №2, якщо ні – нічого не робити;

– Підпрограма 2: Екран списку зображень з даними: КАТЕГОРІЯ;

На рисунку 4.3 наведено блок-схему підпрограми №2 – екран списку зображень. Її робота складається з таких пунктів:

– Перевірка, чи було обрано категорію. Якщо так – категорія зберігається, якщо ні – API-запит: завантаження списку зображень за замовчуванням;

– API-запит: завантаження списку зображень за обраною [Категорія];

– Оптимізоване, поступове завантаження зображень в інтерфейс;

– Перевірка, чи обрано зображення. Якщо так – запустити підпрограму №3, якщо ні – нічого не робити;

– Підпрограма 3: Екран обраного зображення з даними: ID зображення;

На рисунку 4.4 наведено блок-схему підпрограми №3 – екран деталей зображення. Її робота складається з таких пунктів:

– API-запит: завантаження зображення під обраним ID та мета-даних;

– Виведення зображення на екран;

– Перевірка, яку обрано дію. Якщо жодної – нічого не робити,

якщо «Перегляд» – Адаптивний перегляд зображення при взаємодії,

якщо «Детальніше» – Виведення нижньої панелі з мета-даними,

якщо «Завантажити» – Завантажити зображення на девайс;

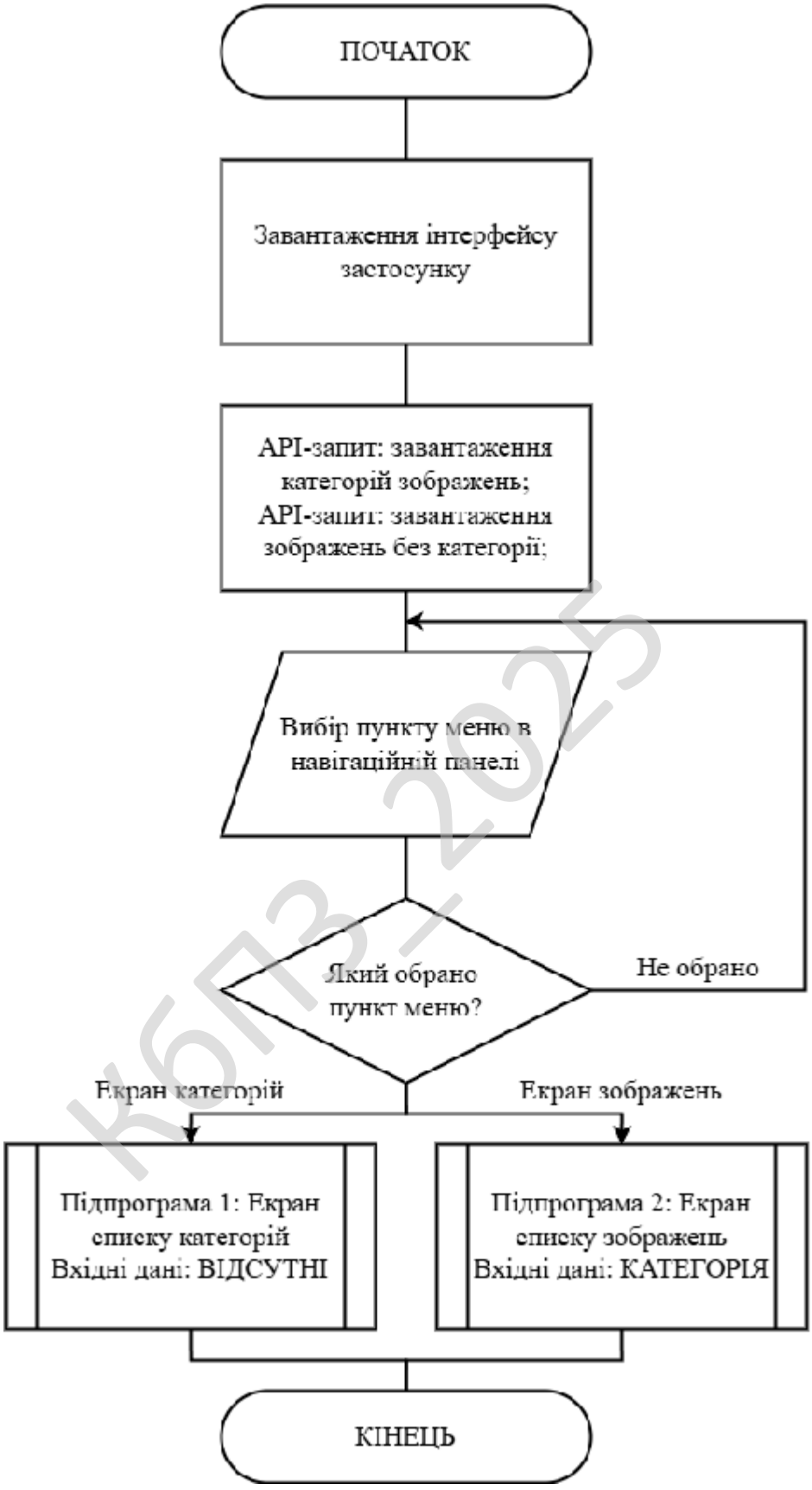

Рисунок 4.1 - Блок-схема основної програми

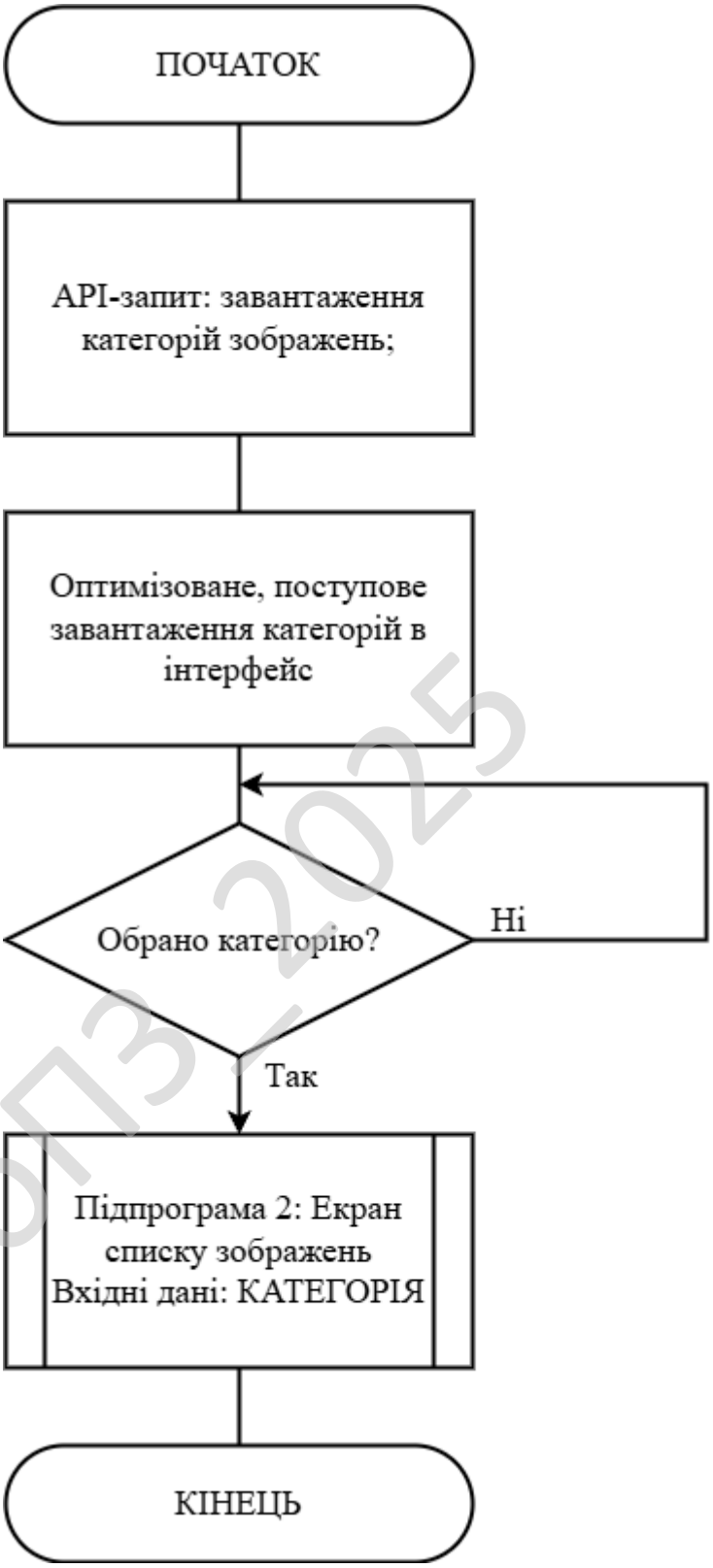

Рисунок 4.2 - Блок-схема підпрограми №1 – екран списку категорій

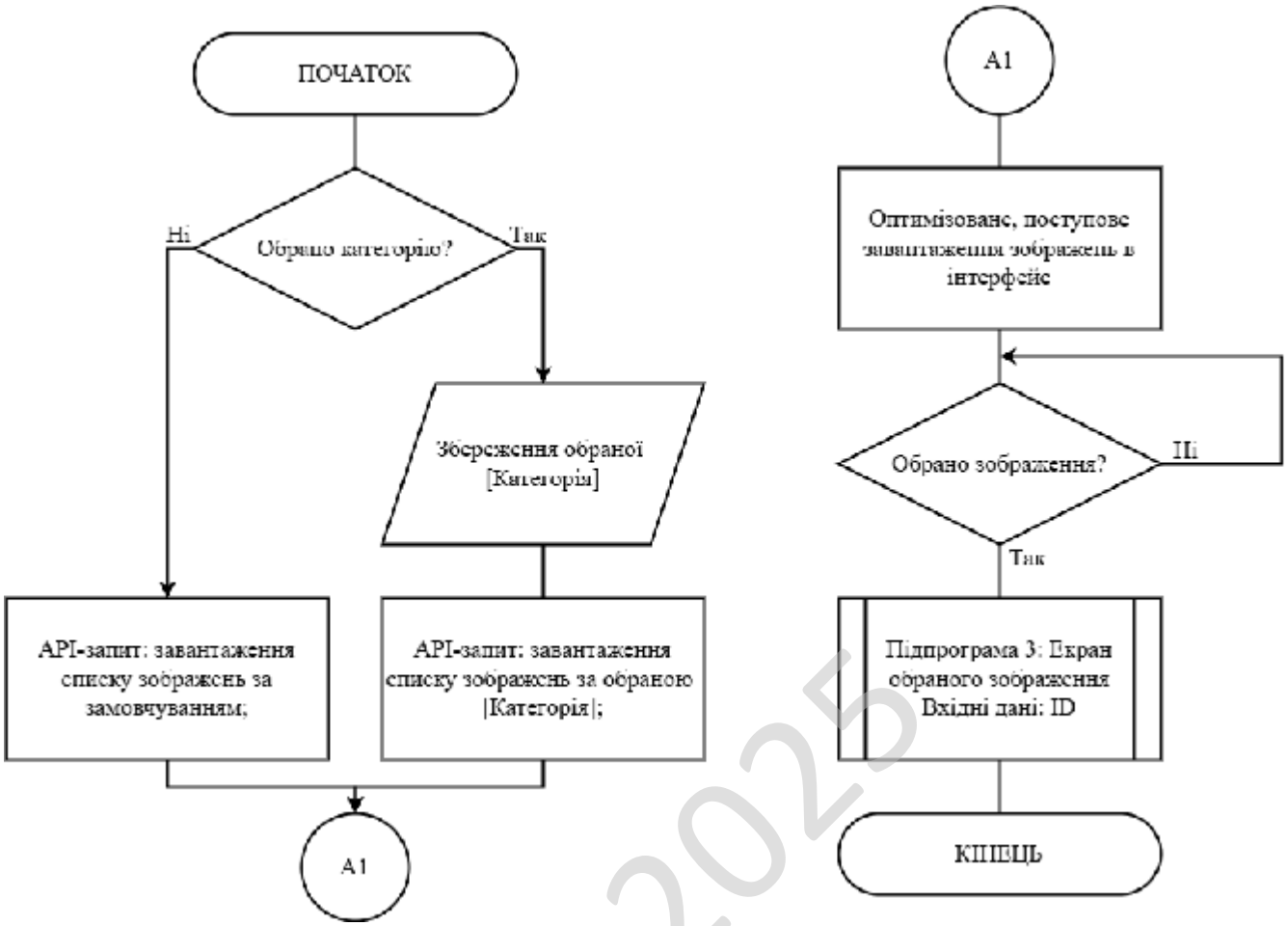

Рисунок 4.3 - Блок-схема підпрограми №2 – екран списку зображень

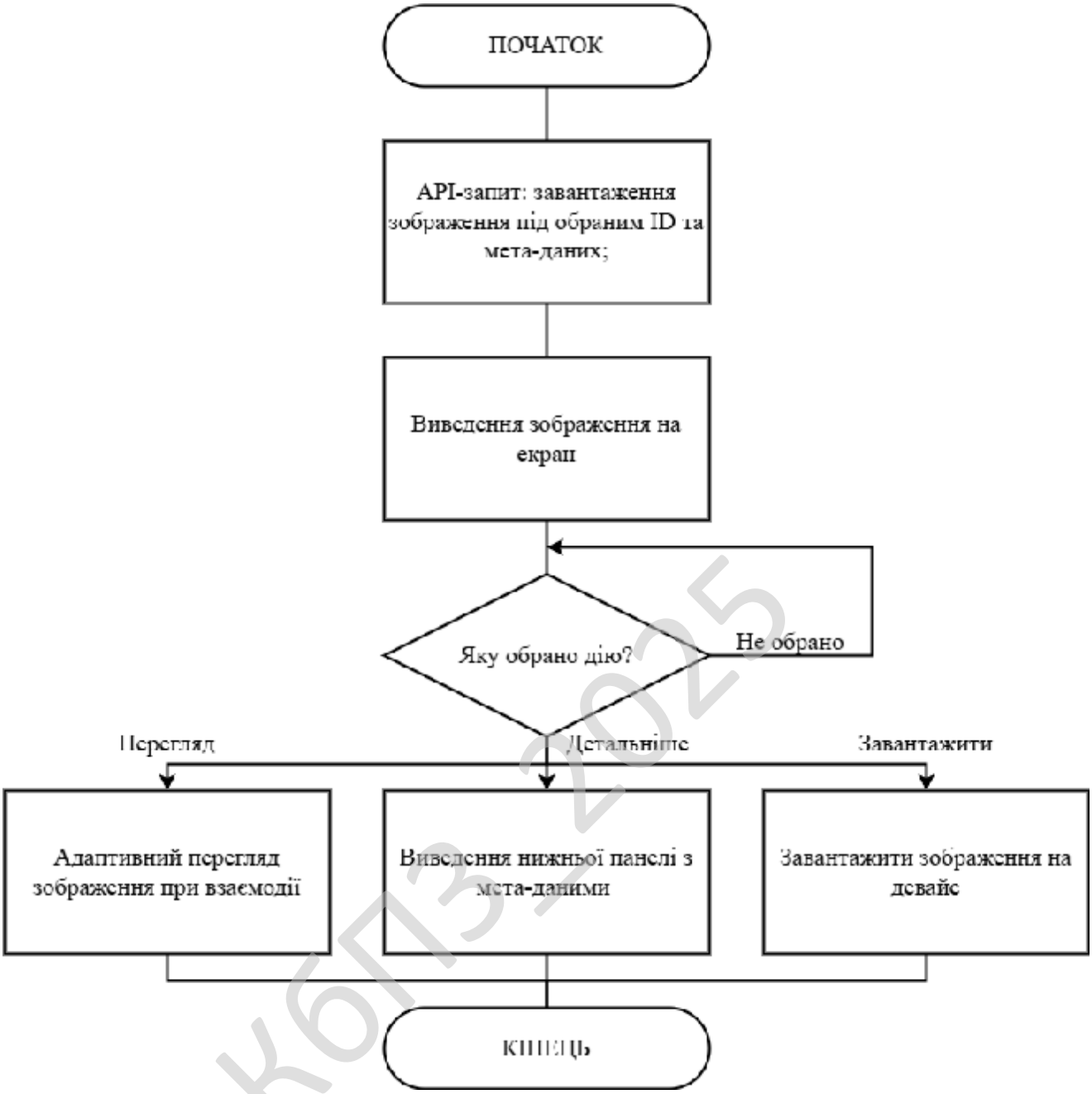

Рисунок 4.4 - Блок-схема підпрограми №3 – екран деталей зображення

Опис алгоритмів функціонування системи

Для реалізації функціоналу мобільного застосунку було реалізовано 1 універсальний алгоритм для отримання категорій та зображень, 1 алгоритм завантаження ілюстрацій на девайс та 2 алгоритми що відповідають за розподілення отриманих категорій / зображень на відповідних екранах як на frontend частині – так і на backend частині.

Також використано низку бібліотек, кожна з яких відповідає за окремий аспект роботи системи: отримання та відображення зображень, керування станами та бізнес-логікою компонентів, взаємодію з файловою системою та правами доступу, а також забезпечення сучасного користувацького інтерфейсу. В результаті компонування всіх бібліотек до єдиного алгоритму – отримується робочий мобільний Flutter-застосунок.

Flutter SDK - це основа для розробки крос-платформних застосунків, яка надає набір інструментів для створення UI-компонентів на iOS та Android з єдиної кодової бази. Фреймворк відкриває Доступ до великої кількості віджетів для побудови сучасних інтерфейсів та забезпечує високу продуктивність завдяки власному рушію. У даній програмній реалізації, SDK використовується для побудови всіх екранних компонентів, організації навігації між сторінками та інтеграції сторонніх пакетів.

Бібліотека flutter_bloc реалізує патерн BLoC (Business Logic Component), який дозволяє розділити бізнес-логіку функцій від реалізації графічного інтерфейсу. Даний набір правил забезпечує керування станом додатку через потоки подій та станів, а також дозволяє легко тестувати бізнес-логіку окремо від інтерфейсу, що має позитивний вплив на підтримку коду. У системі створено окремі BLoC-компоненти для завантаження категорій (BoardsBloc) та зображень (GalleryBloc).

Кожен BLoC отримує події та викликає відповідні методи API, обробляє результати та змінює стан. Це забезпечує реактивність UI - інтерфейс автоматично оновлюється при зміні стану, а бізнес-логіка залишається ізольованою від графічних віджетів.

Бібліотека dio – це потужний HTTP-клієнт для Dart/Flutter, який використовується для роботи з мережевими запитами. Бібліотека підтримує глобальні налаштування, інтерцептори, скасування запитів, завантаження/вивантаження файлів, обробку помилок та автоматичне повторення запитів. Вона забезпечує надсилання HTTP-запитів до сторонніх API, обробку

відповідей, повторення запитів у разі помилок та завантаження файлів.

У реалізованому мобільному додатку dio використовується як для отримання списку зображень з API, так і для безпосереднього завантаження файлів на пристрій. Це дозволяє ефективно працювати з великими обсягами даних, контролювати прогрес завантаження та обробляти мережеві помилки.

Для відображення зображень у вигляді сучасної, адаптивної сітки використовується flutter_staggered_grid_view. Ця бібліотека дозволяє створювати сітки з елементами різної висоти, що підвищує візуальну привабливість та зручність для галерей із фотографіями різних розмірів.

У системі вона використовується для побудови основної галереї, де кожне зображення займає оптимальне місце у сітці.

Бібліотека full_screen_image забезпечує можливість перегляду зображень у повноекранному режимі. Ці функцій покращують взаємодію з додатком, дозволяючи детально розглянути вибране фото, використовуючи жести масштабування та прокручування.

У системі інтеграція цієї бібліотеки дозволяє відкривати будь-яке зображення з галереї на весь екран одним натисканням.

Для роботи з файлами на пристрої потрібні відповідні дозволи операційної системи. Бібліотека permission_handler дозволяє перевіряти та запитувати у користувача права доступу до сховища. Підтримує перевірку статусу дозволу, відкриття налаштувань додатку, показ пояснень користувачу.

У системі використання permission_handler є обов’язковим етапом перед завантаженням зображень на пристрій. Перед початком завантаження ImageDownloader викликає _requestPermission, який перевіряє статус дозволу та, при необхідності, ініціює його запит.

Для збереження файлів на пристрої необхідно отримати коректний шлях до директорії. Бібліотека path_provider надає методи для отримання доступу до основних директорій файлової системи (наприклад, getExternalStorageDirectory для Android, getApplicationDocumentsDirectory для iOS).

У системі path_provider використовується для визначення директорії, у яку буде збережено завантажене зображення, що Забезпечує зручне та безпечне збереження файлів без хардкодингу шляхів, незалежно від операційної системи.

Взаємодія бібліотек у системі

Взаємодія зазначених бібліотек забезпечує повний цикл роботи з зображеннями:

– отримання списку категорій та зображень з API (dio, flutter_bloc),

– відображення їх у адаптивній сітці (flutter_staggered_grid_view),

– перегляд у повноекранному режимі (full_screen_image),

– завантаження файлів на пристрій із дотриманням прав доступу (permission_handler, path_provider, dio).

Завдяки такій архітектурі система є масштабованою, зручною для користувача та відповідає сучасним вимогам до мобільних застосунків.

Алгоритм завантаження категорій / зображень з API:

```
import 'dart:math';

import 'package:dio/dio.dart';

class PixabayApi {
  final Dio _dio = Dio();
  final String _baseURL = 'https://pixabay.com/api/';
  final String _apiKey = 'API_KEY';

  Future<List<dynamic>> fetchPhotos({
    required String imageSearch,
    required String imageType,
    required int amount
  }) async {

    try {
      final randomPage = Random().nextInt(100) + 1;
      Response response;
      do {
        response = await _dio.get(
          _baseURL,
          queryParameters: {
            'key': _apiKey,
            'q': imageSearch,
            'image_type': imageType,
            'per_page': amount,
          },
        );
        if (response.statusCode == 429 || response.statusCode == 400) {
          await Future.delayed(Duration(seconds: 5));
        }
      } while (response.statusCode == 429);

      if (response.statusCode == 200) {
        return response.data['hits'];
      }
      else {
        return Future.error('Error when loading photos!\n');
      }
    }
    catch (e) {
      return Future.error('Error when fetching photos!\n ${e.toString()}');
    }
  }
}
```

Алгоритм завантаження зображення на девайс:

```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:path_provider/path_provider.dart';
import 'package:dio/dio.dart';

class ImageDownloader {
  final Dio _dio = Dio();

  Future<bool> _requestPermission() async {
```

```
    var status = await Permission.storage.status;
    if (!status.isGranted) {
      status = await Permission.storage.request();
    }
    return status.isGranted;
  }

  Future<Directory?> _getDownloadDirectory() async {
    if (Platform.isAndroid) {
      return await getExternalStorageDirectory();
    }
    else if (Platform.isIOS) {
      return await getApplicationDocumentsDirectory();
    }
    return null;
  }

  Future<void> downloadFile(String url) async {
    String fileName = url.split('/').last;
      final Directory? directory = await _getDownloadDirectory();

      if (directory != null) {
        final String filePath = '/storage/emulated/0/Download/$fileName';
        final File file = File(filePath);

        try {
          Response response = await _dio.download(
            url,
            file.path,
            onReceiveProgress: (received, total) {
              if (total != -1) {
                print(
                  'Downloading: ${(received / total *
100).toStringAsFixed(0)}%',
                );
              }
            },
          );
          if (response.statusCode == 200) {
            print('File downloaded to: $filePath');
          } else {
            print('Failed to download file: ${response.statusCode}');
          }
        } catch (e) {
          print('Download failed: $e');
        }
      } else {
        print('Unable to get directory for file saving.');
      }
  }
}
```

Алгоритм збереження завантажених категорій для виведення на екран:

```
import 'package:creartify/network/pixabay_api.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_event.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_state.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class BoardsBloc extends Bloc<BoardsEvent, BoardsState> {
  final PixabayApi api;
```

```
  BoardsBloc(this.api) : super(BoardsInitialState()) {
    on<BoardsFetchMultipleCategoriesEvent>(_onBoardsFetchMultiplePhotos);
  }

  Future<void> _onBoardsFetchMultiplePhotos(
      BoardsFetchMultipleCategoriesEvent event,
      Emitter<BoardsState> emit) async {

    emit(BoardsLoadingState());
    try {
      final Map<String, List<dynamic>> results = {};
      for (String query in event.searchQueries) {
        final photos = await api.fetchPhotos(
            imageSearch: query,
            imageType: event.type,
            amount: event.amount
        );
        results[query] = photos;

        emit(BoardsPartialLoadedState(
          categorizedPhotos: Map.from(results),
          lastLoadedCategory: query,
        ));
      }

      emit(BoardsLoadedState(categorizedPhotos: results));
    }
    catch (e) {
      emit(BoardsErrorState(errorMessage: e.toString()));
    }
  }

}

abstract class BoardsEvent {}

class BoardsFetchMultipleCategoriesEvent extends BoardsEvent {
  BoardsFetchMultipleCategoriesEvent({
    required this.searchQueries,
    required this.type,
    required this.amount});

  final List<String> searchQueries;
  final String type;
  final int amount;
}

class NavigationState {
  NavigationState({required this.pageIndex});

  final int pageIndex;
}

abstract class BoardsState {}

class BoardsInitialState extends BoardsState {}
class BoardsLoadingState extends BoardsState {}

class BoardsPartialLoadedState extends BoardsState {
  BoardsPartialLoadedState({
    required this.categorizedPhotos,
    required this.lastLoadedCategory,
  });
```

```
  final Map<String, List<dynamic>> categorizedPhotos;
  final String lastLoadedCategory;
}

class BoardsLoadedState extends BoardsState {
  BoardsLoadedState({required this.categorizedPhotos});
  final Map<String, List<dynamic>> categorizedPhotos;
}

class BoardsErrorState extends BoardsState {
  BoardsErrorState({required this.errorMessage});
  final String errorMessage;
}
```

Алгоритм виведення отриманих зображень на екран:

```
import 'package:creartify/network/pixabay_api.dart';
import
'package:creartify/presentation/gallery_page/bloc/gallery_event.dart';
import
'package:creartify/presentation/gallery_page/bloc/gallery_state.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class GalleryBloc extends Bloc<GalleryEvent, GalleryState> {
  final PixabayApi api;
  final List<dynamic> _photos = [];
  final int _retryAttempts = 3;
  final int _retryDelay = 5;

  GalleryBloc(this.api) : super(GalleryInitialState()) {
    on<GalleryFetchPhotosEvent>(_onGalleryFetchPhotos);
    on<GalleryDisposeEvent>(_onGalleryDisposeEvent);
  }

  void _onGalleryDisposeEvent(GalleryDisposeEvent event,
Emitter<GalleryState> emit) {
    _photos.clear();
    emit(GalleryDisposeState());
  }

  Future<void> _onGalleryFetchPhotos(GalleryFetchPhotosEvent event,
Emitter<GalleryState> emit) async {
    int attempts = 0;
    while (attempts < _retryAttempts) {
      try {
        if (attempts > 0) {
          await Future.delayed(Duration(seconds: _retryDelay));
        }
        if (state is! GalleryLoadingState) {
          emit(GalleryLoadingState());
        }

        final loadedPhotos = await api.fetchPhotos(
            imageSearch: event.search,
            imageType: event.type,
            amount: event.amount);
        final uniquePhotos = loadedPhotos.where((photo) {
```

```
          return !_photos.any((existingPhoto) => existingPhoto['id'] ==
photo['id']);
        }).toList();

        _photos.addAll(uniquePhotos);
        emit(GalleryLoadedState(photos: List.from(_photos)));
        return;
      }
      catch (e) {
        attempts++;
        if (attempts >= _retryAttempts) {
          emit(GalleryErrorState(errorMessage: e.toString()));
          break;
        }
      }
    }
    if (_photos.isNotEmpty) {
      emit(GalleryLoadedState(photos: List.from(_photos)));
    }
  }
}
abstract class GalleryEvent {}

class GalleryFetchPhotosEvent extends GalleryEvent {
  GalleryFetchPhotosEvent({
    required this.search,
    required this.type,
    required this.amount});
  final String search;
  final String type;
  final int amount;
}

class GalleryDisposeEvent extends GalleryEvent {}


abstract class GalleryState {}

class GalleryInitialState extends GalleryState {}
class GalleryLoadingState extends GalleryState {}

class GalleryLoadedState extends GalleryState {
  GalleryLoadedState({required this.photos});
  final List<dynamic> photos;
}

class GalleryDisposeState extends GalleryState {}

class GalleryErrorState extends GalleryState {
  GalleryErrorState({required this.errorMessage});
  final String errorMessage;
}
```

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення є відкритим (open-source) і розповсюджується за ліцензією Creative Commons CC BY-SA 4.0. Це означає, що будь-який користувач може вільно використовувати, змінювати та поширювати його, за умови дотримання вимог ліцензії.

Захист від несанкціонованих змін. Для забезпечення стабільності та безпеки програмного забезпечення реалізовано такі механізми:

– Контроль офіційного репозиторію. Основний код зберігається в репозиторії (GitHub/GitLab), де внесення змін здійснюється через Pull Requests з обов'язковою перевіркою.

– Використання підписаних релізів. GPG (GNU Privacy Guard) — це система шифрування та цифрового підпису, яка дозволяє перевіряти справжність джерела коду. Використання GPG-ключів дає змогу підтвердити, що певний коміт або реліз походить від офіційного розробника.

– Захист від небажаних змін. Впроваджено механізми обмеження прав доступу, щоб лише перевірені розробники могли редагувати критичні компоненти коду.

Захист від зловживання. Оскільки програмне забезпечення є відкритим, важливо дотримуватися ліцензійних умов:

– Код не може бути використаний у комерційних цілях без відповідного дозволу (якщо використовується ліцензія CC BY-NC).

– При розповсюдженні зміненої версії необхідно вказувати авторів оригінального проєкту (за умовами CC BY-SA).

– Не допускається використання коду для створення шкідливих програм або маніпулятивних алгоритмів.

– Використання програмного забезпечення у дослідницьких цілях можливе лише при дотриманні умов ліцензії та етичних норм.

– Всі модифікації, зроблені користувачами, повинні мати відповідну

документацію та бути доступними для перевірки іншими учасниками спільноти.

Забезпечення доступності та резервне копіювання. Для запобігання втрати даних реалізовано такі заходи:

– Резервне копіювання репозиторію на кількох платформах для захисту від втрати основного сховища коду.

– Документація для користувачів і розробників, яка пояснює принципи роботи, правила внесення змін та особливості алгоритмів.

– Тестування на стабільність. Перед публікацією нових версій проходять тестування на сумісність із попередніми версіями для забезпечення безперебійної роботи системи.

Завдяки відкритому коду та прозорості розробки, система може бути покращена спільнотою, що сприяє її безпеці та ефективності. Це дозволяє швидко виявляти потенційні загрози, усувати їх і підтримувати високу якість програмного забезпечення.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 показано головний екран програми – картки категорій з зображеннями. З нього видно, що інтерфейс головного меню складається з таких блоків:

- Заголовок сторінки категорій з описом
- Список карток з категоріями зображень
- Навігаційне меню

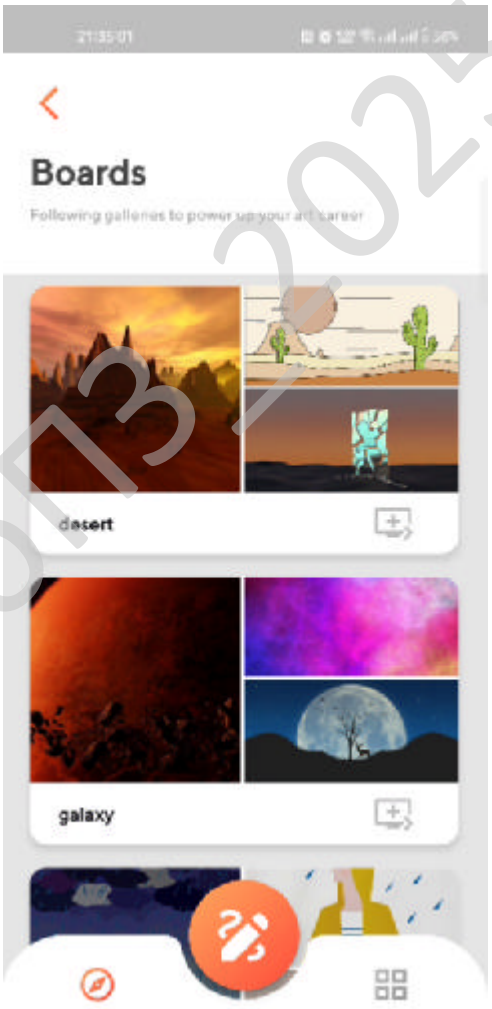

Рисунок 5.1 – Головний екран програми з картками категорій

Після натискання на кнопку переходу до категорії (або кнопку плитки в навігаційній панелі) відбудеться перехід до вікна перегляду зображеннь (рис. 5.2), що складається з наступних блоків:

- Заголовок сторінки зображень з описом
- Список зображень обраної категорії
- Навігаційне меню

Рисунок 5.2 – Екран з плитками зображень відповідної категорії

Після натискання на будь яке зображення відкриється вікно, що містить більше деталей щодо обраної ілюстрації (рис. 5.3), що складається з наступних блоків:

– Кнопки повернення назад та завантаження
– Повноекранне зображення
– Нижня панель з кнопкою огляду та назвою зображення

Рисунок 5.3 – Екран з повноекранним обраним зображенням

Після натискання на кнопку огляду «OVERVIEW» відкриється панель з детальним описом даних про зображення (рис. 5.4), що складається з наступних блоків:

– Кнопки «OVERVIEW»
– Назви зображення
– Списку з підзаголовків та деталей

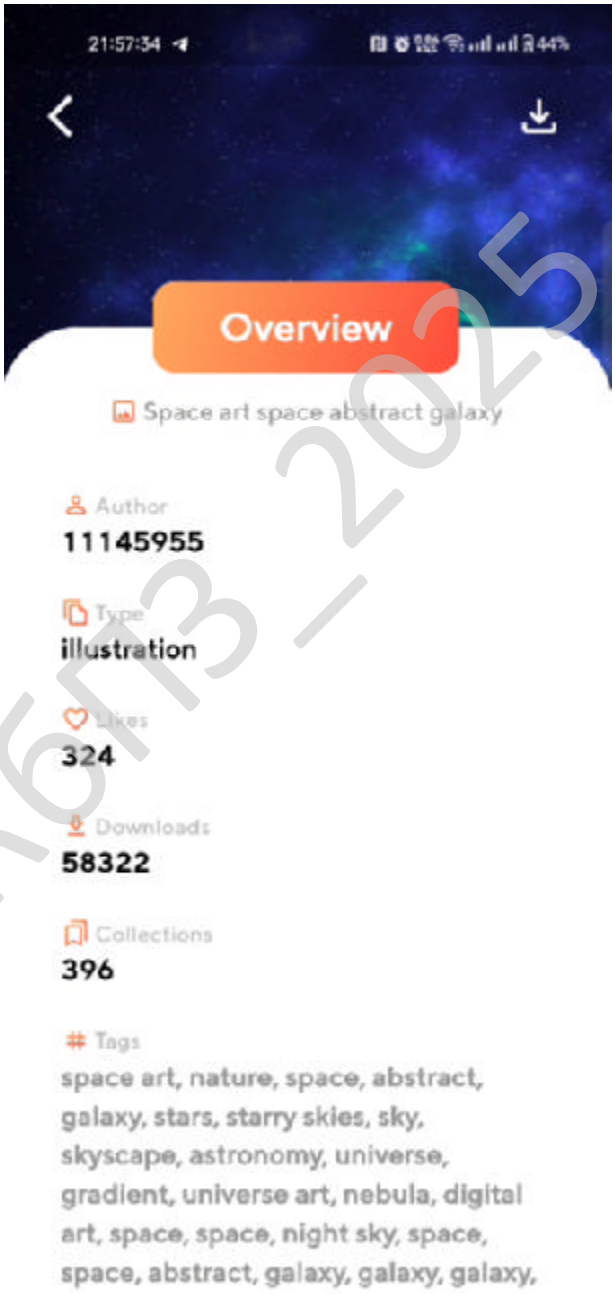

Рисунок 5.4 – Панель з детальним описом даних про зображення

Розроблений користувацький інтерфейс програмного забезпечення складається з декількох основних екранів, кожен з яких виконує окрему функціональну роль та побудований за принципами зручності та логічної структурованості.

Головний екран реалізує навігацію за категоріями зображень. Його компонентами є: заголовок з коротким описом, список карток категорій із зображеннями та нижня навігаційна панель. Забезпечується швидкий доступ до основного функціоналу програми.

При виборі категорії відбувається перехід до екрана перегляду зображень, який містить заголовок сторінки, список зображень, що належать до обраної категорії, а також навігаційне меню. Інтерфейс даного екрана адаптовано для зручного перегляду візуального контенту.

Подальша взаємодія з будь-яким із зображень ініціює відкриття повноекранного перегляду. У цьому режимі доступні елементи керування – кнопки повернення до попереднього екрана, завантаження зображення, а також нижня панель з назвою зображення та кнопкою виклику розширеного опису.

Натискання кнопки «OVERVIEW» відкриває панель із додатковими метаданими, де відображаються назва зображення та структурований список характеристик і описових елементів.

Загальна архітектура інтерфейсу розроблена з урахуванням принципів UX-дизайну, що забезпечує зручність користування без потреби в додатковому навчанні або спеціальних інструкціях. Інтерфейс є адаптивним та оптимізованим для використання на мобільних пристроях.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для мобільного додатку творчого арт-простору.

Для вирішення поставленої мети було проведено:

– дослідження існуючих методів реалізації мобільного додатку творчого арт-простору.

– розробку мобільного додатку творчого арт-простору.

– реалізацію програмного забезпечення мобільного додатку творчого арт-простору.

Реалізовані під час виконання кваліфікаційної роботи алгоритми дозволяють успішно вирішувати завдання мобільного додатку творчого арт-простору.

Розроблене програмне забезпечення представляє собою мобільний додаток, що можна використовувати практично на будь-якому сучасному мобільному девайсі, підключеному до інтернету.

Програмне забезпечення розроблялося у об’єктно-орієнтованій парадигмі, що робить його гнучким та зручним для підтримки та масштабування.

Для розробки програмного забезпечення системи мобільного додатку творчого арт-простору використовувалися мова програмування Dart з фреймворком Flutter та додатковими користувацькими бібліотеками. Дані мови програмування дозволили ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

У п’ятому розділі пояснювальної записки надаються усі необхідні рекомендації з встановлення серверного програмного забезпечення та інструкція для використання клієнтського програмного забезпечення.

Для захисту розробленого програмного забезпечення було обрано вільну ліцензію Creative Commons.

Програмне забезпечення мобільного додатку творчого арт-простору, є актуальним інструментом у сучасному інформаційному середовищі. Воно може застосовуватись у дизайні, освіті, творчості та контент-маркетингу, забезпечуючи швидкий і зручний спосіб знаходження та перегляду якісних зображень. З огляду на зростання попиту на мобільні рішення та кросплатформенні інтерфейси, розробка подібних застосунків є не лише доцільною, а й необхідною. Таким чином, розроблене у цій кваліфікаційній роботі програмне забезпечення має практичне значення для підтримки творчої діяльності, підвищення продуктивності та оптимізації роботи з візуальним контентом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pinterest [Електронний ресурс] // pinterest.com – Режим доступу: https://www.pinterest.com/

2. Pinterest [Електронний ресурс] // wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/Pinterest

3. Definition Pinterest [Електронний ресурс] // techtarget.com – Режим доступу: https://www.techtarget.com/whatis/definition/Pinterest

4. DeviantArt [Електронний ресурс] // deviantart.com – Режим доступу: https://www.deviantart.com

5. DeviantArt [Електронний ресурс] // wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/DeviantArt

6. How deviantART Works [Електронний ресурс] // howstuffworks.com – Режим доступу: https://computer.howstuffworks.com/internet/social-networking/networks/deviantart.htm

7. ArtStation [Електронний ресурс] // artstation.com – Режим доступу: https://www.artstation.com/

8. Що це за веб-сайт ArtStation? [Електронний ресурс] // dimet.in.ua – Режим доступу: https://dimet.in.ua/ukraincyam/shho-ce-za-veb-sayt-artstation.html

9. Introduction to ArtStation! [Електронний ресурс] // medium.com – Режим доступу: https://kennshinabery.medium.com/introduction-to-artstation-b3b3caa47a10

10. Dribbble [Електронний ресурс] // dribbble.com – Режим доступу: https://dribbble.com/

11. Dribble [Електронний ресурс] // wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/Dribbble

12. What is Dribbble? What is the Use of Dribbble? [Електронний ресурс] // ruul.io – Режим доступу: https://ruul.io/blog/what-is-dribbble-what-is-the-use-of-dribbble

13. Pixabay [Електронний ресурс] // pixabay.com – Режим доступу: https://pixabay.com/

14. Pixabay [Електронний ресурс] // wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/Pixabay

15. Pixabay API [Електронний ресурс] // pixabay.com – Режим доступу: https://pixabay.com/api/docs/

16. Прикладний програмний інтерфейс [Електронний ресурс] // wikipedia.org – Режим доступу: https://uk.wikipedia.org/wiki/Прикладний_програмний_інтерфейс

17. AndroidStudio [Електронний ресурс] // wikipedia.org – Режим доступу: https://uk.wikipedia.org/wiki/Android_Studio

18. Definition AndroidStudio [Електронний ресурс] // techtarget.com – Режим доступу: https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio

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

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

21. Dart [Електронний ресурс] // wikipedia.org – Режим доступу: https://uk.wikipedia.org/wiki/Dart

22. Створення мобільних додатків на Dart [Електронний ресурс] // brander.ua – Режим доступу: https://brander.ua/technologies/dart

23. Flutter Documentation [Електронний ресурс] // Flutter.dev. – Режим доступу: https://docs.flutter.dev/

24. Flutter (Software) [Електронний ресурс] // wikipedia.org – Режим доступу: https://en.wikipedia.org/wiki/Flutter_(software)

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

26. Course: The Complete Flutter Development Bootcamp with Dart [Електронний ресурс] // udemy.com – Режим доступу: https://www.udemy.com/course/flutter-bootcamp-with-dart/

27. Figma- Art gallery app UI [Електронний ресурс] // figma.com – Режим доступу: https://www.figma.com/design/UufT4BPRsMEt6Cbp9Buq2L/Art-gallery-app-UI--Community-?node-id=0-1

28. Bloc State Management Library [Електронний ресурс] // bloclibrary.dev – Режим доступу: https://bloclibrary.dev/

29. bloc – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/bloc

30. BLoC у Flutter без стресу: Як нарешті розібратися зі станами [Електронний ресурс] // robotdreams.cc – Режим доступу: https://robotdreams.cc/uk/blog/686-bloc-in-flutter-without-stress

31. dio – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/dio

32. Нетворкінг у Flutter додатках — про просте і складне на прикладі Tide. Частина 3: HTTP клієнт та перехоплювачі запитів з dio [Електронний ресурс] // dou.ua – Режим доступу: https://dou.ua/forums/topic/39242/

33. Deep Dive into Popular Flutter Packages: Dio [Електронний ресурс] // medium.com – Режим доступу: https://ms3byoussef.medium.com/deep-dive-into-popular-flutter-packages-dio-09e9aea86df8

34. path_provider – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/path_provider

35. Exploring the Various Functions and Features of the Flutter Path Provider [Електронний ресурс] // dhiwise.com – Режим доступу: https://www.dhiwise.com/post/functions-and-features-of-the-flutter-path-provider

36. Unlocking file system access with path_provider [Електронний ресурс] // medium.com – Режим доступу: https://medium.com/@flutternewshub/unlocking-file-

system-access-with-path-provider-a-comprehensive-guide-for-flutter-developers-356644908d5a

37. permission_handler – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/permission_handler

38. Permission handling in Flutter with permission_handler [Електронний ресурс] // medium.com – Режим доступу: https://medium.com/@flutternewshub/permission-handling-in-flutter-with-permission-handler-a501e970ad3c

39. flutter_staggered_grid_view – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/flutter_staggered_grid_view

40. zoom_tap_animation – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/zoom_tap_animation

41. loading_animation_widget – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/loading_animation_widget

42. full_screen_image – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/full_screen_image

43. gradient_icon – Dart package [Електронний ресурс] // pub.dev – Режим доступу: https://pub.dev/packages/gradient_icon

44. Payne R. Beginning App Development with Flutter [Електронний ресурс] / Rap Payne. – Apress, 2019. – 350 с. – Режим доступу: https://digilib.stekom.ac.id/assets/dokumen/ebook/feb_3872ce7467cbdc7beedfcdc12b2b607b0ba36429_1649057575.pdf

45. Kurniawan B. Flutter for Beginners [Електронний ресурс] / Alessandro Biessek. – Packt Publishing, 2019. – 350 с. – Режим доступу: http://livre21.com/LIVREF/F6/F00645.pdf

46. Flutter Bloc: Навчальний посібник [Електронний ресурс] // SpaceLab.ua. – Режим доступу: https://spacelab.ua/lessons/uchebnoe-posobie-po-biblioteke-flutter-bloc/

47. Buckett C. Dart in Action [Електронний ресурс] / Chris Buckett. – Manning

Publications, 2015. – 350 с. – Режим доступу: https://www.manning.com/books/dart-in-action

48. Відкрите програмне забезпечення – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Відкрите_програмне_забезпечення

49. Ліцензії Creative Commons – Вікіпедія [Електронний ресурс] // Вікіпедія. – Режим доступу: https://uk.wikipedia.org/wiki/Ліцензії_Creative_Commons

50. Що таке Creative Commons [Електронний ресурс] – Режим доступу до ресурсу: https://www.creativecommons.org.ua/about-creative-commons

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку мобільного застосунку для отримання доступу до арт-простору.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №48-02 від 17.01.2025 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення мобільного додатку творчого арт-простору за допомогою фреймворку Flutter.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– робочий мобільний застосунок творчого арт-простору;

– безперебійне завантаження ілюстрацій та їх коректне відображення;

– адаптивний та інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію та тип мобільної операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на Core OS / ART архітектурах та має працювати в ОС Android та iOS і з сумісними з цими платформами пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок компіляції встановлювального файлу відповідно до вимог операційних систем Android та iOS.

5.8.1 Обладнання

Девайс з операційною системою Android Lollipop 5.0 з процесором ARM A9 1 ГГц та оперативною пам’яттю RAM 512 МБ або девайси з операційною системою iOS 12.0 або більше (iPhone 5s, iPad Air 1-го покоління, iPad mini 2, iPod touch 6-го покоління).

5.8.2 Мова програмування

Програму розроблено на мові програмування Dart з використанням фреймворку Flutter.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Перелік документів, що розробляються

– Структурна схема системи.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Пояснювальна записка.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 24.05.2025 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист __.__.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Є.В. Мелешко

Програмне забезпечення мобільного додатку творчого арт-простору за допомогою фреймворку Flutter

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 33

Літера: РП

Кропивницький – 2025 року

// pubspec.yaml – Файл регулює підключені бібліотеки та ресурси

```
name: creartify
description: "A new Flutter project."
publish_to: 'none'
version: 1.0.0+1

environment:
  sdk: ^3.5.0

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.8
  flutter_bloc: ^9.0.0
  dio: ^5.7.0
  # UI
  flutter_staggered_grid_view: ^0.7.0
  zoom_tap_animation: ^1.1.0
  loading_animation_widget: ^1.3.0
  full_screen_image: ^2.0.0
  gradient_icon: ^2.0.9
  # Downloader
  permission_handler: ^11.3.1
  path_provider: ^2.1.5

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^5.0.0

flutter:
  uses-material-design: true
  assets:
    - assets/
  fonts:
    - family: TTNorms
      fonts:
        - asset: assets/fonts/TTNorms-Medium.otf
```

// lib/main.dart – Файл з головним скриптом запуску програми та підключенням функціональних блоків BLoC.

```
import 'package:creartify/network/pixabay_api.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_bloc.dart';
import
'package:creartify/presentation/navigation_page/bloc/navigation_bloc.dart';
import
'package:creartify/presentation/navigation_page/navigation_view.dart';
import 'package:creartify/presentation/gallery_page/bloc/gallery_bloc.dart';
import 'package:creartify/theme/theme_app.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter/services.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const CreArtifyApp());
}


class CreArtifyApp extends StatelessWidget {
  const CreArtifyApp ({super.key});

  @override
  Widget build(BuildContext context) {
    SystemChrome.setPreferredOrientations([
      DeviceOrientation.portraitUp,
      DeviceOrientation.portraitDown,
    ]);
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeApp.light,
      home: MultiBlocProvider(
        providers: [
          BlocProvider(create: (context) => NavigationBloc(),),
          BlocProvider(create: (context) => GalleryBloc(PixabayApi()),),
          BlocProvider(create: (context) => BoardsBloc(PixabayApi()),),
        ],
        child: NavigationView(),
      ),
    );
  }
}
```

// lib/theme/units.dart – Файл з константами

```
import 'package:creartify/theme/colors.dart';
import 'package:flutter/material.dart';

double getScreenWidth(BuildContext context) =>
MediaQuery.of(context).size.width;
double getScreenHeight(BuildContext context) =>
MediaQuery.of(context).size.height;

final AppColors globalColors = AppColors();

// RADIUS
const double kRadiusDefault = 15.0;
const double kRadiusLarge = 35.0;
const double kRadiusXLarge = 55.0;

// MARGINS & PADDINGS
const double kIndentLittle = 3.0;
const double kIndentSmall = 5.0;
const double kIndentSmallDouble = 10.0;
const double kIndentDefault = 25.0;
const double kNotchMargin = 8.0;

// ILLUSTRATION VIEW
const double kInitialDraggableSheetSize = 0.09;
const double kMaximumDraggableSheetSize = 0.75;
const double kFabPositionPadding = 35;

// SIZES
const double kBottomAppBarHeight = 85;
const double kLoaderSmallSize = 0.07;
const double kLoaderLargeSize = 0.20;
const double kIconSmall = 35;
const double kIconLarge = 60;
```

// lib/theme/colors.dart – файл з моделлю класу кольорів програми

```
import 'package:flutter/material.dart';

abstract class AppColor {
  const AppColor({
    required this.primaryForeground,
    required this.primaryBackground,
    required this.inactiveLight,
    required this.inactiveDark,
    required this.textBlack,
    required this.activeLight,
    required this.activeDark,
  });

  final Color primaryForeground;
  final Color primaryBackground;
  final Color inactiveLight;
  final Color inactiveDark;
  final Color textBlack;
  final Color activeLight;
  final Color activeDark;
}

class AppColors implements AppColor {
  @override
  Color get primaryForeground => const Color(0xffffffff);

  @override
  Color get primaryBackground => const Color(0xffe5e5e5);

  @override
  Color get inactiveLight => const Color(0xffbebebe);

  @override
  Color get inactiveDark => const Color(0xff989898);

  @override
  Color get textBlack => const Color(0xff000000);

  @override
  Color get activeLight => const Color(0xffFFAC5F);

  @override
  Color get activeDark => const Color(0xffFF4D3C);
}
```

// lib/theme/theme_app.dart – файл конфігурацій дизайну певних віджетів

```
import 'package:creartify/theme/colors.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';

class ThemeApp {
  static final AppColor _colors = AppColors();

  static ThemeData light = ThemeData(
    textTheme: textTheme,
    scaffoldBackgroundColor: _colors.primaryBackground,
    floatingActionButtonTheme: _floatingActionButtonTheme(),
    bottomAppBarTheme: _bottomAppBarTheme(),
  );

  static TextTheme textTheme = TextTheme(
    titleSmall: TextStyle(fontFamily: 'TTNorms', fontSize: 19, color:
globalColors.inactiveDark),
    labelSmall: TextStyle(fontFamily: 'TTNorms', fontSize: 17, color:
globalColors.inactiveLight),
    labelMedium: TextStyle(fontFamily: 'TTNorms', fontSize: 21, color:
globalColors.textBlack, fontWeight: FontWeight.bold),
  );

  static FloatingActionButtonThemeData _floatingActionButtonTheme() {
    return  FloatingActionButtonThemeData(
      shape: CircleBorder(),
      focusElevation: 500,
      backgroundColor: _colors.activeDark,
      foregroundColor: _colors.primaryForeground,
    );
  }

  static BottomAppBarTheme _bottomAppBarTheme() {
    return BottomAppBarTheme(
      color: _colors.primaryForeground,
      elevation: 12.0,
      shape: CircularNotchedRectangle(),
      height: kBottomAppBarHeight,
    );
  }

  static LinearGradient gradientColors(Color firstColor, Color secondColor)
{
    return LinearGradient(
      begin: Alignment.topLeft,
      end: Alignment.bottomRight,
      colors: <Color>[firstColor, secondColor],
    );
  }
}
```

// lib/network/download_script.dart – Скрипт для завантаження зображення локально на мобільний девайс

```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:path_provider/path_provider.dart';
import 'package:dio/dio.dart';

class ImageDownloader {
  final Dio _dio = Dio();

  Future<bool> _requestPermission() async {
    var status = await Permission.storage.status;
    if (!status.isGranted) {
      status = await Permission.storage.request();
    }
    return status.isGranted;
  }

  Future<Directory?> _getDownloadDirectory() async {
    if (Platform.isAndroid) {
      return await getExternalStorageDirectory();
    }
    else if (Platform.isIOS) {
      return await getApplicationDocumentsDirectory();
    }
    return null;
  }

  Future<void> downloadFile(String url) async {
    String fileName = url.split('/').last;
      final Directory? directory = await _getDownloadDirectory();

      if (directory != null) {
        final String filePath = '/storage/emulated/0/Download/$fileName';
        final File file = File(filePath);

        try {
          Response response = await _dio.download(
            url,
            file.path,
            onReceiveProgress: (received, total) {
              if (total != -1) {
                print(
                  'Downloading: ${(received / total *
100).toStringAsFixed(0)}%',
                );
              }
            },
          );
          if (response.statusCode == 200) {
            print('File downloaded to: $filePath');
          } else {
            print('Failed to download file: ${response.statusCode}');
          }
        } catch (e) {
          print('Download failed: $e');
        }
      } else {
        print('Unable to get directory for file saving.');
      }
  }
}
```

// lib/network/pixabay_api.dart – Скрипт для отримання категоризованого списку зображень з API-сервісу

```
import 'dart:math';

import 'package:dio/dio.dart';

class PixabayApi {
  final Dio _dio = Dio();
  final String _baseURL = 'https://pixabay.com/api/';
  final String _apiKey = 'API_KEY';

  Future<List<dynamic>> fetchPhotos({
    required String imageSearch,
    required String imageType,
    required int amount
  }) async {

    try {
      final randomPage = Random().nextInt(100) + 1;
      Response response;
      do {
        response = await _dio.get(
          _baseURL,
          queryParameters: {
            'key': _apiKey,
            'q': imageSearch,
            'image_type': imageType,
            'per_page': amount,
          },
        );
        if (response.statusCode == 429 || response.statusCode == 400) {
          await Future.delayed(Duration(seconds: 5));
        }
      } while (response.statusCode == 429);

      if (response.statusCode == 200) {
        return response.data['hits'];
      }
      else {
        return Future.error('Error when loading photos!\n');
      }
    }
    catch (e) {
      return Future.error('Error when fetching photos!\n ${e.toString()}');
    }
  }
}
```

// lib/presentation/widgets/ – Тека з файлами-заготовками віджетів

```
import 'package:creartify/theme/theme_app.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';
import 'package:gradient_icon/gradient_icon.dart';
import 'package:creartify/presentation/widgets/app_loader_widget.dart';
import 'package:full_screen_image/full_screen_image.dart';
import 'package:creartify/theme/colors.dart';
import 'package:loading_animation_widget/loading_animation_widget.dart';
import 'package:zoom_tap_animation/zoom_tap_animation.dart';

class ApplicationHeaderWidget extends StatelessWidget {
  const ApplicationHeaderWidget({
    super.key,
    required this.title,
    required this.description
  });

  final String title;
  final String description;

  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.white,
      padding: const EdgeInsets.symmetric(horizontal: kIndentDefault,
vertical: kIndentSmallDouble),
      width: double.infinity,
      child: SafeArea(
        bottom: false,
        child: Column(
          mainAxisAlignment: MainAxisAlignment.start,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            IconButton(
              onPressed: () => {},
              highlightColor: globalColors.activeLight.withOpacity(0.07),
              icon: Container(
                padding: EdgeInsets.only(bottom: kIndentSmall * 2.9),
                child: GradientIcon(
                  icon: Icons.arrow_back_ios,
                  gradient:
ThemeApp.gradientColors(globalColors.activeLight, globalColors.activeDark),
                  size: kIconSmall,
                ),
              ),
            ),
            Padding(
              padding: const EdgeInsets.symmetric(vertical: kIndentSmall),
              child: Text(
                title,
                style: TextStyle(
                  fontFamily: 'TTNorms', fontWeight: FontWeight.bold,
                  fontSize: 35, color: Color(0xff464646),
                ),
              ),
            ),
            Text(
              description,
              style: TextStyle(
                fontFamily: 'TTNorms', fontSize: 15,
                color: Color(0xffa9a9a9),
              ),
            ),
          ],
```

```
        ),
      ),
    );
  }
}

class ApplicationImageLoaderWidget extends StatelessWidget {
  const ApplicationImageLoaderWidget({super.key, required this.imageURL});
  final String imageURL;

  @override
  Widget build(BuildContext context) {
    return Image.network(
      imageURL,
      fit: BoxFit.cover,

      loadingBuilder: (context, child, loadingProgress) {
        if (loadingProgress == null) { return child; }
        return Container(
          color: Colors.white,
          child: Center(
            child: AppLoaderWidget(size: kLoaderSmallSize),
          ),
        );
      },
      errorBuilder: (context, error, stackTrace) {
        return Container(
          color: Colors.grey[300],
          child: const Icon(Icons.broken_image, color: Colors.grey),
        );
      },

    );
  }
}

class AppLoaderWidget extends StatelessWidget {
  const AppLoaderWidget({super.key,required this.size});
  final double size;

  @override
  Widget build(BuildContext context) {
    return LoadingAnimationWidget.discreteCircle(
      color: globalColors.primaryBackground,
      secondRingColor: globalColors.activeLight,
      thirdRingColor: globalColors.activeDark,
      size: getScreenWidth(context) * size,
    );
  }
}

class AppZoomTapAnimation extends StatelessWidget {
  const AppZoomTapAnimation({super.key, required this.child});
  final Widget child;

  @override
  Widget build(BuildContext context) {
    return ZoomTapAnimation(
      begin: 1.0, end: 1.4,
      beginDuration: const Duration(milliseconds: 40),
      endDuration: const Duration(milliseconds: 290),
      child: child
    );
  }
}
```

// lib/presentation/navigation_page/bloc/ – Тека з BLoC-архітектурою навігаційної панелі

```
import
'package:creartify/presentation/navigation_page/bloc/navigation_event.dart';
import
'package:creartify/presentation/navigation_page/bloc/navigation_state.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class NavigationBloc extends Bloc<NavigationEvent, NavigationState> {
  NavigationBloc() : super(NavigationState(pageIndex: 0)) {
    on<NavigationChangedPageEvent>(
      (event, emit) => emit(NavigationState(pageIndex: event.pageIndex))
    );
  }
}


abstract class NavigationEvent {}

class NavigationChangedPageEvent extends NavigationEvent {
  NavigationChangedPageEvent({required this.pageIndex});

  final int pageIndex;
}


class NavigationState {
  NavigationState({required this.pageIndex});

  final int pageIndex;
}
```

// lib/presentation/navigation_page/navigation_view.dart – Файл з дизайном та функціоналом навігаційної панелі

```
// FLUTTER LIBRARIES
import 'package:creartify/presentation/widgets/app_zoom_tap_animation.dart';
import 'package:creartify/theme/theme_app.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
// BOTTOM NAVIGATION PAGE
import
'package:creartify/presentation/navigation_page/bloc/navigation_bloc.dart';
import
'package:creartify/presentation/navigation_page/bloc/navigation_event.dart';
import
'package:creartify/presentation/navigation_page/bloc/navigation_state.dart';
// NAVIGATION PAGES
import 'package:creartify/presentation/boards_page/boards_view.dart';
import 'package:creartify/presentation/gallery_page/gallery_view.dart';
import 'package:gradient_icon/gradient_icon.dart';

class NavigationView extends StatefulWidget {
  const NavigationView({super.key});
  static final ValueNotifier<int> navClickedIndexNotifier =
ValueNotifier<int>(0);
  @override
  State<NavigationView> createState() => _NavigationViewState();
}
class _NavigationViewState extends State<NavigationView> {
  final List<Map<String, dynamic>> navButtonData = [
    { 'page': BoardsView(), 'icon': Icons.explore_outlined, 'index': 0 },
    { 'page': GalleryView(), 'icon': Icons.grid_view, 'index': 1 },
  ];

  @override
  Widget build(BuildContext context) {
    NavigationBloc navBloc = context.read<NavigationBloc>();
    BottomAppBarTheme bottomAppBarTheme =
Theme.of(context).bottomAppBarTheme;
    return Scaffold(

      extendBody: true,
      body: BlocBuilder<NavigationBloc, NavigationState>(
        builder: (context, state) {
          return navButtonData[state.pageIndex]['page'];
        },
      ),

      floatingActionButtonLocation:
FloatingActionButtonLocation.centerDocked,
      floatingActionButton: AppZoomTapAnimation(
        child: FloatingActionButton.large(
          onPressed: () {},
          child: Container(
            width: 100,
            height: 100,
            decoration: BoxDecoration(
              shape: BoxShape.circle,
              gradient: ThemeApp.gradientColors(globalColors.activeLight,
globalColors.activeDark),
            ),
            child: Icon(Icons.draw_rounded, size: kIconLarge,),
          ),
        ),
      ),
```

```
      bottomNavigationBar: ClipRRect(
        borderRadius: BorderRadius.only(
          topLeft: Radius.circular(kRadiusXLarge),
          topRight: Radius.circular(kRadiusXLarge),
        ),
        child: BottomAppBar(
          color: bottomAppBarTheme.color,
          elevation: bottomAppBarTheme.elevation,
          shape: bottomAppBarTheme.shape,
          height: bottomAppBarTheme.height,
          notchMargin: kNotchMargin,
          clipBehavior: Clip.antiAlias,

          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: [
              for (var item in navButtonData) ...[
                if (item['index'] == 1) SizedBox(width: 120.0),
                Expanded(
                  child: ValueListenableBuilder(
                    valueListenable: NavigationView.navClickedIndexNotifier,
                    builder: (context, currentIndex, child) {
                      return IconButton(
                        onPressed: () {
                          navBloc.add(NavigationChangedPageEvent(pageIndex:
item['index']));
                          setState(() =>
NavigationView.navClickedIndexNotifier.value = item['index']);
                        },
                        icon: Container(
                          padding: EdgeInsets.only(bottom: kIndentSmall *
2.9),
                          child: GradientIcon(
                            icon: item['icon'],
                            gradient:
NavigationView.navClickedIndexNotifier.value == item['index']
                                ?
ThemeApp.gradientColors(globalColors.activeLight, globalColors.activeDark)
                                :
ThemeApp.gradientColors(globalColors.inactiveLight,
globalColors.inactiveDark),
                            size: kIconSmall,
                          ),
                        ),
                      );
                    },
                  ),
                ),
              ],
            ),
          ),
        ),
      );
    }
}
```

// lib/presentation/boards_page/bloc/ – Тека з BLoC-архітектурою екрану вибору категорій іллюстрацій

```
import 'package:creartify/network/pixabay_api.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_event.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_state.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class BoardsBloc extends Bloc<BoardsEvent, BoardsState> {
  final PixabayApi api;
  BoardsBloc(this.api) : super(BoardsInitialState()) {
    on<BoardsFetchMultipleCategoriesEvent>(_onBoardsFetchMultiplePhotos);
  }

  Future<void> _onBoardsFetchMultiplePhotos(
      BoardsFetchMultipleCategoriesEvent event,
      Emitter<BoardsState> emit) async {

    emit(BoardsLoadingState());
    try {
      final Map<String, List<dynamic>> results = {};
      for (String query in event.searchQueries) {
        final photos = await api.fetchPhotos(
            imageSearch: query,
            imageType: event.type,
            amount: event.amount
        );
        results[query] = photos;

        emit(BoardsPartialLoadedState(
          categorizedPhotos: Map.from(results),
          lastLoadedCategory: query,
        ));
      }

      emit(BoardsLoadedState(categorizedPhotos: results));
    }
    catch (e) {
      emit(BoardsErrorState(errorMessage: e.toString()));
    }
  }

}


abstract class BoardsEvent {}

class BoardsFetchMultipleCategoriesEvent extends BoardsEvent {
  BoardsFetchMultipleCategoriesEvent({
    required this.searchQueries,
    required this.type,
    required this.amount});

  final List<String> searchQueries;
  final String type;
  final int amount;
}

class NavigationState {
  NavigationState({required this.pageIndex});

  final int pageIndex;
}
```

```
abstract class BoardsState {}

class BoardsInitialState extends BoardsState {}
class BoardsLoadingState extends BoardsState {}

class BoardsPartialLoadedState extends BoardsState {
  BoardsPartialLoadedState({
    required this.categorizedPhotos,
    required this.lastLoadedCategory,
  });
  final Map<String, List<dynamic>> categorizedPhotos;
  final String lastLoadedCategory;
}

class BoardsLoadedState extends BoardsState {
  BoardsLoadedState({required this.categorizedPhotos});
  final Map<String, List<dynamic>> categorizedPhotos;
}

class BoardsErrorState extends BoardsState {
  BoardsErrorState({required this.errorMessage});
  final String errorMessage;
}
```

// lib/presentation/boards_page/widgets/ – Тека з файлами-заготовками віджетів для екрану вибору категорій іллюстрацій

```
import 'package:creartify/theme/colors.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import
'package:creartify/presentation/navigation_page/navigation_view.dart';
import
'package:creartify/presentation/navigation_page/bloc/navigation_bloc.dart';
import
'package:creartify/presentation/navigation_page/bloc/navigation_event.dart';
import 'package:zoom_tap_animation/zoom_tap_animation.dart';
import
'package:creartify/presentation/widgets/app_image_loader_widget.dart';

class BoardCardWidget extends StatelessWidget {
  const BoardCardWidget({
    super.key,
    required this.previewImages,
    required this.categoryTitle,
  });

  final List<dynamic>? previewImages;
  final String categoryTitle;

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(kRadiusDefault),
        color: Colors.white,
        boxShadow: [
          BoxShadow(
            color: Colors.grey.withOpacity(0.3),
            spreadRadius: 1,
            blurRadius: 3,
            offset: Offset(0, 5),
          ),
        ],
      ),
      margin: EdgeInsets.symmetric(horizontal: kIndentDefault, vertical:
kIndentSmall*2),
      width: double.maxFinite,
      height: getScreenHeight(context)*0.26,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Expanded(
                child: SizedBox(
                  width: getScreenWidth(context)*0.5,
                  height: getScreenHeight(context)*0.20,
                  child: ClipRRect(
                    borderRadius: BorderRadius.only(topLeft:
Radius.circular(kRadiusDefault)),
                    child: Padding(
                      padding: const EdgeInsets.only(right: kIndentLittle),
                      child: BoardImageWidget(url:
previewImages![0]['webformatURL']),
                    ),
                  ),
                ),
```

```
              ),
              Expanded(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    // top image
                    SizedBox(
                      width: getScreenWidth(context)*0.5,
                      height: getScreenHeight(context)*0.10,
                      child: ClipRRect(
                        borderRadius: BorderRadius.only(topRight:
Radius.circular(kRadiusDefault)),
                        child: Padding(
                          padding: const EdgeInsets.only(bottom:
kIndentLittle),
                          child: BoardImageWidget(url:
previewImages![1]['webformatURL']),
                        ),
                      ),
                    ),
                    // bottom image
                    SizedBox(
                      width: getScreenWidth(context)*0.5,
                      height: getScreenHeight(context)*0.10,
                      child: BoardImageWidget(url:
previewImages![2]['webformatURL']),
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
      ),

      Expanded(
        flex: 1,
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal:
kIndentDefault),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              BoardTittleWidget(inputText: categoryTitle),
              BoardIconWidget(queryToSend: categoryTitle,)
            ],
          ),
        ),
      ),
    ],
  ),
);
  }
}

class BoardEmptyCardWidget extends StatelessWidget {
  const BoardEmptyCardWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(kRadiusDefault),
        color: globalColors.primaryForeground,
        boxShadow: [
          BoxShadow(
            color: Colors.grey.withOpacity(0.3),
            spreadRadius: 1,
```

```
            blurRadius: 3,
            offset: Offset(0, 5),
          ),
        ],
      ),
      margin: EdgeInsets.symmetric(
          horizontal: kIndentDefault, vertical: kIndentSmall),
      width: double.maxFinite,
      height: getScreenHeight(context) * kLoaderLargeSize * 2,
      child: Center(
        child: Icon(
          Icons.cancel_presentation_rounded,
          color: globalColors.inactiveDark,
          size: 100,
        ),
      ),
    );
  }
}

class _BoardIconWidgetState extends State<BoardIconWidget> with
SingleTickerProviderStateMixin {
  bool isButtonPressed = false;

  @override
  Widget build(BuildContext context) {
    return ZoomTapAnimation(
      begin: 1.0,
      end: 0.7,
      beginDuration: const Duration(milliseconds: 50),
      endDuration: const Duration(milliseconds: 250),
      child: TextButton(
        onPressed: () {
          GalleryView.categorizedQuery = widget.queryToSend;
          NavigationView.navClickedIndexNotifier.value = 1;

context.read<NavigationBloc>().add(NavigationChangedPageEvent(pageIndex:
1));
        },
        child: Icon(
          Icons.queue_play_next,
          color: isButtonPressed ? Colors.orangeAccent : Colors.grey[400],
          size: 35,
        ),
      ),
    );
  }
}
```

```
class BoardIconWidget extends StatefulWidget {
  const BoardIconWidget({
    super.key,
    required this.queryToSend,
  });

  final String queryToSend;

  @override
  State<BoardIconWidget> createState() => _BoardIconWidgetState();
}

class BoardImageWidget extends StatelessWidget {
  const BoardImageWidget({super.key, required this.url});
  final String url;

  @override
  Widget build(BuildContext context) {
    return ApplicationImageLoaderWidget(imageURL: url);
  }
}

class BoardTittleWidget extends StatelessWidget {
  const BoardTittleWidget({super.key, required this.inputText});
  final String inputText;

  @override
  Widget build(BuildContext context) {
    return Text(
      inputText,
      style: TextStyle(
        fontFamily: 'TTNorms',
        fontWeight: FontWeight.bold,
        fontSize: 18,
      ),
    );
  }
}
```

// lib/presentation/boards_page/boards_view.dart – Файл з дизайном та функціоналом екрану вибору категорій іллюстрацій

```
import
'package:creartify/presentation/boards_page/widgets/board_empty_card_widget.
dart';
import 'package:creartify/presentation/widgets/app_loader_widget.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_bloc.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_event.dart';
import 'package:creartify/presentation/boards_page/bloc/boards_state.dart';
import 'package:creartify/presentation/widgets/app_header_widget.dart';
import
'package:creartify/presentation/boards_page/widgets/board_card_widget.dart';

class BoardsView extends StatefulWidget {
  const BoardsView({super.key});
  @override
  State<BoardsView> createState() => _BoardsViewState();
}

class _BoardsViewState extends State<BoardsView> {

  @override
  void initState() {
    super.initState();
    final List<String> searchQueries = [
      'anime', 'sunset', 'mountain', 'ocean', 'forest',
      'cityscape', 'abstract', 'wildlife', 'desert', 'galaxy',
      'rain', 'snow', 'spring', 'autumn', 'summer',
      'winter', 'night', 'day', 'urban', 'rural'
    ];
    context.read<BoardsBloc>().add(
      BoardsFetchMultipleCategoriesEvent(
        searchQueries: searchQueries,
        type: 'illustration',
        amount: 3,
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: [

          Expanded(
            flex: 5,
            child: ApplicationHeaderWidget(
              title: 'Boards',
              description: 'Following galleries to power up your art
career',
            ),
          ),

          Expanded(
            flex: 14,
            child: BlocBuilder<BoardsBloc, BoardsState>(
              builder: (context, state) {

                if (state is BoardsInitialState || state is
BoardsLoadingState) {
```

```
                  return Center(child: AppLoaderWidget(size:
kLoaderLargeSize));
                }

                if (state is BoardsErrorState) {
                  return Center(child: AppLoaderWidget(size:
kLoaderLargeSize));
                }

                if (state is BoardsPartialLoadedState || state is
BoardsLoadedState) {
                  final illustrations = state is BoardsPartialLoadedState
                      ? state.categorizedPhotos
                      : (state as BoardsLoadedState).categorizedPhotos;

                  return ListView.builder(
                    padding: EdgeInsets.only(top: kIndentSmall, bottom:
kBottomAppBarHeight + kIndentSmallDouble),
                    itemCount: illustrations.keys.length,
                    itemBuilder: (context, index) {
                      final category = illustrations.keys.elementAt(index);
                      final previewImages = illustrations[category];
                      return BoardCardWidget(
                        previewImages: previewImages,
                        categoryTitle: category,
                      );
                    },
                  );
                }

                return const SizedBox.shrink();
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

// lib/presentation/gallery_page/bloc/ – Тека з BLoC-архітектурою екрану з категоризованим списком іллюстрацій

```
import 'package:creartify/network/pixabay_api.dart';
import
'package:creartify/presentation/gallery_page/bloc/gallery_event.dart';
import
'package:creartify/presentation/gallery_page/bloc/gallery_state.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

class GalleryBloc extends Bloc<GalleryEvent, GalleryState> {
  final PixabayApi api;
  final List<dynamic> _photos = [];
  final int _retryAttempts = 3;
  final int _retryDelay = 5;

  GalleryBloc(this.api) : super(GalleryInitialState()) {
    on<GalleryFetchPhotosEvent>(_onGalleryFetchPhotos);
    on<GalleryDisposeEvent>(_onGalleryDisposeEvent);
  }

  void _onGalleryDisposeEvent(GalleryDisposeEvent event,
Emitter<GalleryState> emit) {
    _photos.clear();
    emit(GalleryDisposeState());
  }

  Future<void> _onGalleryFetchPhotos(GalleryFetchPhotosEvent event,
Emitter<GalleryState> emit) async {
    int attempts = 0;
    while (attempts < _retryAttempts) {
      try {
        if (attempts > 0) {
          await Future.delayed(Duration(seconds: _retryDelay));
        }
        if (state is! GalleryLoadingState) {
          emit(GalleryLoadingState());
        }

        final loadedPhotos = await api.fetchPhotos(
            imageSearch: event.search,
            imageType: event.type,
            amount: event.amount);
        final uniquePhotos = loadedPhotos.where((photo) {
          return !_photos.any((existingPhoto) => existingPhoto['id'] ==
photo['id']);
        }).toList();

        _photos.addAll(uniquePhotos);
        emit(GalleryLoadedState(photos: List.from(_photos)));
        return;
      }
      catch (e) {
        attempts++;
        if (attempts >= _retryAttempts) {
          emit(GalleryErrorState(errorMessage: e.toString()));
          break;
        }
      }
    }
    if (_photos.isNotEmpty) {
      emit(GalleryLoadedState(photos: List.from(_photos)));
    }
  }
}
```

```
abstract class GalleryEvent {}

class GalleryFetchPhotosEvent extends GalleryEvent {
  GalleryFetchPhotosEvent({
    required this.search,
    required this.type,
    required this.amount});
  final String search;
  final String type;
  final int amount;
}

class GalleryDisposeEvent extends GalleryEvent {}


abstract class GalleryState {}

class GalleryInitialState extends GalleryState {}
class GalleryLoadingState extends GalleryState {}

class GalleryLoadedState extends GalleryState {
  GalleryLoadedState({required this.photos});
  final List<dynamic> photos;
}

class GalleryDisposeState extends GalleryState {}

class GalleryErrorState extends GalleryState {
  GalleryErrorState({required this.errorMessage});
  final String errorMessage;
}
```

// lib/presentation/gallery_page/widgets/ – Тека з файлами-заготовками віджетів для екрану з категоризованим списком іллюстрацій

```
import
'package:creartify/presentation/widgets/app_image_loader_widget.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';
import 'package:full_screen_image/full_screen_image.dart';

class GalleryImageWidget extends StatelessWidget {
  const GalleryImageWidget({
    super.key,
    required this.width,
    required this.height,
    required this.url
  });

  final double width;
  final double height;
  final String url;

  @override
  Widget build(BuildContext context) {
    return UnconstrainedBox(
      child: Container(
        width: width,
        height: height,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(kRadiusDefault),
        ),
        child: ClipRRect(
          borderRadius: BorderRadius.circular(kRadiusDefault),
          child: ApplicationImageLoaderWidget(imageURL: url),
        ),
      ),
    );
  }
}
```

// lib/presentation/gallery_page/gallery_view.dart – Файл з дизайном та функціоналом екрану з категоризованим списком іллюстрацій

```
import 'dart:math';
import
'package:creartify/presentation/illustration_page/illustration_view.dart';
import
'package:creartify/presentation/gallery_page/widgets/gallery_image_widget.da
rt';
import 'package:creartify/presentation/widgets/app_loader_widget.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:creartify/presentation/gallery_page/bloc/gallery_bloc.dart';
import
'package:creartify/presentation/gallery_page/bloc/gallery_event.dart';
import
'package:creartify/presentation/gallery_page/bloc/gallery_state.dart';
import 'package:creartify/presentation/widgets/app_header_widget.dart';
import 'package:creartify/theme/units.dart';
import
'package:flutter_staggered_grid_view/flutter_staggered_grid_view.dart';

class GalleryView extends StatefulWidget {
  const GalleryView({ super.key });
  static String categorizedQuery = '';
  @override
  State<GalleryView> createState() => _GalleryViewState();
}

class _GalleryViewState extends State<GalleryView> {
  int _page = 1;
  final int _pageSize = 10;
  final ScrollController _scrollController = ScrollController();
  final Map<int, double> _imageHeights = {};
  bool isImageClicked = false;

  void _loadPhotos() {
    context.read<GalleryBloc>().add(
      GalleryFetchPhotosEvent(
        search: GalleryView.categorizedQuery,
        type: 'illustration',
        amount: _page * _pageSize,
      ),
    );
  }

  void _onScroll() {
    if (_scrollController.position.pixels >=
_scrollController.position.maxScrollExtent - kBottomAppBarHeight) {
      _page++;
      _loadPhotos();
    }
  }

  double _getImageHeight(int index) {
    if (!_imageHeights.containsKey(index)) {
      _imageHeights[index] = (Random().nextInt(4) * 60 + 120).toDouble();
    }
    return _imageHeights[index]!;
  }

  @override
  void didChangeDependencies() {
    context.read<GalleryBloc>().add(GalleryDisposeEvent());
    super.didChangeDependencies();
  }
```

```
  @override
  void dispose() {
    _scrollController.dispose();
    super.dispose();
  }

  @override
  void initState() {
    _loadPhotos();
    _scrollController.addListener(_onScroll);
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    double imageWidth = getScreenWidth(context) / 2 - kIndentSmall * 2.5;
    double crossAxisCount = getScreenWidth(context) / (imageWidth);

    return Column(
      children: [

        Expanded(
          flex: 5,
          child: ApplicationHeaderWidget(
            title: 'Illustrations',
            description: 'CURATED GALLERIES',
          ),
        ),

        Expanded(
          flex: 14,
          child: BlocBuilder<GalleryBloc, GalleryState>(
            buildWhen: (previous, current) {
              return current is GalleryLoadedState ||
                     current is GalleryErrorState ||
                     current is GalleryLoadingState && _page == 1;
            },

            builder: (context, state) {
              if (state is GalleryInitialState || state is
GalleryLoadingState && _page == 1) {
                return Center(child: AppLoaderWidget(size:
kLoaderLargeSize));
              }

              if (state is GalleryErrorState) {
                return Center(child: AppLoaderWidget(size:
kLoaderLargeSize));
              }

              else if (state is GalleryLoadedState) {
                final illustrations = state.photos;
                return Padding(
                  padding: const EdgeInsets.only(right: kIndentSmall, left:
kIndentSmall, bottom: kIndentSmall),
                  child: MasonryGridView.count(
                    padding: EdgeInsets.only(top: kIndentSmallDouble,
bottom: kBottomAppBarHeight + kIndentSmallDouble),
                    controller: _scrollController,
                    itemCount: illustrations.length,
                    mainAxisSpacing: kIndentSmall * 1.5,
                    crossAxisCount: crossAxisCount.toInt(),

                    itemBuilder: (context, index) {
```

```
                        final webFormatUrl =
illustrations[index]['webformatURL'] ?? '';
                        return GestureDetector(
                          onTap: () {
                            Navigator.push(
                              context,
                              MaterialPageRoute(
                                builder: (context) {
                                  IllustrationView.illustrationWidth =
getScreenWidth(context);
                                  IllustrationView.illustrationHeight =
getScreenHeight(context) * kInitialDraggableSheetSize;
                                  return IllustrationView(imageData:
illustrations[index]);
                                },
                              ),
                            );
                          },
                          child: GalleryImageWidget(
                            width: imageWidth,
                            height: _getImageHeight(index),
                            url: webFormatUrl,
                          ),
                        );
                      },

                    ),
                  );
                }
                return const SizedBox.shrink();
              },
            ),
          ),
        ],
      );
    }
}
```

// lib/presentation/illustration_page/widgets/ - Тека з файлами-заготовками віджетів для екрану з іллюстрацією та деталями

```
import 'package:creartify/theme/colors.dart';
import 'package:creartify/theme/theme_app.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';
import 'package:gradient_icon/gradient_icon.dart';

class IllustrationDescriptionWidget extends StatelessWidget {
  const IllustrationDescriptionWidget({
    super.key,
    required this.imageData
  });
  final dynamic imageData;

  String processPixabayUrl(String url) {
    url = url.replaceFirst(RegExp(r'https://pixabay\.com(/[^/]+)?/'), '');
    url = url.replaceFirst(RegExp(r'-\d+/'), '');
    url = url.replaceAll('-', ' ');

    if (url.isNotEmpty) {
      url = url[0].toUpperCase() + url.substring(1);
    }

    return url;
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: kIndentDefault * 1.7),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          SizedBox(height: kFabPositionPadding * 1.3),
          Center(
            child: IllustrationLabelWidget(
              style: ThemeApp.textTheme.titleSmall,
              icon: Icons.image_outlined,
              text: " ${processPixabayUrl(imageData['pageURL'])}",
            ),
          ),
          SizedBox(height: kIndentDefault * 1.7),
          IllustrationLabelWidget(
            style: ThemeApp.textTheme.labelSmall,
            icon: Icons.person_outline_rounded,
            text: " Author",
          ),
          Text(imageData['user'], style: ThemeApp.textTheme.labelMedium),

          SizedBox(height: kIndentDefault),
          IllustrationLabelWidget(
            style: ThemeApp.textTheme.labelSmall,
            icon: Icons.file_copy_outlined,
            text: " Type",
          ),
          Text(imageData['type'], style: ThemeApp.textTheme.labelMedium),

          SizedBox(height: kIndentDefault),
          IllustrationLabelWidget(
            style: ThemeApp.textTheme.labelSmall,
            icon: Icons.favorite_border_rounded,
            text: " Likes",
          ),
```

```
          Text(imageData['likes'].toString(), style:
ThemeApp.textTheme.labelMedium),

          SizedBox(height: kIndentDefault),
          IllustrationLabelWidget(
            style: ThemeApp.textTheme.labelSmall,
            icon: Icons.download_outlined,
            text: " Downloads",
          ),
          Text(imageData['downloads'].toString(), style:
ThemeApp.textTheme.labelMedium),

          SizedBox(height: kIndentDefault),
          IllustrationLabelWidget(
            style: ThemeApp.textTheme.labelSmall,
            icon: Icons.bookmarks_outlined,
            text: " Collections",
          ),
          Text(imageData['collections'].toString(), style:
ThemeApp.textTheme.labelMedium),

          SizedBox(height: kIndentDefault),
          IllustrationLabelWidget(
            style: ThemeApp.textTheme.labelSmall,
            icon: Icons.tag_rounded,
            text: " Tags",
          ),
          Text(imageData['tags'], style:
ThemeApp.textTheme.labelMedium?.copyWith(color: AppColors().inactiveDark)),
        ],
      ),
    );
  }
}

class IllustrationIconWidget extends StatelessWidget {
  const IllustrationIconWidget({
    super.key,
    required this.icon,
    this.size = 22,
  });
  final IconData icon;
  final double size;

  @override
  Widget build(BuildContext context) {
    return GradientIcon(
      offset: Offset(0, 1.0),
      icon: icon,
      gradient: ThemeApp.gradientColors(globalColors.activeLight,
globalColors.activeDark),
      size: size,
    );
  }
}

class IllustrationLabelWidget extends StatelessWidget {
  const IllustrationLabelWidget({
    super.key,
    required this.style,
    required this.icon,
    required this.text
  });
  final TextStyle? style;
  final IconData icon;
  final String text;
```

```
  @override
  Widget build(BuildContext context) {
    return Text.rich(
      style: style,
      TextSpan(
        children: [
          WidgetSpan(child: IllustrationIconWidget(icon: icon)),
          TextSpan(text: text),
        ],
      ),
    );
  }
}

class IllustrationOverviewWidget extends StatelessWidget {
  const IllustrationOverviewWidget({super.key});

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Text(
        'Overview',
        style: TextStyle(
            fontFamily: 'TTNorms',
            fontWeight: FontWeight.bold,
            fontSize: 30,
            color: Colors.white
        ),
      ),
    );
  }
}
```

// lib/presentation/illustration_page/illustration_view.dart – Файл з дизайном та функціоналом екрану з іллюстрацією та деталями

```
import 'package:creartify/network/download_script.dart';
import
'package:creartify/presentation/illustration_page/widgets/illustration_descr
iption_widget.dart';
import
'package:creartify/presentation/illustration_page/widgets/illustration_overv
iew_widget.dart';
import 'package:creartify/theme/theme_app.dart';
import 'package:creartify/theme/units.dart';
import 'package:flutter/material.dart';

class IllustrationView extends StatefulWidget {
  static late double illustrationWidth;
  static late double illustrationHeight;
  final dynamic imageData;

  const IllustrationView({
    super.key,
    required this.imageData
  });

  @override
  State<IllustrationView> createState() => _IllustrationViewState();
}

class _IllustrationViewState extends State<IllustrationView> {
  ImageDownloader downloader = ImageDownloader();
  double _dragScrollSheetExtent = 0;
  double _widgetHeight = IllustrationView.illustrationHeight;
  double _widgetWidth = IllustrationView.illustrationWidth;
  double _fabPosition = IllustrationView.illustrationHeight;
  late final DraggableScrollableController _scrollableController;
  late final ScrollController _imageScrollController;
  double _dragOffset = 0;
  bool _isSheetExpanded = false;

  void _expandDraggableSheet() {
    _scrollableController.animateTo(
      _isSheetExpanded ? kInitialDraggableSheetSize :
kMaximumDraggableSheetSize,
      duration: Duration(milliseconds: 300),
      curve: Curves.easeInOut,
    );
    _isSheetExpanded = !_isSheetExpanded;
  }

  void _changeFabPositionOnDrag(
      BuildContext context,
      DraggableScrollableNotification notification)
  {
    _widgetHeight = context.size!.height;
    _widgetWidth = context.size!.width;
    _dragScrollSheetExtent = notification.extent;
    _fabPosition = _dragScrollSheetExtent * _widgetHeight;
  }

  void _imageDragDown(BuildContext context) {
    if (_dragOffset > 110) { Navigator.pop(context); }
    else { setState(() => _dragOffset = 0); }
  }

  void _imageDragUp(DragUpdateDetails details) {
    setState(() {
```

```
      _dragOffset += details.delta.dy;
      if (_dragOffset < 0) _dragOffset = 0;
    });
  }

  @override
  void initState() {
    _scrollableController = DraggableScrollableController();
    _imageScrollController = ScrollController();
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [

          Positioned.fill(
            child: GestureDetector(
              onTap: () {
                _imageScrollController.animateTo(
                  _imageScrollController.position.maxScrollExtent / 2,
                  duration: Duration(milliseconds: 1000),
                  curve: Curves.ease,
                );
              },
              onVerticalDragUpdate: (details) => _imageDragUp(details),
              onVerticalDragEnd: (details) => _imageDragDown(context),
              child: Transform.translate(
                offset: Offset(0, _dragOffset),
                child: SingleChildScrollView(
                  controller: _imageScrollController,
                  scrollDirection: Axis.horizontal,
                  child: Image.network(
                    widget.imageData['fullHDURL'] ??
widget.imageData['largeImageURL'],
                    height: getScreenHeight(context),
                    fit: BoxFit.cover,
                  ),
                ),
              ),
            ),
          ),

          SafeArea(
            child: Padding(
              padding: const EdgeInsets.only(left: kIndentDefault, right:
kIndentDefault, top: kIndentSmall * 2),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  IconButton(
                    onPressed: () => Navigator.pop(context),
                    color: globalColors.primaryForeground,
                    iconSize: kIconSmall,
                    icon: Icon(Icons.arrow_back_ios),
                  ),
                  IconButton(
                    onPressed: () async {
                      await
downloader.downloadFile(widget.imageData['largeImageURL']);
                    },
                    color: globalColors.primaryForeground,
                    iconSize: kIconSmall,
                    icon: Icon(Icons.file_download_outlined),
```

```
                ),
              ],
            ),
          ),
        ),

        NotificationListener<DraggableScrollableNotification>(
          onNotification: (DraggableScrollableNotification notification) {
            setState(() {
              _changeFabPositionOnDrag(context, notification);
            });
            return true;
          },
          child: DraggableScrollableSheet(
            controller: _scrollableController,
            initialChildSize: kInitialDraggableSheetSize,
            minChildSize: kInitialDraggableSheetSize,
            maxChildSize: kMaximumDraggableSheetSize,
            builder: (context, scrollController) {
              return Container(
                decoration: BoxDecoration(
                  color: Colors.white,
                  borderRadius: BorderRadius.vertical(top:
Radius.circular(kRadiusXLarge)),
                ),
                child: SingleChildScrollView(
                  controller: scrollController,
                  child: IllustrationDescriptionWidget(imageData:
widget.imageData),
                ),
              );
            },
          ),
        ),

        Theme(
          data: ThemeData(useMaterial3: true),
          child: Positioned(
            bottom: _fabPosition - kFabPositionPadding * 0.9,
            right: _widgetWidth/4,
            width: _widgetWidth/2,
            height: kFabPositionPadding * 1.8,

            child: FloatingActionButton(
              backgroundColor: Colors.transparent,
              elevation: 0,
              onPressed: _expandDraggableSheet,
              child: Container(
                width: _widgetWidth/2,
                height: kFabPositionPadding * 1.8,
                decoration: BoxDecoration(
                  borderRadius:
BorderRadius.all(Radius.circular(kRadiusDefault)),
                  gradient:
ThemeApp.gradientColors(globalColors.activeLight, globalColors.activeDark),
                ),
                child: IllustrationOverviewWidget(),
              ),
            ),
          ),
        ),
      ],
    ),
  );
 }
}
```