МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МОРОЗЮК АНАСТАСІЯ АНДРІЇВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА ІНТЕРФЕЙСУ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ВЗАЄМОДІЇ КОРИСТУВАЧІВ НА ОСНОВІ СПІЛЬНИХ ІНТЕРЕСІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Т. В. Січко, доцент кафедри
інформаційних технологій,
к. т. н., доцент
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця - 2025

АНОТАЦІЯ

Морозюк А.А. Розробка інтерфейсу мобільного додатку для взаємодії користувачів на основі спільних інтересів. Спеціальність 122 «Комп’ютерні науки», освітня програма «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі проаналізовано теоретичні основи проєктування інтерфейсів мобільних додатків для соціальної взаємодії. Метою роботи є створення інтерфейсу додатка, що допомагає користувачам знаходити однодумців на основі спільних інтересів. Розроблено повноцінний дизайн та реалізовано інтерфейс за допомогою React Native, Expo, TypeScript і системи маршрутизації Expo Router. Подано опис дизайну, структури застосунку та варіанти його подальшого вдосконалення.

Ключові слова: мобільний додаток, інтерфейс, дизайн, React Native.

60 стор., 46 рис., 3 дод., 38 джерел.

ABSTRACT

Moroziuk A.A. Development of a mobile application interface for user interaction based on shared interests. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2025.

The bachelor's qualification thesis analyzes the theoretical foundations of designing mobile application interfaces for social interaction. The aim of the work is to develop an application interface that helps users find like-minded people based on shared interests. A complete design was created, and the interface was implemented using React Native, Expo, TypeScript, and the Expo Router navigation system. The thesis presents the design, application structure, and options for its further improvement.

Keywords: mobile application, interface, design, React Native.

60 pages, 46 figures, 3 appendices, 38 references.

ЗМІСТ

ВСТУП

У сучасному світі цифрових технологій мобільні додатки відіграють ключову роль у формуванні соціальних зв’язків та взаємодії між людьми. Швидкий розвиток онлайн-спільнот і соціальних платформ свідчить про зростаючу потребу у зручних і доступних інструментах для пошуку однодумців. Проте чимало існуючих сервісів мають низку обмежень, зокрема надмірне інформаційне навантаження, недостатню гнучкість персоналізації, а також складність у користуванні через перевантажений або непродуманий інтерфейс. Наведені недоліки знижують загальний рівень взаємодії між людьми та заважають налагодженню тривалих соціальних контактів. Саме тому розробка інтерфейсу мобільного додатка для спілкування на основі спільних інтересів є актуальним завданням, що має важливе практичне значення.

Метою дипломної роботи є створення зручного, естетично привабливого та інтуїтивно зрозумілого інтерфейсу мобільного застосунку, який забезпечуватиме ефективну взаємодію між користувачами. Основний акцент робиться на проєктуванні інтерфейсу, який сприятиме швидкій навігації, легкій комунікації, комфортному перегляду контенту і налаштуванню особистих уподобань.

Для досягнення поставленої мети необхідно реалізувати наступні завдання:

1. Визначити основні вимоги до інтерфейсу мобільного додатка.
2. Проаналізувати сучасні тенденції UI/UX-дизайну мобільних додатків, орієнтованих на соціальну взаємодію.
3. Створити дизайн ключових екранів згідно з принципами зручності та естетики.
4. Реалізувати інтерфейс та функціональні компоненти із використанням сучасних технологій.
5. Проаналізувати отриманий результат та запропонувати можливі шляхи покращення.

Об’єктом дослідження є процес соціальної взаємодії користувачів у цифровому середовищі, що реалізується через мобільні додатки, зокрема, через інтерфейси платформ для спілкування на основі спільних інтересів.

Предметом дослідження є принципи, методи та засоби розробки інтерфейсу мобільного додатка, зокрема UX/UI-підходи до створення зручної навігації, візуальної ієрархії, механізмів пошуку й фільтрації за інтересами, а також елементів персоналізації.

Теоретичне значення дослідження полягає в систематизації знань щодо сучасних практик проєктування інтерфейсів для мобільних додатків, орієнтованих на соціальну взаємодію. Практичне значення полягає у безпосередній розробці інтерфейсу, що забезпечує ефективну взаємодію користувачів на основі спільних інтересів. Розроблені рішення можуть бути використані в подальших проєктах, що спрямовані на створення комфортного цифрового середовища та персоналізацію взаємодії.

Таким чином, дипломна робота орієнтована на вирішення актуальної проблеми покращення користувацького досвіду в соціальних цифрових платформах через розробку функціонального та естетичного інтерфейсу, що відповідає сучасним вимогам до зручності і ефективності взаємодії.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІНТЕРФЕЙСУ МОБІЛЬНОГО ДОДАТКУ

1.1 Аналіз понять і підходів до створення інтерфейсів мобільних додатків

Створення інтерфейсу мобільного додатка є складним процесом, що базується на інтеграції теоретичних концепцій дизайну користувацького досвіду та практичних рекомендацій з проєктування користувацького інтерфейсу. Основною метою є забезпечення інтуїтивної, доступної та приємної взаємодії користувача з цифровим продуктом.

У науковій та професійній літературі поняття «інтерфейс користувача» трактується як сукупність засобів взаємодії користувача з додатком, що включає графічні елементи, навігацію, структуру інформації та способи введення даних [1]. Основними вимогами до інтерфейсу мобільного додатка є простота, передбачуваність, доступність та відповідність очікуванням цільової аудиторії [2].

Особливу увагу в сучасних підходах приділяють адаптивності інтерфейсу до різних пристроїв та умов використання. Згідно з рекомендаціями W3C, інтерфейси повинні забезпечувати доступність контенту для користувачів із різними можливостями, враховуючи обмеження мобільних пристроїв, такі як невеликий розмір екрану, варіативність способів введення та нестабільність з’єднання [3].

Серед основних принципів розробки інтерфейсів мобільних додатків виділяють наступні положення:

- простота та ясність, що полягають у зменшенні когнітивного навантаження через мінімізацію кількості дій, необхідних для досягнення цілі [1];

- консистентність, яка забезпечується використанням узгоджених елементів дизайну та навігаційних шаблонів для сприяння швидшому навчанню користувача [2];
- ефективне використання простору шляхом оптимізації інтерфейсу під обмежений розмір екрану та забезпечення чіткості відображення основної інформації [1];
- адаптивність до контексту використання з урахуванням зміни умов експлуатації, таких як освітлення, доступність мережі або режим використання однією рукою [3].

Важливим аспектом створення інтерфейсів є орієнтація на потреби кінцевого користувача. Методологія проєктування, орієнтована на користувача, передбачає залучення реальних користувачів на всіх етапах створення продукту: від аналізу вимог до тестування прототипів [4]. Такий підхід дозволяє виявити очікування користувачів, їхні труднощі та адаптувати інтерфейс відповідно до реальних сценаріїв використання.

Дослідження принципів UX-дизайну мобільних додатків вказують на важливість створення інтерфейсів, що підтримують емоційний комфорт користувача. Згідно з кращими практиками, зручний мобільний додаток має бути передбачуваним, оперативним та візуально привабливим [2]. Крім того, під час розробки мобільних інтерфейсів необхідно враховувати загальні вимоги до доступності, визначені міжнародними стандартами. Зокрема, важливо забезпечувати можливість масштабування тексту, альтернативні текстові описи для графічних елементів, достатній контраст кольорів та простоту навігації [3].

На підставі викладеного можна сформулювати висновок, що процес створення інтерфейсу вимагає комплексного підходу, який об’єднує теоретичні знання про поведінку користувачів, технічні обмеження мобільних пристроїв та практичні рекомендації щодо організації інформаційної архітектури й візуального дизайну.

1.2 Дослідження існуючих рішень та формування функціональних вимог до інтерфейсу додатка

Проєктування інтерфейсу мобільного застосунку доцільно розпочинати з аналізу наявних аналогів та дослідження практик, які вже довели свою ефективність у подібних системах. Такий підхід дозволяє не лише краще зрозуміти очікування користувачів, а й врахувати сучасні тенденції у сфері UI/UX-дизайну, що, своєю чергою, сприяє формуванню доцільних функціональних вимог.

Одним із ключових аспектів у цьому процесі є дослідження цільової аудиторії. Врахування поведінкових моделей, вподобань та цифрових звичок потенційних користувачів дозволяє сформувати більш релевантний до їхніх потреб інтерфейс [4]. У контексті цього проєкту, оскільки передбачається створення застосунку для соціальної взаємодії на основі спільних інтересів, орієнтація здійснюється на широку вікову аудиторію. Це зумовлює необхідність розробки інтерфейсу, що буде водночас доступним, зрозумілим та інтуїтивним як для молодшого покоління, так і для старших користувачів із різним рівнем цифрової підготовки.

Для кращого розуміння очікувань цільових груп доречно буде здійснити порівняльний аналіз популярних соціальних платформ, таких як Instagram, TikTok, X (Twitter), Threads, Facebook та Reddit. Ці продукти задають стандарти користувацького досвіду, тому їхні інтерфейсні рішення є цінним джерелом для вивчення. Далі буде розглянуто особливості інтерфейсів зазначених платформ, щоб зрозуміти, як кожна з них вирішує завдання зручності та взаємодії з користувачем.

Instagram орієнтований переважно на візуальний контент. Його інтерфейс вирізняється мінімалізмом, продуманою композицією елементів та інтуїтивною логікою навігації. Нижня навігаційна панель включає лише п’ять основних розділів, що забезпечують швидкий доступ до стрічки новин, пошуку, створення публікацій, відеоформату Reels та профілю користувача. Окрему увагу приділено зручності публікації — інтерфейс дозволяє виконати цю дію в кілька

кроків, із доступом до вбудованого редактора. Комунікація реалізована через особисті повідомлення та інтерактивні «історії», що посилює залучення користувачів. Загалом, дизайн спрямований на миттєве сприйняття контенту й просту взаємодію, що забезпечує позитивний користувацький досвід.

Інтерфейс TikTok демонструє інший підхід: його головна мета — максимальне залучення користувачів через безперервну подачу короткого відеоконтенту. Головний екран представлений стрічкою відео з вертикальною прокруткою, що не потребує жодних додаткових дій для споживання контенту. Центральна кнопка на нижній панелі полегшує створення відео, а решта елементів забезпечують доступ до персоналізованої стрічки рекомендацій, пошуку, чатів та профілю. Усі елементи навігації мають високий рівень візуальної доступності, а дизайн зберігає баланс між динамічністю та простотою. Це дозволяє платформі ефективно втримувати увагу користувача впродовж тривалого часу.

Інтерфейс платформи X, раніше відомої як Twitter, побудований за принципами класичної стрічки з текстовими повідомленнями. Основний фокус зроблено на швидкість і простоту створення контенту. Користувач має постійний доступ до функцій створення допису, перегляду сповіщень, пошуку та особистого профілю через нижню панель. Візуальна мова інтерфейсу стримана, з мінімальним використанням графіки, що відповідає функціональності платформи — оперативному обміну думками. Завдяки цьому користувачі швидко орієнтуються в інформаційному потоці, не відволікаючись на вторинні елементи.

Threads — новий застосунок, тісно інтегрований із екосистемою Instagram. Його головною особливістю є зосередженість на коротких текстових повідомленнях, коментарях і відповідях. Інтерфейс вирізняється винятковою простотою: мінімум навігаційних елементів, чітка ієрархія контенту, акцент на діалозі. Дизайн платформи створює відчуття «легкості» взаємодії — користувачі можуть швидко долучатися до розмов або створювати власні повідомлення, що стимулює участь у спільнотах і природну комунікацію.

Facebook є одним із найстаріших гравців на ринку, що значною мірою пояснює його функціональну насиченість. Інтерфейс платформи включає в себе великий набір можливостей: новинна стрічка, групи, події, сторінки, фото, відео та інше. Незважаючи на складність структури, платформа прагне зберегти інтуїтивність навігації шляхом використання вкладених меню, персоналізованих підказок та адаптивного відображення елементів. Такий підхід дозволяє задовольнити потреби широкої аудиторії, включаючи користувачів з різним рівнем технічної обізнаності.

Reddit має унікальну структуру, що ґрунтується на концепції тематичних спільнот і текстово-дискусійному форматі. Інтерфейс виглядає значно простішим у порівнянні з іншими соціальними платформами, однак забезпечує високий рівень кастомізації. Навігація реалізована через вертикальне меню, що дозволяє швидко перемикатися між категоріями, підписками та обговореннями. Акцент робиться на зміст повідомлень, а не на візуальний стиль, що особливо приваблює користувачів, зацікавлених у глибокому тематичному обговоренні. Це робить Reddit ідеальним середовищем для створення спеціалізованих спільнот за інтересами.

На основі цього проведеного аналізу стає можливим сформувати базові функціональні вимоги до інтерфейсу майбутнього застосунку, з урахуванням особливостей цільової аудиторії та завдань, які виконує платформа. Насамперед, важливо відзначити необхідність забезпечення інтуїтивної навігації: інтерфейс має використовувати стандартні, вже знайомі користувачам елементи управління — такі як нижнє меню, свайпи, піктограми швидкого доступу до основних функцій. Це сприятиме зменшенню когнітивного навантаження й полегшуватиме первинне освоєння застосунку навіть для користувачів із низьким рівнем цифрової підготовки.

Особливу увагу необхідно приділити формуванню мінімалістичного дизайну. Це передбачає відмову від перевантаження інтерфейсу декоративними чи надлишковими елементами, натомість фокусуючись на змісті та функціональності.

Важливим принципом є адаптивність — інтерфейс повинен бути однаково доступним як для підлітків, звичних до динамічних форматів взаємодії, так і для старших користувачів, які цінують передбачуваність, логічність структури та наявність підказок. Це можливо досягнути шляхом достатнього контрасту, логічної ієрархії інформації та адаптивної верстки елементів.

Одним із ключових функціональних аспектів є зручність створення контенту. Враховуючи соціальну спрямованість платформи, публікація дописів, коментування, участь у дискусіях повинні бути реалізовані максимально просто — без необхідності проходити кілька етапів чи шукати відповідні функції у складних меню. Доступ до створення нового контенту має бути постійно в полі зору користувача — це може бути плаваюча кнопка або іконка на сторінці користувача.

Не менш важливою є оптимізація продуктивності. У дизайні та реалізації інтерфейсу необхідно уникати перевантажених анімацій, надлишкової графіки чи ресурсомістких ефектів. Натомість пріоритетом є швидкість відкриття екранів, стабільна робота навіть за умов слабкого інтернет-з’єднання та плавність переходів між розділами.

Таким чином, сформульовані функціональні вимоги до інтерфейсу застосунку базуються на принципах доступності, простоти, адаптивності та ефективності. Вони не лише відображають кращі практики провідних платформ, а й враховують потреби широкої аудиторії, що забезпечує високу якість користувацького досвіду незалежно від соціально-вікових відмінностей.

1.3 Вибір інструментів для розробки інтерфейсу мобільного додатка

На початковому етапі проєктування інтерфейсу мобільного додатка доцільним є використання графічного редактора Figma, який на сьогодні вважається одним із найпопулярніших і найбільш функціонально насичених інструментів для створення UI/UX-дизайну. Цей інструмент надає змогу не лише створювати статичні макети інтерфейсів, а й будувати інтерактивні прототипи, що імітують логіку взаємодії користувача з елементами майбутнього застосунку.

Також вибір саме Figma для реалізації поставлених завдань обумовлений і його універсальністю, гнучкістю, наявністю великої бібліотеки готових компонентів, а також можливістю використання різноманітних плагінів для автоматизації процесів дизайну. Платформа забезпечує повний цикл роботи над графічним наповненням проєкту — від побудови прототипу до розробки фінального візуального оформлення [5].

Для реалізації інтерфейсу мобільного додатка доцільно буде використовувати Visual Studio Code як основне середовище розробки. Редактор відзначається зручністю, гнучкістю налаштувань і підтримкою численних розширень, що спрощують написання та налагодження коду. Зокрема, функції автодоповнення, підсвічування синтаксису та інтеграція з інструментами візуалізації інтерфейсу дозволяють ефективно контролювати процес створення інтерфейсних компонентів [6].

Основним фреймворком для створення мобільного додатка обрано React Native, що дозволяє розробляти нативні застосунки для платформ iOS та Android. Для спрощення процесу розробки та управління залежностями буде використано платформу Expo, яка надає набір інструментів та сервісів для ефективної роботи з React Native [7].

Організація навігації між екранами здійснюватиметься за допомогою бібліотеки Expo Router, яка забезпечує зручний механізм маршрутизації та підтримує глибоке посилання, що спрощує навігацію в межах застосунку [8].

Мовою програмування обрано TypeScript, який є надмножиною JavaScript та підтримує статичну типізацію. Завдяки цьому можливим є виявлення помилок ще на етапі компіляції, що значно знижує ризики виникнення критичних збоїв під час виконання застосунку. Це, у свою чергу, підвищує стабільність та надійність програмного забезпечення [9]. Крім того, TypeScript забезпечує кращу підтримку сучасних концепцій об’єктно-орієнтованого програмування, таких як інтерфейси, узагальнення та модулі, що робить розробку інтерфейсу більш структурованою і зручною у модифікаціях в довгостроковій перспективі [10]. Таким чином, вибір TypeScript є виваженим та ефективним рішенням для

створення надійного, масштабованого та зрозумілого у підтримці програмного продукту.

Для трансформації сучасного JavaScript-коду у формат, сумісний з різними середовищами виконання, буде застосовано компілятор Babel. Цей інструмент дає змогу перетворювати новітній синтаксис JavaScript у більш старі версії, які підтримуються широким колом браузерів та середовищ, що особливо важливо для забезпечення кросплатформеної сумісності [11].

Таким чином, обґрунтований вибір інструментів і технологій — від графічного проєктування у Figma до реалізації інтерфейсу у середовищі Visual Studio Code з використанням React Native, Expo, TypeScript та супровідних засобів — забезпечує комплексний підхід до розробки сучасного, зручного та ефективного інтерфейсу мобільного застосунку. Поєднання візуального дизайну і функціонального коду створює передумови для досягнення високої якості кінцевого продукту.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ МОБІЛЬНОГО ДОДАТКА

2.1 Формування структури і навігації мобільного додатка

Побудова логічної структури та зручної навігації в мобільному застосунку є одним із ключових чинників, що визначають ефективність його використання, доступність функціоналу та загальний досвід користувача. Враховуючи рекомендації сучасних підходів до UX-дизайну, у межах розробки застосунку було сформовано багаторівневу систему навігації, адаптовану до поведінкових сценаріїв користувачів [12].

Загальна логіка побудови навігації представлена на схемі далі (рис. 2.1).

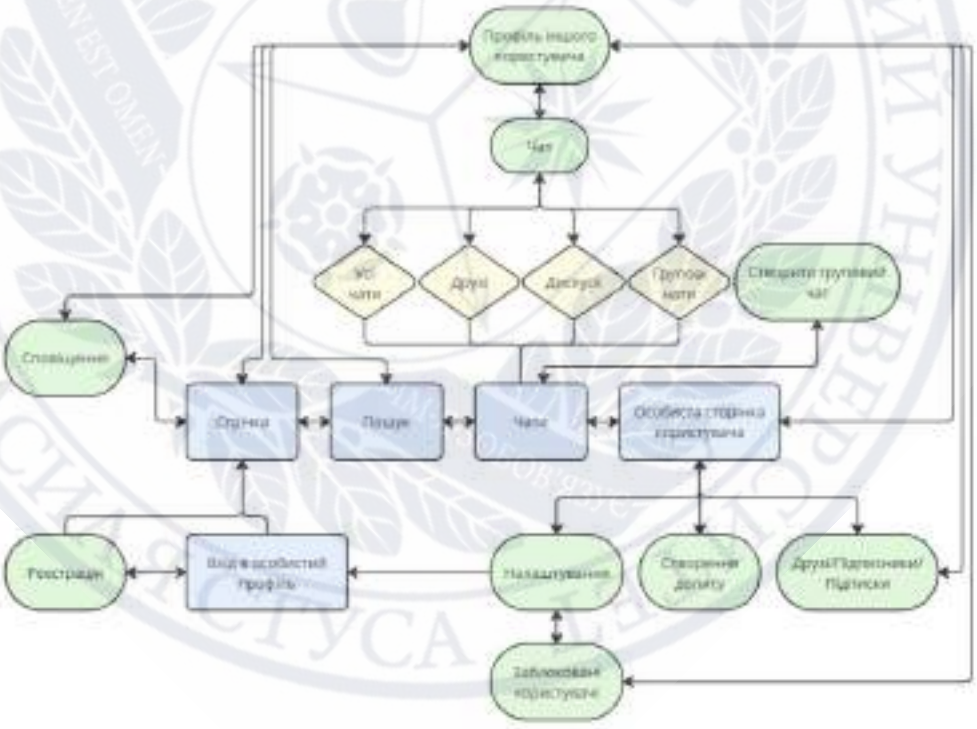

Рисунок 2.1 – Структура та навігація застосунку

Основним елементом початкової взаємодії є екран авторизації, що також містить перехід на сторінку реєстрації. Така структура відповідає вимогам інтуїтивного доступу до створення облікового запису у разі його відсутності. Передбачено можливість повернення до попереднього екрану, у випадку

сторінки реєстрації, що сприяє свободі навігації та не обмежує користувача у виборі подальших дій.

Користувацький шлях побудовано з урахуванням принципів юзабіліті, зокрема — мінімізації кількості дій для досягнення мети, що є одним з ключових факторів ефективності інтерфейсів [12]. Перехід до основного функціоналу застосунку відбувається виключно після успішної автентифікації, що також відповідає сучасним вимогам безпеки даних та структурної логіки цифрових сервісів.

Після успішного входу в акаунт або ж завершення реєстрації відбувається автоматичний перехід на одну із головних сторінок застосунку — «Стрічку». Цей розділ виконує роль стартової точки взаємодії користувача з основним контентом платформи. Реалізовано швидкий доступ до чотирьох ключових розділів інтерфейсу: «Стрічка», «Пошук», «Чати» та «Особиста сторінка користувача». Така структура забезпечує зручність орієнтації та відповідає логіці типових сценаріїв використання застосунку.

Особливістю сторінки «Стрічка» є наявність функціональної кнопки сповіщень. Натискання на неї ініціює перехід на окремий розділ зі списком сповіщень, що структурно відокремлений від основної навігаційної системи. Водночас, реалізовано інтуїтивно зрозумілий механізм повернення до попереднього екрану — «Стрічки». Це рішення відповідає принципам передбачуваного переміщення, зменшує ймовірність помилкових дій та гарантує контроль користувача над переміщенням між екранами.

Окрім того, кожен допис у стрічці містить інтерактивні елементи навігації, що дозволяють здійснити перехід до персонального чату з автором або до сторінки обговорення відповідного допису. Такі переходи розширюють структуру переміщення на рівень вкладених екранів і забезпечують доступ до контекстно пов’язаного функціоналу. Це відповідає принципу «глибокої навігації», коли користувач має можливість гнучко та послідовно занурюватися у пов’язані розділи застосунку, не втрачаючи логічного зв’язку між діями.

Розділ «Пошук» орієнтований на виконання запитів щодо дописів та користувачів. Це забезпечує простий і швидкий доступ до іншого контенту, зберігаючи логічну послідовність дій. Профілі відкриваються при натисканні як окремі сторінки, з яких користувач може повернутись назад до пошуку, що підтримує лінійність навігації та запобігає втраті контексту пошуку.

У розділі «Чати» реалізовано перемикання між кількома категоріями листування: «Усі чати», «Друзі», «Дискусії» та «Групові чати». Передбачено також функцію створення групового чату, активація якої переносить користувача на окрему сторінку для вибору учасників та завершення створення групи. Після цього користувач автоматично повертається до основного списку чатів.

Усередині кожного чату можна перейти на сторінку користувача, аби отримати про нього більш детальну інформацію.

Сторінка особистого акаунта містить доступ до налаштувань профілю, зокрема функції виходу та видалення облікового запису, після чого користувач повертається на сторінку авторизації. Крім того, реалізовано можливість створення нового допису через окрему сторінку з формою, яка відкривається при натисканні відповідної кнопки. Також зі сторінки акаунта можна переглядати списки друзів, підписок і підписників, доступ до яких здійснюється через відповідні лічильники. Вони відкриваються як окремі функціональні екрани, з яких передбачено лише повернення на особисту сторінку користувача.

Запропонована навігаційна структура побудована відповідно до принципів дизайну орієнтованого на користувачів, що передбачає адаптацію під потреби та звички цільової аудиторії [13]. Завдяки чітко визначеній ієрархії екранів, реалізованій можливості повернення назад і доступу до ключових функцій через основну навігацію, забезпечено логічну послідовність користування та зменшено ризики помилок під час переміщення. Структура також передбачає відокремлення специфічних дій на окремі сторінки, що сприяє концентрації користувача на поточному завданні, відповідно до принципу «одна задача — один екран» [14].

2.2 Аналіз сучасних тенденцій у дизайні мобільних додатків

Досвід взаємодії користувача з мобільним додатком є визначальним чинником успіху цифрового продукту та тривалості співпраці між користувачем і компанією. У зв’язку з цим дизайн інтерфейсу, структура навігації та інші елементи взаємодії повинні бути інтуїтивно зрозумілими, візуально привабливими та відповідати трендам. Сучасні тенденції у сфері дизайну інтерфейсів мобільних застосунків спрямовані не лише на забезпечення функціональності, а й на створення приємного, гармонійного користувацького досвіду, який впливає на рівень задоволеності, лояльності та залученості аудиторії.

Проте, варто зазначити, що лише візуально привабливий інтерфейс не є гарантією успішності продукту. Якщо застосунок не відповідає очікуванням користувачів щодо функціоналу або, наприклад, має труднощі в навігації, ймовірність його використання знижується, незважаючи на приємний зовнішній вигляд. Саме тому провідні компанії дотримуються принципу балансу між формою та змістом — дизайн має не лише привертати увагу, а й ефективно вирішувати задачі користувача.

Оскільки цифрове середовище розвивається надзвичайно динамічно, тенденції в дизайні інтерфейсів також постійно змінюються. Те, що було трендом учора, сьогодні може вважатися застарілим. У зв’язку з цим компанії повинні постійно стежити за розвитком ринку, оновлювати свої підходи до дизайну, впроваджувати новітні інструменти та практики. Ігнорування змін у вподобаннях користувачів або технологічних інноваціях може призвести до втрати конкурентоспроможності.

Однією з ключових сучасних тенденцій у дизайні мобільних інтерфейсів є мінімалізм. Цей підхід передбачає свідоме зменшення кількості візуальних елементів, обмежену кольорову гаму, лаконічну типографіку та акцент на функціональність. У межах мінімалістичного дизайну важливим є створення чіткої візуальної ієрархії, яка виділяє основні елементи, водночас уникаючи надмірної декоративності [15]. Такий підхід особливо цінний у контексті

мобільних платформ, де розмір екрана обмежений, а швидке сприйняття інформації є критично важливим. Мінімалізм також сприяє інклюзивності: завдяки простому і зрозумілому інтерфейсу додатки стають зручнішими для людей похилого віку та користувачів з різними рівнями цифрової грамотності.

З технічного боку, мінімалістичний дизайн має низку переваг. Він дозволяє зменшити навантаження на пристрій, що позитивно впливає на швидкість завантаження, плавність роботи інтерфейсу та економію ресурсів. Крім того, мінімалізм сприяє уніфікації дизайну під різні платформи, оскільки чиста та структурована візуальна мова легко адаптується до різних розмірів екранів і типів пристроїв [15].

Однак надмірне спрощення може мати зворотний ефект. У гонитві за лаконічністю важливо не втратити функціональність і легке розуміння інтерфейсу [16]. Мінімалізм має бути продуманим і відповідати логіці взаємодії користувача, а не лише візуальним уподобанням дизайнерів.

Наступною поширеною тенденцією у дизайні мобільних інтерфейсів є використання анімації. Якщо раніше вона виконувала переважно декоративну функцію, то з 2019 року її значення змінилося у бік функціональної підтримки взаємодії користувача з інтерфейсом [16]. Сьогодні анімація активно використовується для покращення зручності використання, орієнтації в застосунку, передачі емоційного контексту або простої візуалізації змін у статусі елементів інтерфейсу [17]. Анімація виконує кілька важливих функцій:

- створює відчуття динаміки та «живого» інтерфейсу;
- сприяє кращому розумінню логіки навігації;
- допомагає новим користувачам легше адаптуватися;
- може направляти увагу до важливих елементів або змін у додатку.

Однак, впровадження анімацій потребує уважного балансу: надмірна або занадто складна анімація може негативно вплинути на швидкодію додатка, збільшити навантаження на процесор і пам'ять мобільного пристрою, а отже, погіршити загальний досвід користування.

Серед популярних типів анімації, що використовуються в мобільних додатках, можна виділити [15]:

- мікроанімація — невеликі, але помітні рухи елементів, які служать підказками або зворотним зв'язком (наприклад, сповіщення про додавання до кошика, зміна стану кнопки тощо);
- рухомі фони — створюють ефект глибини або простору, підсилюючи візуальне сприйняття інтерфейсу. Часто використовуються в заставках або головних екранах;
- анімація переходів — полегшує перемикання між екранами, дозволяє користувачеві краще зрозуміти зв'язок між елементами. Це може бути ефект «зсуву», «розгортання», «масштабування» тощо;
- інтерактивна анімація — активується під час взаємодії користувача з інтерфейсом (наприклад, натискання кнопок, переміщення об'єктів);
- анімовані ілюстрації — використовуються як декоративний або емоційний елемент (наприклад, під час завантаження або як частина ознайомлення із додатком).

Загалом, якісно використана анімація підсилює інтуїтивність інтерфейсу, покращує зворотний зв’язок із користувачем і формує позитивне емоційне сприйняття продукту.

Ще однією актуальною тенденцією у створенні інтерфейсів мобільних додатків є впровадження темного режиму. Він стрімко набирає популярності, зокрема завдяки своїм ергономічним та естетичним перевагам. Темний режим забезпечує зниження навантаження на зір, особливо в умовах низького освітлення, а також сприяє економії енергії на пристроях з OLED-екранами, де темні пікселі споживають менше енергії або повністю вимикаються [18].

Темний режим сьогодні часто розглядається не лише як стильовий вибір, а як важлива опція доступності. Багато користувачів обирають саме темний інтерфейс з особистих причин комфорту або медичних показань.

Серед основних дизайнерських принципів реалізації темного режиму виділяють [15]:

- високу контрастність — необхідна для чіткого відображення елементів інтерфейсу на темному тлі;
- монохроматичність — використання обмеженої кількості відтінків однієї гами, щоб уникнути візуального перевантаження;
- акцентування — яскраві або контрастні кольори (наприклад, синій, зелений, бірюзовий в поєднанні із жовтим, помаранчевим, червоним) використовуються вибірково для виділення важливих елементів або індикаторів дії.

Темний режим часто вбудовується як альтернатива світлому інтерфейсу з можливістю перемикання вручну або автоматично відповідно до системних налаштувань пристрою.

Наступною важливою тенденцією в розробці інтерфейсів мобільних додатків є впровадження дизайн-системи. Дизайн-система — це цілісний набір елементів, інструментів, принципів і правил, що використовуються для створення єдиного, послідовного дизайну продукту [15]. Її роль у сучасному процесі розробки важко переоцінити, адже вона уніфікує візуальний стиль, зменшує кількість повторюваних рішень і значно пришвидшує робочі процеси як для дизайнерів, так і для розробників.

Завдяки чітко визначеним компонентам — як-от палітри кольорів, типографіка, відступи, UI-компоненти (кнопки, поля введення тощо) — команда уникає постійного «вигадування з нуля» та має змогу зосередитися на функціональності та користувацькому досвіді. Наприклад, питання про те, де саме розміщувати кнопку або який шрифт обрати, вже вирішується за допомогою вбудованих шаблонів і заздалегідь узгоджених правил.

Окрім дизайнерів і розробників, дизайн-система є корисною і для інших учасників процесу створення продукту — контент-мейкерів, маркетологів, фрілансерів і навіть зовнішніх партнерів. Вона забезпечує узгодженість

зовнішнього вигляду, підсилює ідентичність додатка та полегшує підтримку в довготривалій перспективі.

Ще однією актуальною тенденцією в дизайні мобільних інтерфейсів є використання градієнтів. Після періоду домінування мінімалізму з його стриманою палітрою кольорів, дизайнери дедалі частіше повертаються до виразних, яскравих рішень, зокрема — до градієнтів [16]. Цей підхід дозволяє надати глибини, об’єму та візуальної динаміки окремим елементам інтерфейсу, зробивши загальний вигляд додатка більш привабливим.

Сучасні градієнти вже не обмежуються ніжними переходами відтінків одного кольору — у тренді неонові, насичені та контрастні кольори, які використовуються для [16]:

- виділення кнопок (наприклад, через підсвічування або ефект світіння);
- формування фону, що створює візуальний акцент без перевантаження;
- створення ефекту «слона в кімнаті», тобто м’якого візуального поділу зон або інтерфейсних блоків шляхом поступових переходів між кольорами.

Такий підхід також дозволяє візуально структурувати інтерфейс — вказати користувачам, на що саме варто звернути увагу, а що є фоновим або допоміжним. Однак, як і з будь-яким виразним дизайнерським засобом, важливо дотримуватись балансу. Надмірне використання градієнтів, особливо у поєднанні з великою кількістю яскравих кольорів, може створити візуальний шум, що ускладнює навігацію та негативно впливає на користувацький досвід.

Наступною важливою тенденцією є використання 3D-дизайну в інтерфейсах мобільних додатків. Завдяки розвитку графічних технологій та покращенню продуктивності мобільних пристроїв, тривимірна графіка стала доступнішою для широкого впровадження. 3D-дизайн дозволяє створювати більш реалістичні, деталізовані та інтерактивні елементи, які значно підвищують естетичну привабливість і занурення користувача в середовище додатка. Типовими прикладами застосування є:

- анімації 3D-об’єктів, які реагують на взаємодію користувача;

- тривимірні іконки з глибиною та тінями;
- інтерактивні елементи інтерфейсу, що обертаються або трансформуються під час свайпу чи натискання.

Такі елементи можуть оживити інтерфейс, зробити його унікальним і виділити продукт на фоні конкурентів. Зокрема, вони ефективні у сферах, де важливе візуальне представлення об’єктів: освіта, дизайн, мода, архітектура, ігри, віртуальна реальність. Однак, слід пам’ятати і про недоліки надмірного використання 3D-графіки. Вона може [15]:

- погіршити продуктивність додатка, особливо на пристроях зі слабкими характеристиками;
- збільшити споживання енергії та навантаження на процесор;
- викликати візуальний дискомфорт у користувачів із проблемами зору або чутливістю до руху (наприклад, при схильності до «морської хвороби»).

Таким чином, 3D-дизайн є потужним інструментом для залучення уваги та покращення взаємодії, але його застосування вимагає ретельного балансу між привабливістю та ергономічністю.

Ще однією актуальною тенденцією у дизайні інтерфейсів мобільних додатків є використання ілюстрацій та графічних елементів. Цей підхід не лише додає візуальної привабливості, а й виконує важливу комунікативну функцію, допомагаючи користувачам швидше та легше сприймати інформацію, особливо складну або абстрактну. Ілюстрації сьогодні активно застосовуються для:

- передачі настрою бренду;
- пояснення концепцій або функціоналу застосунку;
- створення унікального стилю.

Сучасні ілюстрації часто мають абстрактний вигляд, використовують нестандартні кольорові поєднання, грайливі форми та нестандартні композиції, що відходять від традиційних схем і створюють емоційний зв’язок із користувачем. Наприклад, у застосунках для дітей освітнього спрямування доречним буде використання яскравих, барвистих ілюстрацій, які не лише

утримуватимуть увагу, але й полегшать процес навчання через візуальне підкріплення інформації. Ілюстрації в такому випадку виконують подвійну роль — естетичну та педагогічну. Також популярністю користуються:

- ілюстрації персонажів, які можуть стати «обличчям» застосунку;
- ілюстровані повідомлення про помилки або успіхи, що замінюють сухі тексти на більш дружні та зрозумілі візуальні елементи;
- анімаційні ілюстрації, які роблять взаємодію з додатком цікавішою.

Водночас важливо не перевантажувати інтерфейс, зберігаючи баланс між візуальним стилем і функціональністю [15].

Ще однією важливою тенденцією в дизайні мобільних додатків є впровадження голосових інтерфейсів. Такий підхід дедалі активніше використовується в сучасних застосунках, оскільки забезпечує швидку, зручну та безконтактну взаємодію з функціоналом, особливо в умовах багатозадачності або для користувачів з обмеженими можливостями [19]. Голосовий інтерфейс дозволяє здійснювати низку типових дій за допомогою голосових команд, зокрема:

- запуск програм;
- пошук інформації;
- відправлення повідомлень;
- планування зустрічей та подій у календарі;
- керування додатком без рук (наприклад, у водіїв або людей з інвалідністю).

Одними з найвідоміших прикладів голосових інтерфейсів є Siri від Apple, Google Assistant від Google, Alexa від Amazon.

У мобільних застосунках голосове керування може бути частиною загальної UX-стратегії, особливо якщо це пов’язано з доступністю, інклюзивністю або потребами певних цільових аудиторій. Такий підхід також сприяє створенню персоналізованого досвіду, оскільки системи зі штучним інтелектом можуть адаптуватися до голосових запитів конкретного користувача. Проте впровадження таких інтерфейсів вимагає особливої уваги до:

- точності розпізнавання мови;
- мовної підтримки;
- розумного діалогового дизайну, що дозволяє уникнути непорозумінь і помилок у взаємодії.

Не менш поширеною тенденцією в дизайні мобільних додатків є плоский дизайн. Цей підхід орієнтований на максимальне спрощення візуальних елементів: замість об’ємних форм, складних текстур і реалістичних деталей використовуються чисті лінії, прості геометричні форми, мінімальні кольорові палітри та чітка типографіка. Особливості плоского дизайну [16]:

- спрощення інтерфейсу, що робить його легким для сприйняття;
- використання тонких градієнтів і тіней, які додають об’єм, зберігаючи мінімалістичність;
- відмінна сумісність з анімацією — завдяки своїй простоті, плоскі елементи легко анімувати;
- сучасний та естетичний вигляд, що добре підходить для більшості сфер застосування.

Попри свої переваги, плоский дизайн має і деякі виклики [15]:

- через візуальну одноманітність складно виділити додаток серед конкурентів, що теж дотримуються цього стилю;
- інтерфейс може стати неінтуїтивним — без тіней, глибини та підказок деякі інтерактивні елементи (наприклад, кнопки або меню) можуть виглядати як простий текст чи фон, що ускладнює навігацію для нових користувачів;
- потрібен особливий підхід до балансу між простотою і функціональністю, щоб користувачі не губилися в інтерфейсі.

Плоский дизайн часто використовується як основа для створення гнучких, адаптивних інтерфейсів, які добре виглядають на екранах будь-якого розміру, що робить його актуальним для розробників кросплатформених рішень.

Наступною та останньою актуальною тенденцією є використання технологій розширеної реальності (AR). Вона відкриває нові можливості у взаємодії з мобільними додатками, дозволяючи користувачам бачити віртуальні об'єкти у реальному світі через камеру свого смартфона. Завдяки цьому зростає рівень занурення у цифровий досвід, а сам застосунок стає більш інтерактивним та цікавим [20]. Основні переваги AR у мобільних додатках:

- покращення користувацького досвіду через реалістичну візуалізацію;
- інтуїтивна взаємодія з додатком за рахунок «фізичної присутності» цифрових елементів у навколишньому середовищі;
- інноваційність — подібний підхід одразу вирізняє додаток серед конкурентів.

Сфери застосування [15]:

- навігація — наприклад, відображення вказівок безпосередньо на екрані в реальному середовищі для кращої орієнтації користувача;
- мода та ритейл — можливість приміряти віртуальні речі або бачити, як меблі виглядатимуть у кімнаті;
- освіта — наочне пояснення складних понять через інтерактивні 3D-моделі;
- ігри — створення ігрових світів, які «оживають» у реальному середовищі, як у випадку з Pokemon GO.

AR вимагає високої обчислювальної потужності та стабільної роботи камери, тому важливо оптимізувати такі додатки для забезпечення плавного функціонування. Однак при правильній реалізації розширена реальність суттєво розширює функціонал і привабливість продукту, а також підвищує залученість користувача.

Варто зазначити, що популярність дизайнерських підходів є динамічною — тренди змінюються під впливом нових технологій, змін у поведінці користувачів та особливостей цифрової екосистеми. Усі перелічені тенденції мають схильність втрачати актуальність з часом, і це значною мірою залежить від таких чинників, як категорія застосунку, цільова аудиторія та професійне

впровадження сучасних рішень у дизайн. Тому при створенні інтерфейсу важливо не лише дотримуватись трендів, а й передбачати можливі зміни у сприйнятті та зручності використання, а також аналізувати наслідки впровадження певного підходу. Основна увага має бути зосереджена на потребах кінцевих користувачів: їхньому віці, сфері діяльності, щоденних завданнях, а також на врахуванні доступності — зокрема для людей із вадами зору, слуху чи моторики [21]. Такий орієнтований на користувача підхід дозволяє створювати не лише сучасний, але й справді функціональний і комфортний інтерфейс.

2.3 Створення дизайну інтерфейсу мобільного додатка

Процес проєктування дизайну мобільного застосунку доцільно розпочинати зі створення прототипу, який дозволяє візуалізувати структуру інтерфейсу, розміщення елементів керування та сценарії користувацької взаємодії. Прототипування на ранніх етапах розробки дозволяє виявити потенційні недоліки навігації, оцінити логіку переходів між екранами та адаптувати інтерфейс до очікувань цільової аудиторії без значних часових чи ресурсних витрат. Як зазначено у рекомендаціях компанії Apple щодо створення інтерфейсів мобільних застосунків, прототипування є ключовим етапом, що дає змогу валідувати рішення до їх реалізації в остаточному дизайні [22].

З урахуванням цього підходу, прототипи були розроблені для всіх основних екранів, зокрема з урахуванням логіки навігації, подання інформації та взаємодії користувача з елементами інтерфейсу. Надалі всі дизайни будуть подані у вигляді зображень із відповідними прототипами.

З огляду на те, що цільова аудиторія продукту є широкою за віком, уподобаннями та цифровою грамотністю, ключовим завданням дизайну є забезпечення інтерфейсу, який буде доступним, нейтральним та інтуїтивно зрозумілим для різних категорій користувачів. Одним із визначальних чинників при створенні такого інтерфейсу є вибір кольорової гами, оскільки саме колір здатен значно впливати на емоційне сприйняття застосунку, викликати довіру або, навпаки, відторгнення [23].

Враховуючи бажання створити спокійне, комфортне середовище для взаємодії, було вирішено уникати агресивних, надто яскравих або психологічно навантажених кольорів (наприклад, насиченого червоного або темного фіолетового), що можуть викликати тривожність, напругу або асоціюватися з помилками [24]. Основою палітри стали нейтральні білі та сірі тони, які добре сприймаються візуально, підтримують чистоту інтерфейсу та дозволяють легко комбінувати інші кольори для акцентів.

Для виділення ключових елементів інтерфейсу, зокрема кнопок та активних компонентів, було обрано блакитний колір із кодом #7EAAED. Згідно з дослідженнями психології кольору, блакитні відтінки асоціюються зі спокоєм, довірою, стабільністю та професійністю, що робить їх популярними в інтерфейсах соціальних застосунків, освітніх платформ і фінансових сервісів. Колір #7EAAED має середню насиченість, що дозволяє використовувати його як акцент без створення візуального перевантаження [25].

Важливим технічним аспектом є забезпечення достатнього рівня контрастності між кольором тексту та фоном відповідно до стандартів доступності. З цією метою колір #7EAAED було перевірено за допомогою онлайн-інструмента WebAIM Contrast Checker, що дозволяє визначити відповідність кольорових поєднань критеріям WCAG 2.1. Отримане значення контрасту становило 2.37:1 у поєднанні зі стандартним білим текстом, що свідчить про наявність акценту при збереженні візуального комфорту. Однак для текстових елементів у цьому кольорі рекомендується додатково підвищувати контрастність або використовувати його лише як фоновий або декоративний колір, не призначений для читання дрібного тексту [26].

Одним із ключових елементів інтерфейсу є навігаційна панель, яка забезпечує швидкий доступ до основних розділів (рис. 2.2). Оскільки більшість користувачів взаємодіятимуть із додатком саме на смартфонах, важливо враховувати ергономіку — усі елементи повинні бути розташовані в зоні зручного доступу. За рекомендаціями Nielsen Norman Group, одним із найкращих

варіантів для мобільних інтерфейсів є розміщення навігаційної панелі внизу екрана, де її легко охопити великим пальцем [27].

Рисунок 2.2 – Головна навігаційна панель

З урахуванням функціональної структури застосунку, навігаційна панель включатиме чотири основні пункти: «Стрічка», «Пошук», «Чати» та «Профіль». Такий порядок дозволяє швидко перемикатися між ключовими розділами платформи.

Першою сторінкою, що відкривається після запуску програми, є «Стрічка» (рис. 2.3). Вона призначена для перегляду публікацій, і саме з неї користувач отримує основну частину контенту. У верхній частині екрана передбачено перемикач між трьома вкладками: «Рекомендації», «Друзі» та «Підписки» (рис. А.1). Це дає змогу персоналізувати перегляд, швидко обираючи бажаний тип дописів. Поруч також розміщено іконку сповіщень, яка інформує про нові події, пов’язані з активністю в додатку.

Рисунок 2.3 – Сторінка «Стрічка»

Основний простір сторінки займають дописи — візуально оформлені блоки з зображеннями, описами та елементами взаємодії. Усі елементи

інтерфейсу виконані у мінімалістичному стилі — з акцентом на чистоту, простоту форм і легкість сприйняття. Такий підхід, як було обґрунтовано у підрозділі 1.3, сприяє створенню інтуїтивно зрозумілого середовища та не відволікає користувача від основного вмісту.

Для візуалізації публікацій було обрано картковий формат розміщення контенту. Незважаючи на те, що більшість сучасних соціальних мереж, включно з проаналізованими раніше, використовують класичну стрічкову подачу інформації, цей підхід має певні недоліки. Зокрема, стрічковий формат сприяє надмірному навантаженню на увагу користувача, оскільки змушує його одночасно обробляти велику кількість інформаційних блоків. У результаті значна частина дописів залишається проігнорованою, оскільки користувач підсвідомо обирає ті, що візуально більше привертають увагу [28].

Більше того, постійне гортання стрічки та пошук найбільш привабливого контенту негативно впливає на концентрацію уваги, спричиняючи зниження здатності до тривалого аналізу окремих інформаційних одиниць. З часом це призводить до формування звички неконтрольованого споживання великої кількості контенту з надією на знаходження матеріалу, що викликатиме сильні емоційні реакції або задовольнятиме інші імпульсивні потреби.

Виходячи з цього, під час створення дизайну було поставлено за мету забезпечити максимальну концентрацію уваги користувача на кожному окремому дописі. Для досягнення цього завдання публікації розміщуватимуться у вигляді окремих карток, що займатимуть центральне положення на екрані [20]. Перехід між дописами здійснюватиметься за допомогою жесту свайпу вгору, після чого поточна картка плавно зникатиме з поля зору, а наступна займатиме її місце. Такий механізм взаємодії дозволяє не лише зменшити когнітивне навантаження, але й стимулювати усвідомлену взаємодію з кожною одиницею контенту окремо, що позитивно позначається на загальній якості користувацького досвіду.

Наступна сторінка — «Пошук», яка у мобільному додатку призначена для забезпечення ефективного та інтуїтивно зрозумілого пошуку контенту (рис. 2.4).

На ній користувачі можуть знаходити як конкретних осіб, так і дописи за введеними ключовими словами. Основним елементом інтерфейсу є поле для введення тексту, розташоване у верхній частині екрана, що забезпечує швидкий доступ до функції пошуку. Поряд із полем введення розміщені фільтри, які дозволяють сортувати результати за актуальністю та часом публікації, сприяючи точнішому відображенню релевантного контенту (рис. А.4 – А.5).

Рисунок 2.4 – Сторінка «Пошук»

З метою покращення користувацького досвіду та забезпечення чіткої візуальної ієрархії, елементи інтерфейсу поділено на дві категорії: інтерактивні та неінтерактивні. Інтерактивні компоненти, такі як профілі користувачів або дописи, що ведуть до окремих сторінок, мають об'ємний дизайн, що створює ефект необхідного натискання. Це відповідає принципам skeuomorphic-дизайну, який використовує візуальні підказки для позначення функціональності елементів. Такий підхід підтверджується дослідженнями, які свідчать, що об'ємні елементи покращують розпізнавання та взаємодію користувачів з інтерфейсом [30].

Натомість неінтерактивні елементи, такі як статичні дописи, оформлені у плоскому стилі без додаткових візуальних ефектів. Це відповідає принципам flat-дизайну, який акцентує увагу на простоті та мінімалізмі, зменшуючи когнітивне навантаження на користувача. Дослідження показують, що плоский дизайн сприяє кращій сприйнятливості та ефективності взаємодії, особливо в умовах обмеженого екранного простору мобільних пристроїв [31].

Таким чином, поєднання skeuomorphic та flat-дизайну на сторінці «Пошук» забезпечує баланс між інтуїтивністю та естетикою, сприяючи покращенню загального користувацького досвіду.

Наступним розділом інтерфейсу є сторінка «Чати» (рис. 2.5). Її ключовим завданням є забезпечення зручної навігації між різними типами чатів, зокрема: «Всі чати», «Друзі», «Дискусії» та «Групові чати». Для реалізації цієї функціональності було обрано верхню горизонтальну навігаційну панель, яка містить відповідні кнопки-перемикачі, розташовані у вигляді рядка. Актуальним у процесі проєктування постало питання візуального виділення активного фільтра. З метою дотримання єдиного стилістичного підходу, для активної кнопки було застосовано вже використаний раніше дизайнерський прийом, реалізований на сторінці «Пошук» у вигляді кнопок «Підписатись»/»Відписатись». Зокрема, активна кнопка має білий фон, темніший колір тексту та контурну рамку, що дозволяє чітко її вирізняти серед інших елементів. Водночас неактивні фільтри залишаються стилістично нейтральними.

Оскільки користувач може мати потребу знайти конкретну розмову серед наявних чатів, було передбачено наявність пошукового поля. Поруч з ним розташовано кнопку створення нового чату. З метою забезпечення кращого користувацького сприйняття усі іконки на сторінці підбиралися згідно з принципом візуальної асоціативності, як і на інших функціональних екранах застосунку.

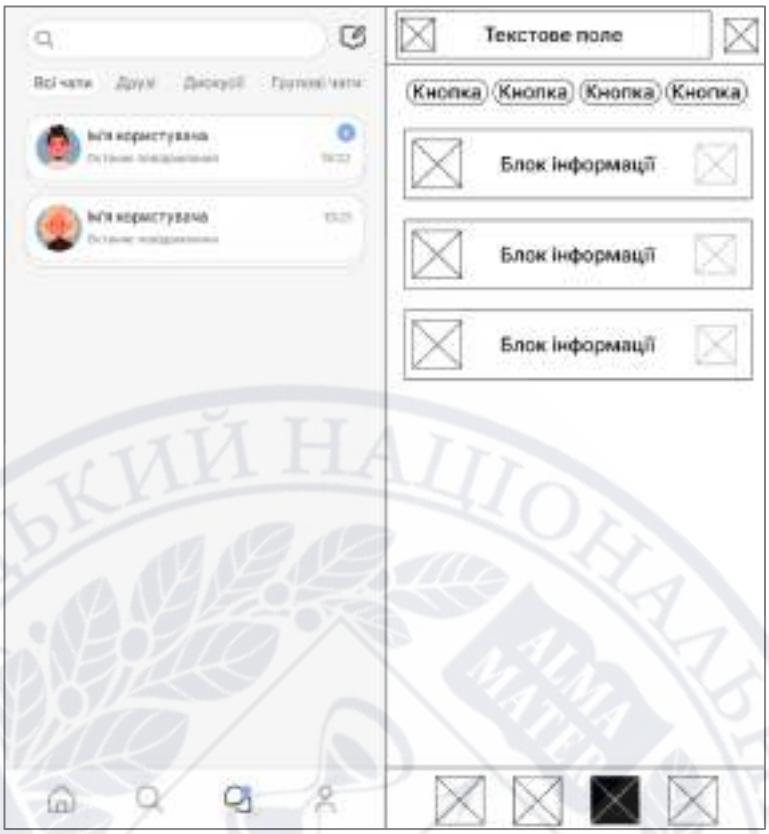

Рисунок 2.5 – Сторінка «Чати»

Нижче навігаційної панелі відображаються самі чати. Їх візуальне оформлення дещо базується на дизайні елементів облікових записів користувачів, із застосуванням об’ємної стилізації для збереження візуальної послідовності всього інтерфейсу, також дотримання єдиного стилістичного напрямку.

Окрему увагу було приділено відображенню сповіщень про нові повідомлення. Для цього на іконці «Чати» у головній навігаційній панелі реалізовано маркер у вигляді кола. Такий підхід дозволяє забезпечити швидке візуальне розпізнавання наявності нових повідомлень незалежно від того, на якій головній сторінці програми перебуває користувач.

Завершальною серед основних інтерфейсних секцій платформи є сторінка профілю користувача (рис. 2.6). З точки зору дизайну, цей розділ є одним з найскладніших, оскільки повинен містити значну кількість функціональних елементів, зберігаючи при цьому чисту візуальну структуру та уникати

перевантаження інформацією. Подібно до інших сторінок, інтерфейс профілю структурований за принципом вертикального розміщення елементів.

Рисунок 2.6 – Сторінка «Акаунт користувача»

У верхній частині сторінки розташовується нікнейм користувача, кнопки створення допису та налаштувань. Розміщення нікнейма у верхньому сегменті є усталеним патерном в інтерфейсах соціальних платформ, оскільки дозволяє користувачеві легко зорієнтуватися, підтверджуючи, що він перебуває саме на власній сторінці.

Наступним блоком є візуальна ідентифікація користувача, що включає: аватар, ім’я користувача, короткий опис, який створюється під час реєстрації та може змінюватися в налаштуваннях.

На відміну від нікнейму, ім’я користувача не є унікальним — воно виконує переважно комунікативну функцію і може бути довільним. Під ім’ям користувача розташовано три інтерактивні елементи:

- «друзі» — користувачі, з якими встановлено взаємну підписку;
- «підписники» — користувачі, які підписані на акаунт;

- «підписки» — користувачі, на яких підписаний власник профілю.

Ці показники виконують не лише інформативну, а й навігаційну функцію (рис. 2.7): натискання на кожен з них відкриває відповідний перелік учасників платформи, з можливістю переходу до їхніх сторінок та взаємодії (наприклад, підписка або її скасування).

Рисунок 2.7 – Сторінка «Перелік друзів/підписників/підписок»

Важливо також зазначити, що, як і було вказано в описі структури та навігації додатка у підрозділі 2.1, з цього переліку ми можемо лише повернутися на сторінку учасника платформи. Для цього було додано іконку стрілки, яка дає зрозуміти наявність цього функціоналу. Відповідний вказівник розташований у верхньому лівому куті інтерфейсу — традиційному місці для повернення до попереднього екрану, що відповідає загальноприйнятим принципам зручності використання мобільних додатків.

Сама сторінка, яка відкривається після натискання на будь-яку з трьох вкладок — «Підписки», «Друзі» або «Підписники» — має просту та зрозумілу структуру. Угорі знаходиться рядок пошуку, який дозволяє швидко знайти

потрібного учасника в переліку. Нижче розташовується список карток користувачів, кожна з яких містить аватар, ім’я, короткий опис і кнопку взаємодії. У залежності від поточного статусу зв’язку, ця кнопка змінюється на «Підписатись» або «Відписатись». Такий підхід дозволяє забезпечити швидку навігацію та керування соціальними зв’язками без потреби переходити на сторінку кожного окремого профілю.

Важливим аспектом у процесі проєктування інтерфейсу є визначення підходу до представлення акаунтів інших користувачів. У межах запропонованої концепції було прийнято рішення відмовитися від відображення кількісних показників на сторінках інших учасників платформи (рис. 2.8). Зокрема, такі дані, як кількість вподобань, підписок, підписників чи друзів, є доступними виключно на персональній сторінці самого користувача.

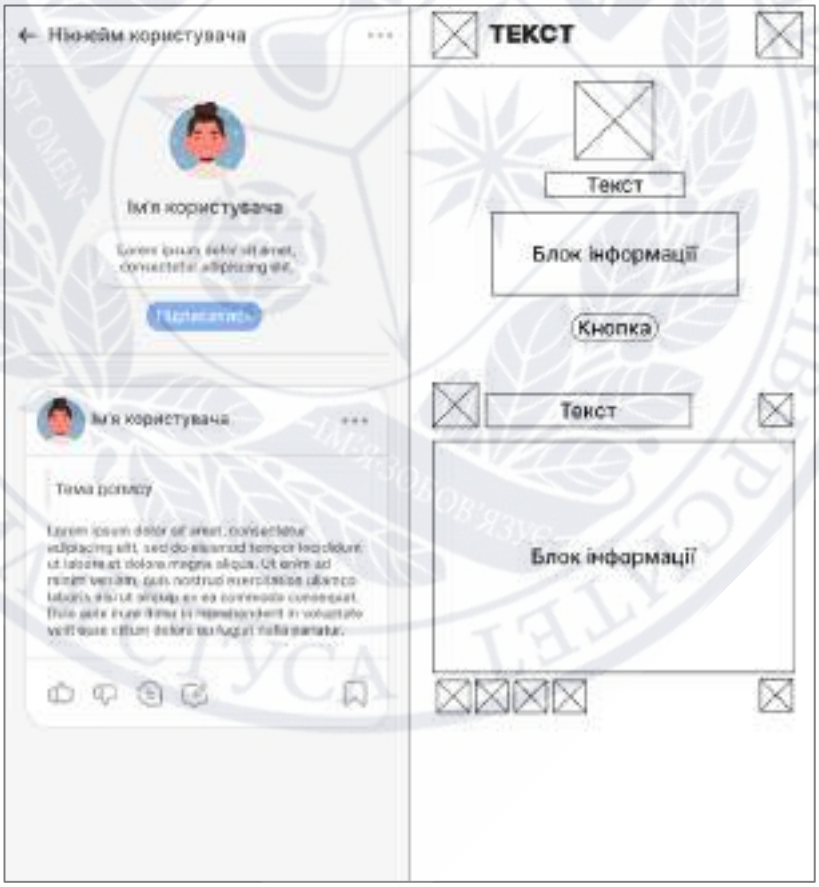

Рисунок 2.8 - Сторінка іншого користувача

Цей дизайнерський вибір ґрунтується на спостереженнях за тенденціями у сфері сучасних соціальних платформ. Аналізуючи зміни в користувацькій поведінці, можна зробити висновок, що все більше людей надають перевагу

платформам, які орієнтовані не на змагальність або публічне порівняння статистичних показників, а на якісне, невимушене спілкування. Зростає запит на простір, вільний від тиску цифрових метрик, таких як кількість вподобань чи охоплень, що часто стають джерелом соціального стресу або почуття меншовартості.

Таким чином, запропонований підхід сприяє формуванню рівноправного та психологічно комфортного середовища, в якому думка кожного користувача є цінною незалежно від кількості взаємодій з його контентом. Така стратегія підтримує ідею відкритої та доброзичливої спільноти, де головним пріоритетом є зміст спілкування, а не його популярність.

Як уже зазначалося раніше, у межах особистої сторінки користувача передбачено розділ налаштувань, що відіграє важливу роль у забезпеченні персоналізованого користувацького досвіду. Враховуючи потребу в інтуїтивно зрозумілій навігації, було прийнято рішення структурувати функціональні можливості налаштувань за тематичними категоріями (рис. 2.9). Такий підхід дозволяє уникнути інформаційного перевантаження та забезпечує логічну послідовність при взаємодії з інтерфейсом.

Зокрема, усі доступні налаштування було згруповано у чотири основні блоки: «Редагування акаунту», «Налаштування додатка», «Додаткова інформація та підтримка» і «Акаунт». Такий розподіл дає змогу швидко орієнтуватися серед великої кількості функцій, відповідно до їх змістового призначення.

З метою поліпшення візуального сприйняття та навігації до кожного пункту було додано графічний індикатор, що підсилює семантичне навантаження текстових позначень і пришвидшує ідентифікацію відповідної дії. Для візуального оформлення категорій використано вже знайомі елементи інтерфейсу — прямокутну рамку та нейтральний фоновий відтінок, що забезпечує збереження стилістичної цілісності всього застосунку.

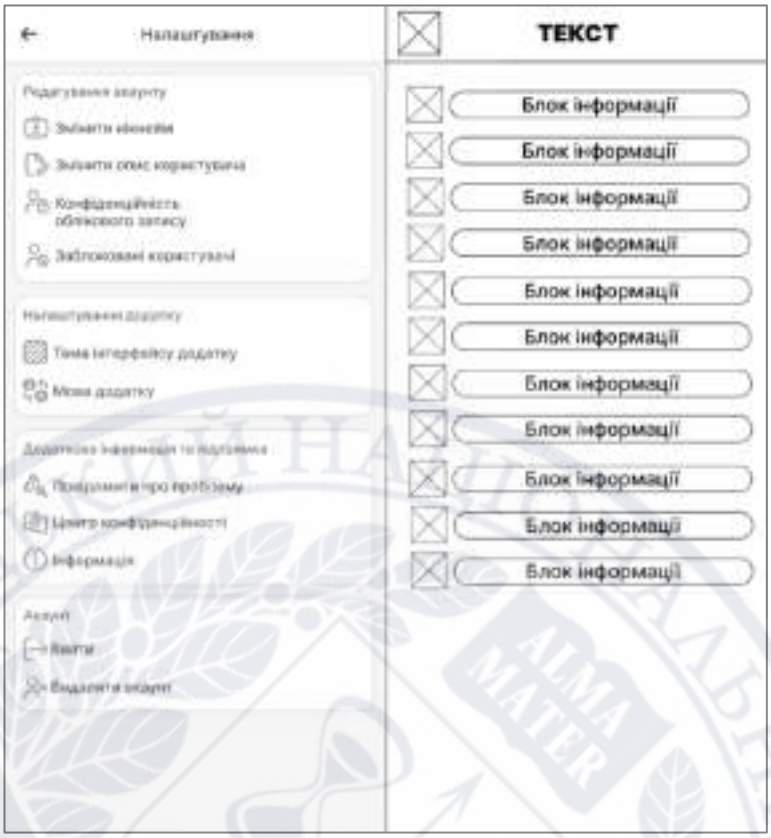

Рисунок 2.9 – Сторінка «Налаштування акаунту»

Кожен із пунктів, представлений у розділі налаштувань, містить у собі додаткові сторінки, що відкриваються у відповідь на взаємодію користувача з відповідним елементом інтерфейсу (рис. А.8 – А.13).

Візуальне та функціональне оформлення цих сторінок повністю відповідає загальній стилістиці застосунку, що сприяє збереженню єдиної дизайн-системи. Це, у свою чергу, забезпечує користувачу інтуїтивно зрозумілу, комфортну та передбачувану взаємодію з інтерфейсом, знижуючи когнітивне навантаження під час виконання типових дій.

У межах користувацького інтерфейсу мобільного застосунку також реалізовано функціональну можливість створення нового допису, яка доступна зі сторінки профілю користувача. Важливим аспектом проєктування цієї сторінки стало забезпечення її візуальної простоти та інтуїтивної зрозумілості з метою уникнення перевантаження інтерфейсу другорядним контентом.

Згідно з логікою взаємодії, першочерговою дією є введення назви теми допису, що відповідає за його тематичну класифікацію (рис. 2.10). Після цього

користувач має можливість заповнити основний текстовий блок, призначений для безпосереднього змісту повідомлення. За потреби додавання медіафайлів, у нижньому лівому куті інтерфейсу передбачено спеціальну кнопку для приєднання файлів. Після завершення всіх етапів заповнення, у нижньому правому куті активується кнопка «Створити», що ініціює публікацію допису.

Рисунок 2.10 – Сторінка зі створення нового допису

Таке розташування елементів та їхня логічна послідовність спрямовані на забезпечення зручності та ефективності взаємодії користувача з інтерфейсом, відповідаючи принципам дизайну орієнтованого на користувачів.

Однією з ключових сторінок мобільного застосунку є сторінка «Сповіщення», на яку було зроблено посилання під час опису інтерфейсу стрічки. Її призначення полягає в інформуванні користувача про зміни у взаємодії з іншими учасниками платформи. Визначено два основних типи сповіщень: повідомлення про нових підписників та про встановлення дружніх зв’язків (рис. 2.11). Для забезпечення чіткої різниці між цими типами передбачено використання візуально диференційованих елементів дизайну, які

при цьому зберігають загальну стилістику застосунку. Основою для кожного повідомлення виступає контейнер з білим фоном і сірим контуром — аналогічний до тих, що використовуються у списках чатів або в переліках друзів, підписок і підписників.

Рисунок 2.11 – Сторінка «Сповіщення»

Кожне сповіщення містить зображення профілю, ім’я користувача та короткий опис події, що забезпечує швидке орієнтування в контексті повідомлення. Крім того, для полегшення візуальної ідентифікації типу звістки реалізовано спеціальні інтерфейсні маркери. Зокрема, у випадку, коли користувач отримує нового підписника, праворуч від повідомлення відображається кнопка «Підписатись», що дозволяє миттєво відповісти на дію. Натомість, якщо йдеться про встановлення дружніх відносин, використовується іконка святкового типу, яка символізує завершення взаємної підписки.

Запропонований підхід до структурування і візуального оформлення сторінки «Сповіщення» спрямований на оптимізацію користувацького досвіду завдяки мінімізації когнітивного навантаження та забезпеченню швидкої інтерпретації повідомлень.

Ще однією з функціонально важливих сторінок мобільного застосунку є інтерфейс відкритого чату. Основною метою цієї сторінки є забезпечення зручного, інтуїтивно зрозумілого процесу комунікації між користувачами. Відповідно до принципів мінімалістичного дизайну, інтерфейс включає лише базові, проте критично необхідні елементи: ім’я співрозмовника у верхній частині екрана, кнопку для переходу до налаштувань чату (рис. А.7), дату поточної розмови, самі повідомлення, поле для введення тексту, кнопку прикріплення медіафайлів та кнопку надсилання (рис. 2.12).

Рисунок 2.12 – Відкрита сторінка розмови

Для ефективного візуального розмежування повідомлень залежно від їхнього відправника реалізовано кольорове кодування. Повідомлення, надіслані користувачем, відображаються на фоні кольору #7EAAED, який уже використовувався раніше в межах застосунку, що сприяє візуальній узгодженості інтерфейсу. Повідомлення співрозмовника мають білий фон із сірим контуром, що забезпечує контраст і чітке сприйняття інформації.

Кожне повідомлення супроводжується позначкою часу його відправлення. Вона розташована у непомітній, але логічно обґрунтованій частині блоку, з використанням кольору, який контрастує з фоном самого повідомлення. Такий підхід сприяє зниженню когнітивного навантаження та забезпечує легке зчитування часової мітки без необхідності зосередженого вивчення інтерфейсу.

У нижній частині екрана знаходиться область для введення тексту. Зліва від неї розташовано прикріплення медіафайлів, а справа — кнопку надсилання повідомлення. Така структура обумовлена прагненням зменшити ймовірність помилкового натискання кнопки надсилання у випадках, коли користувач спершу мав намір додати файл. Просторове розмежування цих елементів є практичним вирішенням потенційної проблеми та сприяє підвищенню точності користувацьких дій.

Загалом, структура відкритого чату орієнтована на функціональність, візуальну простоту, інтуїтивність та відповідність принципам дизайну орієнтованого на користувачів, що забезпечує комфортне використання навіть під час тривалого спілкування.

Більшість сторінок мобільного застосунку доповнені розширеним функціоналом, зокрема можливістю налаштування стрічки контенту, фільтрацією результатів пошуку, параметрами чату та іншими інтерактивними елементами. Окремо розроблено й екрани реєстрації та авторизації, які також оформлено відповідно до загальної стилістики. Усі компоненти інтерфейсу витримано в єдиній візуальній концепції, що забезпечує послідовність дизайну, знижує когнітивне навантаження та сприяє формуванню комфортного користувацького досвіду. Детальні зображення відповідних сторінок наведено в додатках до роботи.

Отже, розроблений дизайн інтерфейсу мобільного застосунку відповідає ключовим функціональним вимогам, сформованим на основі аналізу цільової аудиторії та особливостей платформи. Реалізовано інтуїтивну навігацію, мінімалістичний візуальний стиль і логічну структуру, що сприяє зменшенню

когнітивного навантаження та забезпечує комфортне користування незалежно від рівня цифрової підготовки.

Інтерфейс є адаптивним, доступним для різних вікових груп, та орієнтованим на активну взаємодію користувачів завдяки зручному розміщенню елементів створення контенту. Особливу увагу приділено оптимізації продуктивності: зменшено використання важких анімацій і графіки, що дозволяє забезпечити стабільну роботу застосунку навіть за слабкого інтернет-з'єднання. Такий підхід гарантує якісний користувацький досвід і задовольняє поставлені вимоги до дизайну.

2.4 Реалізація інтерфейсу та функціональних компонентів із використанням обраних технологій

Процес реалізації інтерфейсу мобільного застосунку здійснювався із застосуванням сучасного стеку технологій, зокрема React Native та Expo, що забезпечили ефективну розробку кросплатформеного продукту. Ключовим принципом побудови архітектури стало компонування інтерфейсу за логічними одиницями, що сприяє повторному використанню коду, підвищенню читабельності та спрощенню підтримки проєкту. Для організації навігації використовувалася бібліотека expo-router, яка забезпечує декларативну маршрутизацію на основі файлової структури.

Застосунок побудовано з урахуванням компонентного підходу, притаманного React Native. Кожен функціональний модуль реалізовано у вигляді окремого компонента або екрану, що дозволяє розділити логіку взаємодії з користувачем на незалежні елементи. Для покращення досвіду розробки, налагодження та публікації застосовувався інструментарій Expo, що забезпечив доступ до нативних API, оновлення та швидке розгортання.

Екрани застосунку організовані у вигляді окремих файлів, які відповідають за конкретні функціональні блоки. Основні з них включають:

- відображення та редагування профілю користувача;

- роботу з соціальними зв’язками, зокрема списками друзів, підписників і підписок;
- взаємодію з чатами, включаючи створення, перегляд і структурування діалогів;
- сторінки авторизації та реєстрації;
- перегляд профілю іншого користувача та динамічні маршрути до чатів.

Окрему категорію становлять повторно використовувані компоненти, які інкапсулюють типові елементи інтерфейсу: Post.tsx, UserWithButton.tsx, Chat.tsx, ChatPreview.tsx. Їх застосування мінімізує дублювання коду та сприяє уніфікації візуального оформлення застосунку.

Для реалізації маршрутизації використано бібліотеку expo-router, що базується на файловій структурі. Кожен файл у директорії інтерпретується як окремий маршрут, що дозволяє легко масштабувати проєкт без додаткових конфігурацій. Навігація реалізується за допомогою спеціалізованих React-хуків:

- useRouter — дозволяє керувати переходами між екранами, включаючи переміщення вперед (router.push(...)), назад (router.back()), або заміну поточного маршруту (router.replace(...)) [32];
- useLocalSearchParams — забезпечує доступ до параметрів із динамічних маршрутів, що використовується для відображення персоналізованих сторінок (наприклад, профілю за ідентифікатором користувача).

Застосування цих інструментів дозволяє реалізовувати умовні переходи, будувати універсальні шаблони сторінок та забезпечувати збереження стану при навігації.

Усі мережеві запити та взаємодія з API зосереджені у файлі api.ts, який містить функції для отримання (getRemoteData) і надсилання (postRemoteData) даних. Тут же реалізовано керування авторизаційними токенами, що включає їх зберігання, перевірку дійсності та оновлення (setToken, isTokenValid). Такий підхід дозволяє централізувати мережеву логіку та забезпечує безпечну аутентифікацію користувача.

Для оформлення інтерфейсу використовується стандартна система стилів StyleSheet із React Native. Кожен екран або компонент містить власний набір стилів, ізольований у межах відповідного модуля. Така локалізація стилів спрощує підтримку, полегшує налагодження та зменшує ймовірність конфліктів. Стилі описують не лише зовнішній вигляд, а й поведінкові характеристики елементів, наприклад, реакцію на натискання або зміну стану.

Узагальнюючи вище наведену інформацію, інтеграція React Native, Expo та expo-router забезпечує масштабовану, модульну та зрозумілу архітектуру застосунку. Чітке розділення на екрани та компоненти, централізація API-взаємодії, декларативна навігація та ізольована стилізація є ключовими факторами, що сприяють ефективній реалізації інтерфейсу. Така організація не лише полегшує процес розробки, а й забезпечує високий рівень підтримуваності та можливість подальшого розширення функціональності продукту.

Продовжуючи логічне обґрунтування архітектурних рішень, доцільно розглянути безпосередню реалізацію ключових екранів застосунку, зокрема профілю користувача, сторінок взаємодії, а також чатів. Головна сторінка профілю (profile/index.tsx) реалізована як функціональний компонент із застосуванням хуків стану (useState) для зберігання інформації про користувача та його публікації. Отримання даних здійснюється за допомогою асинхронної функції getRemoteData, що забезпечує оновлення вмісту інтерфейсу при кожному виклику. Візуально сторінка відображає персоналізовані дані користувача, зокрема аватар, ім’я, нікнейм, опис, а також статистику соціальної взаємодії (кількість друзів, підписників і підписок). Для представлення публікацій реалізовано таби, що дозволяють перемикатися між створеними, вподобаними та збереженими постами. Крім цього, передбачено навігаційні кнопки для переходу до налаштувань профілю та створення нової публікації. Вся стилізація виконана за допомогою об’єкта StyleSheet, що забезпечує локальність та ізольованість стилів.

Сторінка перегляду профілю іншого користувача (app/[userId].tsx) має схожу логіку, однак базується на динамічному отриманні параметра userId через

хук useLocalSearchParams. Це дозволяє завантажувати відповідні дані залежно від ідентифікатора профілю. Контент включає в себе аватар, ім’я, нікнейм, опис, а також перелік постів. Навігація реалізована через кнопку повернення назад. Стилі компонента також є локальними та інкапсульованими.

Окремий набір екранів присвячений відображенню соціальних зв’язків користувача. Так, сторінка друзів (profile/(relations)/friends.tsx) завантажує список за допомогою функції getRemoteData, після чого кожен елемент виводиться за допомогою компонента UserWithButton, що містить кнопку для переходу до відповідного профілю. Аналогічним чином реалізовані сторінки підписників (followers.tsx) та підписок (following.tsx).

Функціональність обміну повідомленнями реалізовано через набір екранів, що охоплюють усі аспекти чатів. Головна сторінка чатів (chats/index.tsx) завантажує список розмов за допомогою getRemoteData, після чого дані фільтруються відповідно до локального стану — зокрема передбачено категоризацію на всі, друзі, групові чати та дискусії. Пошук розмов реалізовано через компонент TextInput, який динамічно фільтрує перелік результатів. Для кожної розмови використовується компонент ChatItem, що виконує роль прев’ю. Додатково передбачене контекстне меню з опціями видалення, вимкнення сповіщень або закріплення чату, а також окрема кнопка для створення нового діалогу. Відповідна сторінка створення (chats/create.tsx) виводить список підписників, з якими можливо ініціювати нову розмову, а також поле пошуку для зручного відбору. Перегляд конкретного чату реалізовано на сторінці chats/[chatId]/index.tsx. Вона завантажує історію повідомлень та інформацію про співрозмовника або групу. Повідомлення рендеряться через окремий компонент Chat, а поле введення підтримує багаторядкове введення через TextInput із властивістю multiline. Надсилання реалізовано через функцію postRemoteData, після чого історія чату оновлюється. Окрім цього, наявні додаткові елементи керування: кнопки для додавання вкладень, перегляду інформації про чат та повернення до попереднього екрану.

Загалом усі екрани застосунку реалізовані відповідно до компонентного підходу з використанням локальних стилів, хуків для керування станом (useState) та побічними ефектами (useEffect). Навігаційні переходи реалізовано через функції router.push та router.back, що забезпечує гнучкість і простоту маршрутизації. Отримання даних в усіх випадках виконується асинхронно з використанням API, при цьому інформація зберігається в локальному стані компонента, що сприяє ізольованості логіки та полегшує тестування та відлагодження кожної частини інтерфейсу окремо.

У межах реалізації процесів авторизації та реєстрації у мобільному застосунку застосовано послідовний функціональний підхід із використанням технологій React Native, Expo та бібліотеки expo-router, що забезпечує чітку навігацію та модульність компонентів. Екран входу реалізовано як функціональний компонент, у якому використовуються локальні стани для збереження значень полів імені користувача та пароля. Введення даних здійснюється через елементи керування TextInput, що стилізовані з використанням об’єкта StyleSheet. Після натискання кнопки входу активується функція loginRequest, яка виконує асинхронний запит на сервер за допомогою методу postRemoteData. У разі успішної автентифікації отриманий токен зберігається через функцію setToken, далі здійснюється перевірка його дійсності (isTokenValid) і виконується перенаправлення користувача на головну сторінку за допомогою router.replace. Крім того, на екрані реалізовано можливість переходу до реєстраційної форми та додано опцію для відновлення пароля.

Форма реєстрації реалізована у вигляді покрокового процесу, де кожен етап відповідає за заповнення окремих обов’язкових полів, таких як адреса електронної пошти, пароль, нікнейм та ім’я. Для кожного кроку створено окремий локальний стан, що забезпечує незалежне збереження введених даних. На кожному етапі реалізовано валідацію, зокрема перевірку формату електронної пошти, складності пароля, а також перевірку на унікальність нікнейму. Після завершення усіх етапів зібрані дані надсилаються на сервер через postRemoteData. У разі позитивного результату користувача автоматично

авторизують та перенаправляють до головного екрана. Для переходу між кроками передбачено навігаційні кнопки та індикатор поточної позиції у процесі реєстрації.

Загалом авторизація та реєстрація реалізовані із дотриманням сучасних принципів UX/UI-дизайну: кожен запит до API виконується асинхронно, обробка помилок і валідація виконуються безпосередньо в межах відповідного компонента, а стильове оформлення є уніфікованим завдяки централізованому використанню StyleSheet. Навігаційні дії реалізовано через методи router.replace або router.push, що забезпечує плавність переходів між екранами та загальну узгодженість інтерфейсу.

У контексті компонентного підходу особливе значення мають модулі, що забезпечують повторюване відображення типових елементів інтерфейсу та інтерактивну взаємодію з користувачем. Їх реалізація спрямована на підвищення модульності, забезпечення масштабованості та полегшення супроводу коду.

Компонент Post відповідає за уніфіковане представлення публікацій у стрічці новин. Його структура включає текстовий вміст, медіаелементи (у разі наявності), метаінформацію (автор, дата) та набір елементів взаємодії (реакції, коментарі, збереження). Компонент реалізований як функціональний, з передаванням усіх необхідних даних через пропси. Події, пов’язані з натисканням на окремі елементи, обробляються безпосередньо у межах компонента. Візуальне оформлення забезпечується локальним StyleSheet, що унеможливлює конфлікти зі стилями інших елементів інтерфейсу.

Модуль UserWithButton призначений для відображення профілю користувача у скороченому форматі з додатковим функціональним елементом — кнопкою. Компонент приймає як вхідні параметри ідентифікатор користувача та JSX-елемент кнопки, що забезпечує гнучкість у його адаптації під різні контексти (підписка, перехід, блокування тощо). Крім аватара, імені та нікнейма, може містити допоміжну інформацію, таку як статус відносин або короткий опис. Стилізація виконується ізольовано.

Чат-функціональність реалізована за допомогою двох окремих компонентів — Chat та ChatPreview. Перший відповідає за повноцінне відображення поточного діалогу. Другий компонент використовується в інтерфейсі списку діалогів та надає узагальнену інформацію: аватар співрозмовника, назву чату, останнє повідомлення та індикатор непрочитаних.

Усі перераховані компоненти є прикладом дотримання принципів односпрямованого потоку даних, ізольованої логіки та повторного використання інтерфейсних елементів. Такий підхід дозволяє не лише зменшити обсяг дублікатів, а й забезпечити єдність стилю, прогнозованість поведінки та високу адаптивність до змін вимог або функціоналу застосунку.

Всі операції, пов’язані з обміном даними між клієнтською частиною застосунку та сервером, реалізовано в межах окремого функціонального модуля (utils/api.ts). До його складу входять універсальні методи для здійснення HTTP-запитів: getRemoteData — для отримання інформації, та postRemoteData — для надсилання даних. Дані функції мають асинхронну природу виконання та передбачають обробку потенційних помилок, що виникають у процесі взаємодії з API.

З метою безпечного доступу до ресурсів, що потребують автентифікації, передбачено механізм роботи з токенами. Зокрема, реалізовано функції збереження авторизаційного токена та перевірки його актуальності. Зберігання токенів здійснюється локально, з використанням відповідних механізмів середовища. Такий підхід дозволяє централізувати логіку автентифікації та забезпечити її повторне використання у різних частинах застосунку. Вся мережна взаємодія здійснюється виключно через згаданий модуль, що мінімізує дублювання коду та полегшує подальші модифікації у структурі запитів або правилах авторизації.

Візуальне оформлення застосунку базується на використанні API StyleSheet, притаманного React Native. Стилі визначаються безпосередньо у межах окремих компонентів або екранів, що сприяє ізоляції оформлення та

підвищує його гнучкість. Така структура дозволяє легко підтримувати інтерфейс та локалізувати зміни у візуальному поданні окремих елементів.

З метою забезпечення єдиної стилістики в межах усього застосунку використано уніфіковану кольорову палітру, яка повторюється у різних модулях інтерфейсу (наприклад, відтінки синього, сірого та світлого кольору для кнопок та фону відповідно). Це дозволяє створити цілісне візуальне середовище, що сприяє впізнаваності застосунку.

Інтерфейс побудовано з урахуванням принципів адаптивного дизайну. Використання властивостей flex, внутрішніх відступів (padding) та міжкомпонентного простору (gap) забезпечує коректне масштабування контенту відповідно до розмірів екранів різних пристроїв. Застосунок підтримує світлу тему оформлення, що реалізовано шляхом вибору контрастних кольорів для фону, текстових елементів та елементів керування.

Окрім візуальних аспектів, значну увагу приділено зручності користування. Для реалізації навігації між екранами використано функціональні можливості хуків useRouter, зокрема методи router.push та router.back, що забезпечують інтуїтивну навігацію та плавний перехід між логічними розділами інтерфейсу. Кнопки та інші інтерактивні елементи доповнені зрозумілими іконками й підписами, що підвищує доступність застосунку для користувачів.

У межах цього етапу було реалізовано інтерфейс та функціональні компоненти мобільного застосунку із застосуванням технологій React Native, Expo та expo-router. Компонентна архітектура, декларативна маршрутизація та централізована API-взаємодія забезпечили структурованість, масштабованість і підтримуваність проєкту. Кожен екран реалізовано як ізольований модуль із власними стилями, що сприяє зниженню складності підтримки та мінімізує ризик конфліктів. Повторно використовувані компоненти забезпечують уніфікацію інтерфейсу, а чітке розмежування логіки дозволяє ефективно розвивати функціональність. Таким чином, обрана технологічна база та принципи реалізації інтерфейсу забезпечили високий рівень якості та гнучкості програмного продукту.

РОЗДІЛ 3
ОЦІНКА РЕАЛІЗОВАНОГО ІНТЕРФЕЙСУ ТА ПЕРСПЕКТИВИ ПОДАЛЬШОГО РОЗВИТКУ

3.1 Визначення відповідності розробленого інтерфейсу поставленим вимогам

На підставі попередньо сформульованих функціональних вимог до інтерфейсу мобільного застосунку, проведено аналіз відповідності реалізованого інтерфейсу встановленим критеріям. У процесі розробки особлива увага приділялася дотриманню принципів інтуїтивної навігації, доступності, адаптивності, візуальної простоти та продуктивності, що є ключовими для забезпечення позитивного користувацького досвіду.

Насамперед, реалізовано навігаційні рішення, що відповідають звичним для користувачів мобільних застосунків патернам: використано нижнє меню з іконками швидкого доступу до основних функцій платформи, передбачено використання свайпів та візуально впізнаваних піктограм, що сприяє зниженню когнітивного навантаження. Завдяки цьому забезпечено легке освоєння інтерфейсу навіть для користувачів із базовими цифровими навичками.

Інтерфейс виконано в стилі мінімалізму. Виключено перевантаження декоративними елементами, надлишковими візуальними ефектами та фоновими деталями, що не несуть функціонального навантаження. Візуальний акцент зосереджено на основних елементах взаємодії, що дозволяє користувачу швидко орієнтуватися у структурі застосунку.

Забезпечено адаптивність інтерфейсу шляхом використання контрастних кольорів, логічної ієрархії елементів, чітких підписів і структурованого розташування блоків інформації. Адаптивна верстка забезпечує коректне відображення контенту на різних типах пристроїв, а також враховує вікові особливості користувачів, пропонуючи зручність як для молодої, так і для старшої аудиторії.

Особливу увагу приділено реалізації зручного функціоналу для створення контенту. Створення дописів, додавання коментарів і ведення розмов здійснюються за допомогою мінімальної кількості дій.

Крім того, у проєктуванні інтерфейсу враховано технічні обмеження мобільних пристроїв. Зокрема, реалізовано оптимізацію графіки, обмежено використання ресурсоємних анімацій та ефектів, що дозволило досягти швидкого завантаження екранів, плавного перемикання між розділами та стабільної роботи навіть за умов повільного інтернет-з’єднання.

Також було враховано сучасні принципи UX-дизайну, що передбачають передбачуваність дій користувача, візуальну привабливість, а також підтримку емоційного комфорту. Деякі графічні елементи інтерфейсу мають додаткові текстові описи та забезпечено високий контраст, що відповідає загальним стандартам доступності згідно з міжнародними рекомендаціями.

Таким чином, реалізований інтерфейс повністю відповідає раніше визначеним функціональним вимогам. Врахування кращих практик UX/UI-дизайну, адаптація до потреб різних категорій користувачів, дотримання принципів доступності, зручності та продуктивності забезпечили високий рівень функціональності та ергономіки розробленого застосунку. Результати розробки мають практичну цінність і можуть бути використані як основа для подальших проєктів, спрямованих на створення комфортного цифрового середовища для взаємодії користувачів.

3.2 Аналіз виявлених обмежень і недоліків реалізованого інтерфейсу

У процесі аналізу реалізованого інтерфейсу було виявлено низку функціональних та структурних обмежень, що можуть впливати на зручність користування та подальший розвиток системи.

Одним із суттєвих недоліків є зберігання стану виключно на рівні окремих компонентів, що обмежує ефективність взаємодії між різними частинами застосунку. Відсутність централізованого управління станом ускладнює масштабування та інтеграцію нових функцій.

Також простежується недостатній рівень обробки помилок під час взаємодії з API. Більшість компонентів не надають користувачам чітких повідомлень про помилки, обмежуючись виведенням інформації до консолі або примітивними сповіщеннями.

Інтерфейс реалізовано виключно українською мовою без можливості перемикання на інші мови, що знижує доступність застосунку для ширшої аудиторії.

Функціонал редагування профілю є обмеженим: дозволяється змінювати лише базові параметри, такі як нікнейм, опис чи пароль. Водночас, можливості покращеної безпеки — наприклад, двофакторної аутентифікації — не передбачено.

Крім того, відсутні механізми отримання push-сповіщень, що не дозволяє користувачам оперативно реагувати на нові події, повідомлення чи взаємодії. Також реалізація не підтримує офлайн-режим, що унеможливлює доступ до даних у разі втрати інтернет-з'єднання.

Реалізований інтерфейс забезпечує основну функціональність, зокрема перегляд профілів, обмін повідомленнями та взаємодію з контентом. Структура компонентів є логічною і модульною, що полегшує супровід та подальше розширення системи. Однак для підвищення якості користувацького досвіду та конкурентоспроможності застосунку доцільно удосконалити адаптивність, розширити функції локалізації та безпеки профілю, покращити механізми обробки помилок, а також реалізувати підтримку офлайн-режиму.

3.3 Перспективи розвитку та удосконалення інтерфейсної складової додатка

У межах подальшого розвитку продукту можливим є впровадження низки функціональних та інтерфейсних удосконалень, спрямованих на поглиблення інтерактивності, персоналізації досвіду користувача та посилення залученості аудиторії. Разом із цим важливим аспектом покращення є виявлення та усунення наявних обмежень і недоліків, що впливають на ефективність використання

системи. Запропоновані ініціативи покликані не лише вдосконалити зручність використання додатка, а й сформувати унікальну й впізнавану візуально-комунікативну ідентичність продукту.

Одним із потенційних напрямів удосконалення є реалізація можливості тематичних дискусій під дописами або в окремих групових чатах. Такий функціонал дозволить структурувати комунікацію користувачів за конкретними темами, спрощуючи навігацію в інформаційному просторі платформи. Додатково варто передбачити можливість збереження окремих обговорень до вкладки чатів. Це сприятиме підвищенню зручності комунікації та ефективному управлінню взаємодією користувача з контентом.

Для формування емоційного зв’язку з продуктом перспективною є ідея впровадження персонажів, які виконуватимуть роль віртуальних провідників або маскотів додатка. Чудовим прикладом такого підходу є застосунок Duolingo [33]. Цей сервіс активно використовує персонажа-маскота — зелену сову на ім’я Duo, яка стала впізнаваним символом бренду. Вона не лише візуально супроводжує користувача, але й виконує функції мотиваційного гіда: заохочує до проходження уроків, нагадує про необхідність повернення до навчання та святкує досягнення, що сприяє формуванню емоційного зв’язку з користувачем [34]. На етапі реєстрації вже реалізовано візуальне представлення трьох стилізованих фігур (рожевий трикутник, блакитний квадрат і жовте коло), які можуть бути інтегровані в основну взаємодію як інтерфейсні персонажі-наставники. У контексті даного застосунку вони можуть бути задіяні, зокрема, у форматі навчальних гідів при першому вході в систему, а також у контексті інформативних підказок, повідомлень або святкувань досягнень користувача, наприклад, 10 перших друзів чи дописів.

Окремим напрямом удосконалення є впровадження конструктора персонажів-аватарів, який надасть користувачам можливість формувати персоналізоване візуальне представлення у системі. Така опція, відповідно до принципів персоналізації в UI/UX-дизайні, сприяє формуванню емоційного

зв’язку між користувачем та цифровим середовищем, що, у свою чергу, позитивно впливає на загальний користувацький досвід [35].

Ще одним варіантом удосконалення є візуалізація профілю через аватара, що дозволятиме реалізувати механізм самопрезентації, який є особливо важливим у соціально орієнтованих застосунках. Дослідження свідчать, що персоналізація інтерфейсу може збільшувати тривалість взаємодії з додатком, а також підвищувати рівень задоволення у користувачів [36].

Рекомендується реалізувати базовий набір параметрів для налаштування персонажа, серед яких — форма тіла, відтінок шкіри, колір волосся, елементи одягу та аксесуари. Надалі ці опції можуть бути розширені, зокрема за допомогою, наприклад, системи досягнень, яка відкриватиме нові стилістичні елементи за активність користувача в додатку. Така гейміфікація сприятиме підвищенню мотивації до використання застосунку [37].

Ще одним перспективним напрямом розвитку інтерфейсу є впровадження функціоналу «хмари інтересів» — інтерактивної візуалізації, яка відображає сформований алгоритмом профіль користувача на основі його взаємодії з контентом. Такий підхід дозволяє учасникам застосунку не лише переглядати, але й активно коригувати власний інформаційний простір, видаляючи неактуальні теми або додаючи нові сфери інтересів.

Концепція «хмари інтересів» ґрунтується на принципах персоналізації та прозорості алгоритмічних рішень. Згідно з дослідженнями компанії Vidora, візуалізація інтересів у вигляді графу сприяє кращому розумінню користувачем логіки рекомендаційної системи, що, у свою чергу, підвищує довіру до платформи та задоволення від її використання [38].

Таким чином, запропоновані напрями розвитку інтерфейсної складової поєднують як функціональні, так і емоційно-мотиваційні аспекти взаємодії користувача з додатком. Їх впровадження дозволить створити більш гнучке, інклюзивне й залучене середовище для користувачів, а також підвищити загальну конкурентоспроможність продукту на ринку мобільних застосунків.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано повний цикл проєктування інтерфейсу мобільного застосунку, орієнтованого на соціальну взаємодію користувачів на основі спільних інтересів. Систематизовано сучасні вимоги до UI/UX-дизайну в контексті мобільних платформ, проаналізовано провідні практики побудови цифрового середовища для комунікації, а також виявлено недоліки існуючих рішень, що зумовлюють потребу в нових підходах до проєктування інтерфейсів.

Встановлено, що надмірне інформаційне навантаження, слабка персоналізація та складність навігації залишаються поширеними проблемами у подібних застосунках. У відповідь на це, в роботі підготовлено концепцію інтерфейсу, що базується на принципах мінімалізму, адаптивності та інтуїтивності. Запропонована навігаційна структура забезпечує логічну послідовність користування, підтримує ефективну взаємодію з основними функціями та мінімізує кількість когнітивних бар’єрів. Особливу увагу приділено відокремленню окремих дій на ізольовані екрани, що підвищує концентрацію користувача та спрощує виконання завдань.

Уточнено підходи до побудови інтерфейсу з урахуванням особливостей цільової аудиторії — інтерфейс є доступним для користувачів з різним рівнем цифрової компетентності, підтримує легку взаємодію з контентом, а також враховує вимоги до продуктивності. Зменшення графічного навантаження та відмова від складних анімацій дозволили адаптувати застосунок до умов нестабільного інтернет-з’єднання, що є актуальним для широкого кола користувачів.

Реалізацію інтерфейсу здійснено із використанням сучасних інструментів: React Native, Expo та expo-router. Компонентна архітектура, повторне використання елементів, чітка маршрутизація та централізована взаємодія з API сприяли підвищенню масштабованості, підтримуваності та гнучкості системи.

Реалізований продукт демонструє відповідність технічним та функціональним вимогам, що були сформульовані на початковому етапі дослідження.

Підготовлений інтерфейс не лише задовольняє вимоги до зручності, ефективності та естетики, а й відкриває можливості для подальшого розвитку. Зокрема, доцільним є впровадження функцій локалізації, покращення безпеки профілю користувача, розширення адаптивності під різні пристрої та оптимізація механізмів обробки помилок. Надані рекомендації дозволяють послідовно розширювати функціональність системи відповідно до зростаючих потреб аудиторії.

Таким чином, сформульовані в роботі положення мають як теоретичну, так і практичну цінність. Здобуті результати можуть бути використані в майбутніх дослідженнях та проєктах, спрямованих на вдосконалення цифрового користувацького досвіду, створення інтерфейсу для соціальної взаємодії, а також на розробку мобільних сервісів, здатних підтримувати сталу комунікацію на основі спільних інтересів.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Best Practices for Mobile App UI Design. URL:https://toxigon.com/best-practices-for-mobile-app-ui-design (дата звернення: 16.04.2025).
2. Mobile UX Design Principles Toptal. URL: https://www.toptal.com/designers/mobile-ui/mobile-ux-design-principles (дата звернення: 16.04.2025).
3. Mobile Accessibility at W3C. URL: https://www.w3.org/WAI/standards-guidelines/mobile/ (дата звернення: 17.04.2025).
4. User Centered Design (UCD). URL: https://www.interaction-design.org/literature/topics/user-centered-design (дата звернення: 17.04.2025).
5. What is Figma. URL: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma (дата звернення: 18.04.2025).
6. Visual Studio Code Documentation. URL: https://code.visualstudio.com/docs (дата звернення: 18.04.2025).
7. Get Started with React Native. URL: https://reactnative.dev/docs/environment-setup (дата звернення: 19.04.2025).
8. Introduction to Expo Router. URL: https://docs.expo.dev/router/introduction/ (дата звернення: 19.04.2025).
9. TypeScript is JavaScript with syntax for types. URL: https://www.typescriptlang.org (дата звернення: 19.04.2025).
10. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 21.04.2025).
11. What is Babel? Babel. URL: https://babeljs.io/docs/ (дата звернення: 21.04.2025).
12. 10 Usability Heuristics for User Interface Design. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 22.04.2025).
13. Garrett J.J. The Elements of User Experience: User-Centered Design for the Web and Beyond. 2011. 172.

14. Tidwell J., Brewer C., Valencia A. Designing Interfaces: Patterns for Effective Interaction Design. 2020. 600 с.

15. Пустюльга С.І., Самчук В.П., Шаповалова А.А., Клак Ю.В. Аналіз тенденцій та трендів у дизайні інтерфейсів сучасних мобільних додатків. Луцьк, 2024, С. 135–148.

16. Кравченко С.М. Розроблення UX/UI дизайну мобільних додатків. Житомир, 2022 р.

17. Гуменюк К.В., Січко Т.В. Передові методи анімації вебінтерфейсів: вплив на користувацький досвід. Матеріали V Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених (24 травня 2024 року, м. Вінниця). Вінниця, ДонНУ імені Василя Стуса, 2024. С. 13-15.

18. Is Dark Mode Better for Battery Life? Facts, Myths, and Studies Explained. URL: https://poweringautos.com/is-dark-mode-better-for-battery-life/ (дата звернення: 26.04.2025).

19. The Future of Voice User Interfaces and UX Design. URL: https://www.uxmatters.com/mt/archives/2024/10/the-future-of-voice-user-interfaces-and-ux-design.php/ (дата звернення: 27.04.2025).

20. Incorporating Augmented Reality (AR) into Mobile Apps to Boost Interactivity. URL: https://nandbox.com/ar-in-mobile-apps-user-engagement/ (дата звернення: 27.04.2025).

21. How Accessibility Audits Are Shaping the Future of User-Centered Design. URL: https://www.uxmatters.com/mt/archives/2025/04/how-accessibility-audits-are-shaping-the-future-of-user-centered-design.php (дата звернення: 28.04.2025).

22. Design. Color. URL: https://developer.apple.com/design/human-interface-guidelines/color (дата звернення: 28.04.2025).

23. How Color Psychology Influences Consumer Behavior? URL: https://floowitalent.com/how-color-psychology-influences-consumer-behavior/ (дата звернення: 28.04.2025).

24. Вплив кольору на настрій – психологічний аналіз кольорової палітри. URL: https://fact-news.com.ua/vpliv-koloru-na-nastrij-psixologichnij-analiz-kolorovoi-palitri/ (дата звернення: 28.04.2025).

25. Веретільник Т.І., Мисник Л.Д., Капітан Р.Б., Мамонов Ю.П., Манзюра О.В. Основи теорії кольору. Черкаси, 2020. 130 с.

26. Іттен Й. Мистецтво кольору: Суб'єктивний досвід і об'єктивне пізнання як шлях до мистецтва. Київ, 2022. 96 с.

27. Basic Patterns for Mobile Navigation: A Primer. URL: https://www.nngroup.com/articles/mobile-navigation-patterns/ (дата звернення: 29.04.2025).

28. How People Read Online: New and Old Findings. URL: https://www.nngroup.com/articles/how-people-read-online/ (дата звернення: 29.04.2025).

29. Синєпупова Н. Композиція: Тотальний контроль. Київ, 2019. 240 с.

30. Skeuomorphic or flat? The effects of icon style on visual search and recognition performance. URL: https://www.sciencedirect.com/science/article/abs/pii/S014193822400177X (дата звернення: 09.05.2025).

31. A Comparative Study of Skeuomorphic and Flat Design from a UX Perspective. URL: https://www.mdpi.com/2414-4088/2/2/31 (дата звернення: 09.05.2025).

32. Next.js. UseRouter. URL: https://nextjs.org/docs/app/api-reference/functions/use-router (дата звернення: 10.05.2025)

33. Топ 10 прикладів гейміфікації (перетворення у гру) в освіті, які змінять наше майбутнє. URL: https://osvitanova.com.ua/posts/1143-top-10-prykladiv-heimifikatsii-peretvorennia-u-hru-v-osviti-iaki- (дата звернення: 10.05.2025)

34. Why Is Duolingo Icon Old: Original Design & Evolution Story. URL: https://duolingoguides.com/why-is-duolingo-icon-old/ (дата звернення: 11.05.2025)

35. 7 Fascinating Facts You Need to Know About Your Reddit Avatar! URL: https://locall.host/what-is-a-reddit-avatar/ (дата звернення: 11.05.2025)

36. Norman D.A. The Design of Everyday Things. MIT Press. New York, 2013. 369 с.

37. Deterding S., Dixon D., Khaled R., Nacke L. From game design elements to gamefulness: Defining «gamification». 2011. С. 9-15.

38. Understanding User Interest Graphs for Personalization. URL: https://www.vidora.com/product-updates/user-interest-graphs-personalization/ (дата звернення: 13.05.2025)

ДОДАТКИ

ДОДАТОК А

Дизайн сторінок інтерфейсу додатка

Рисунок А.1 – Панель фільтрів дописів на сторінці «Стрічка»

Рисунок А.2 – Взаємодія із дописом «Три крапки»

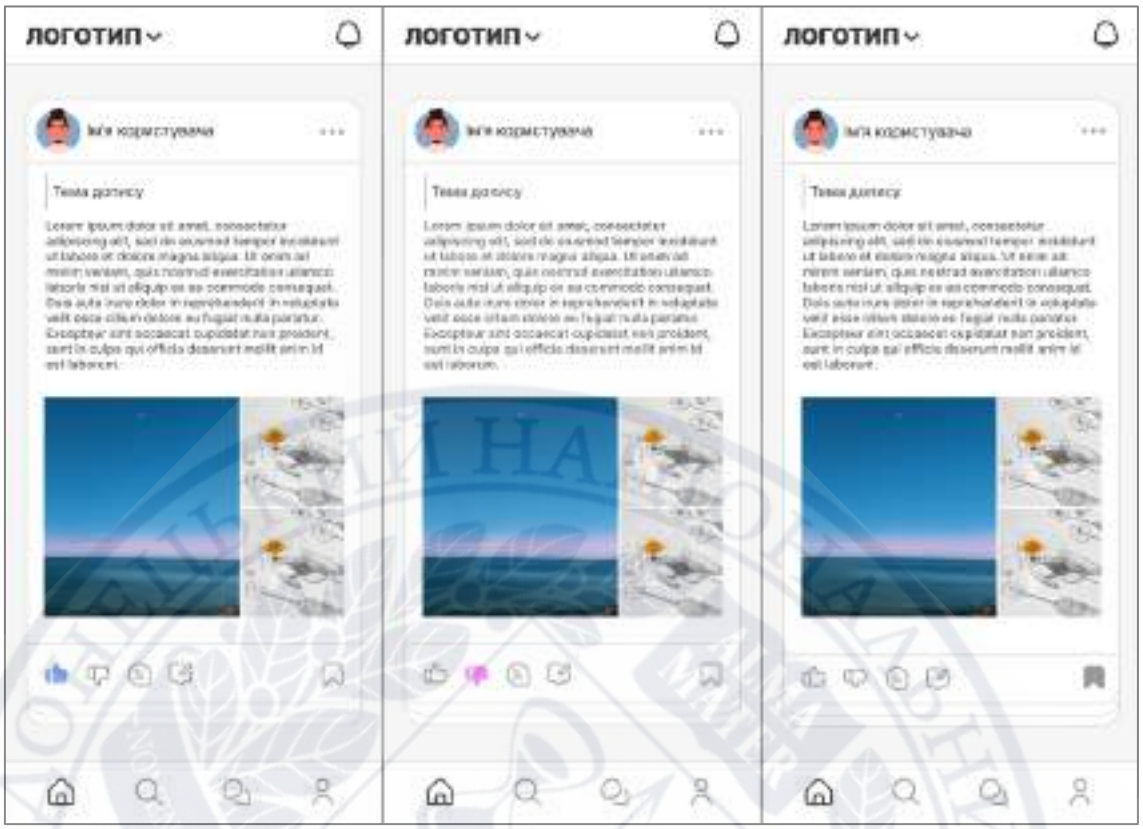

Рисунок А.3 – Взаємодія із дописом «Вподобайка/Дизлайк/Збереження»

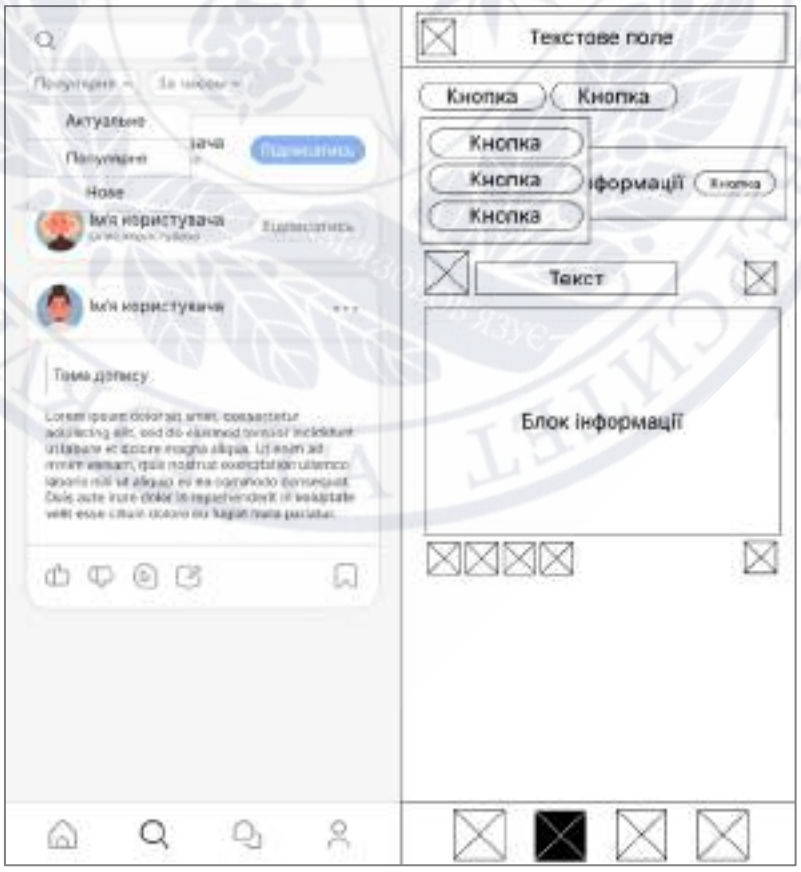

Рисунок А.4 – Фільтрація результатів за актуальністю

Рисунок А.5 – Фільтрація результатів за часом

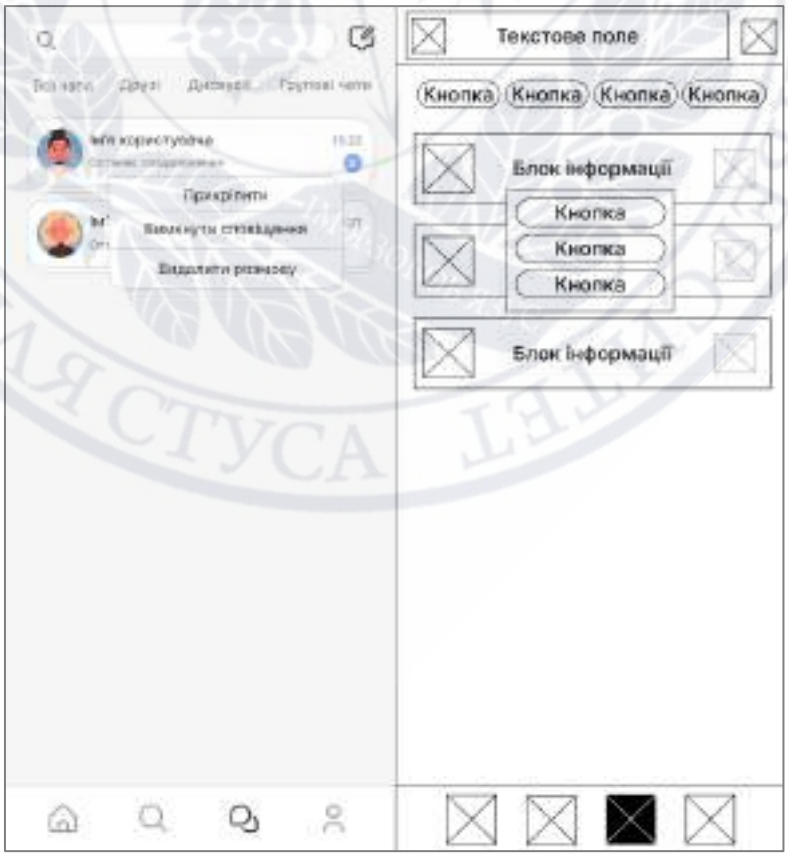

Рисунок А.6 – Редагування чату на головній сторінці «Чати»

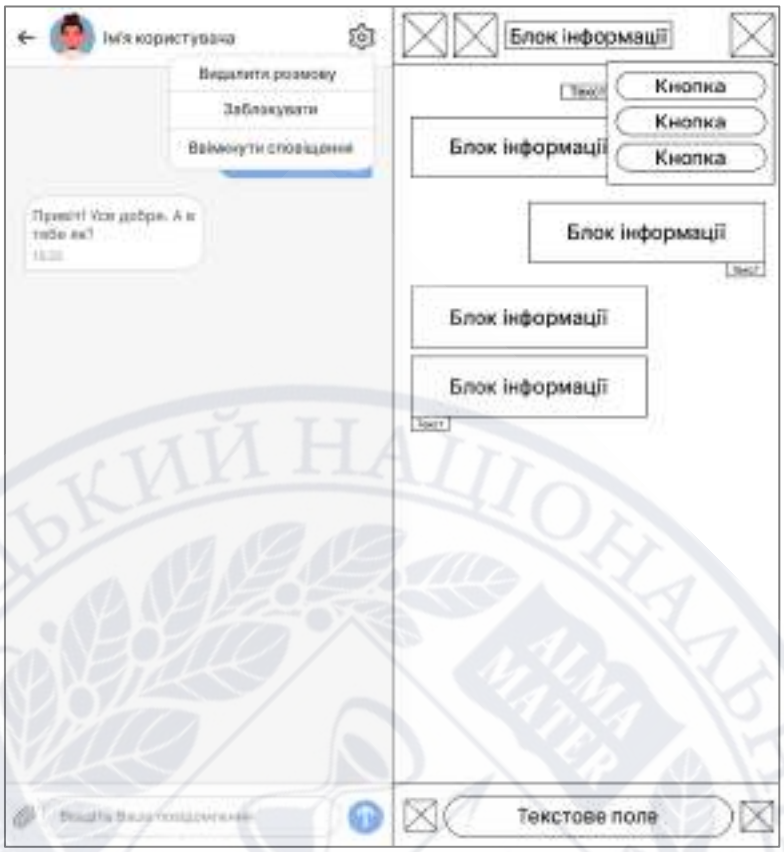

Рисунок А.7 – Редагування чату через його налаштування

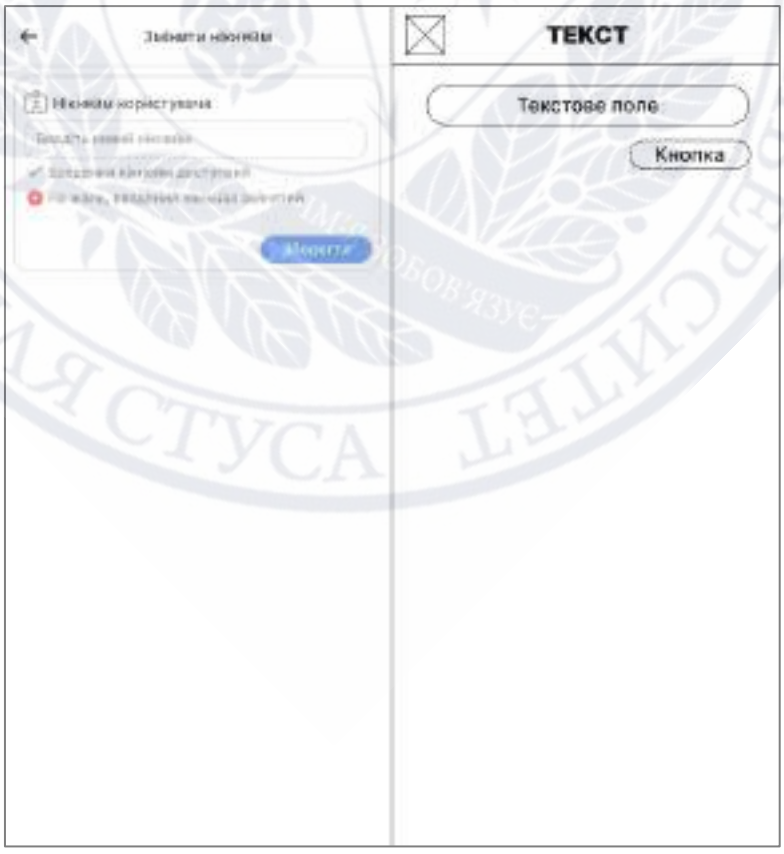

Рисунок А.8 – Налаштування профілю «Змінити нікнейм»

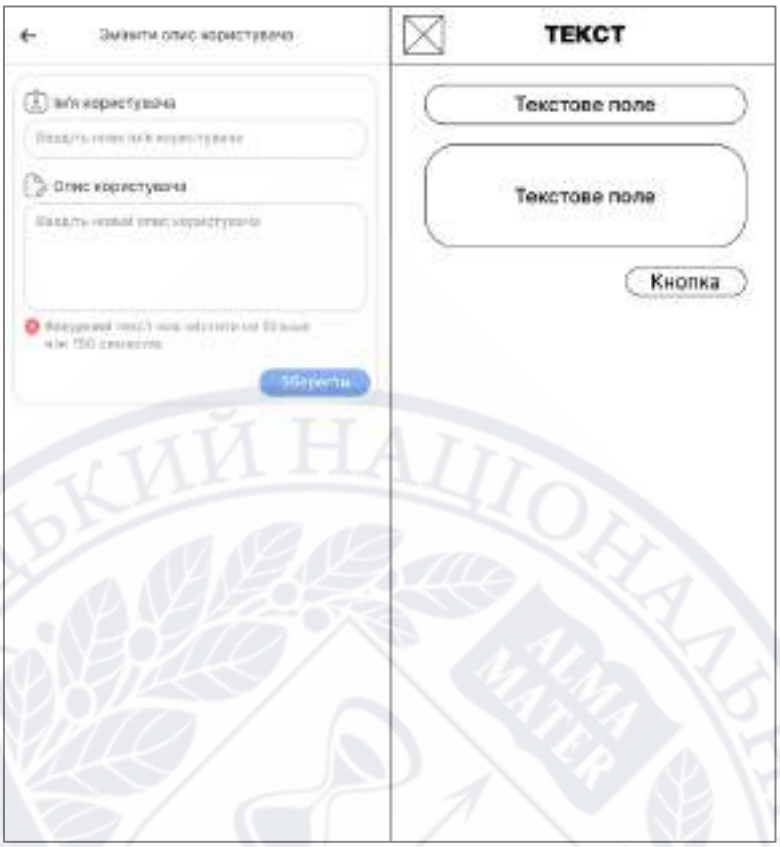

Рисунок А.9 – Налаштування профілю «Змінити опис користувача»

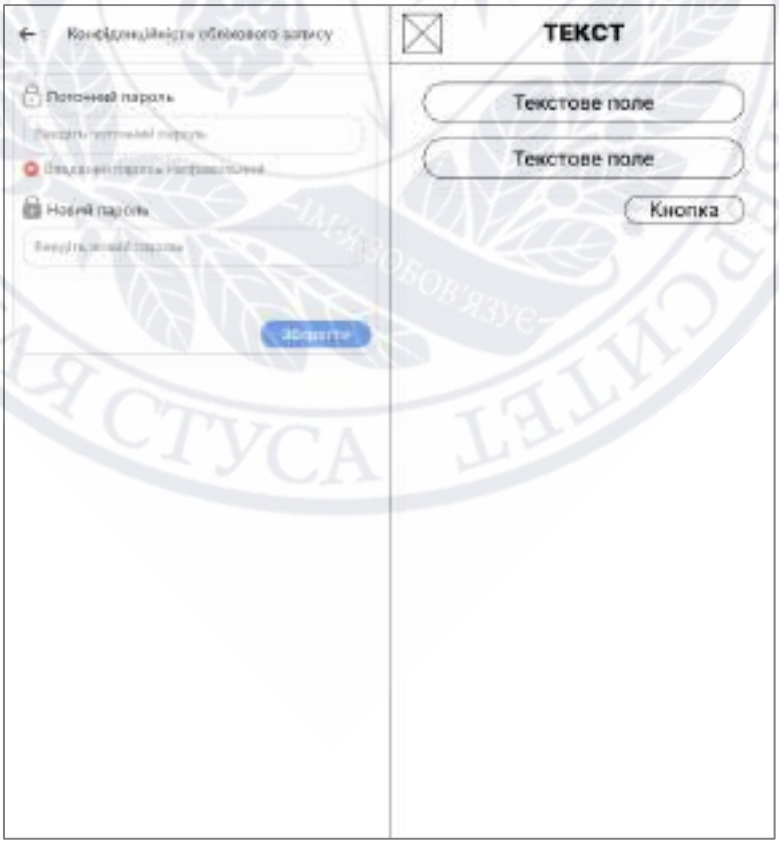

Рисунок А.10 – Налаштування профілю «Конфіденційність облікового запису»

Рисунок А.11 – Налаштування профілю «Заблоковані користувачі»

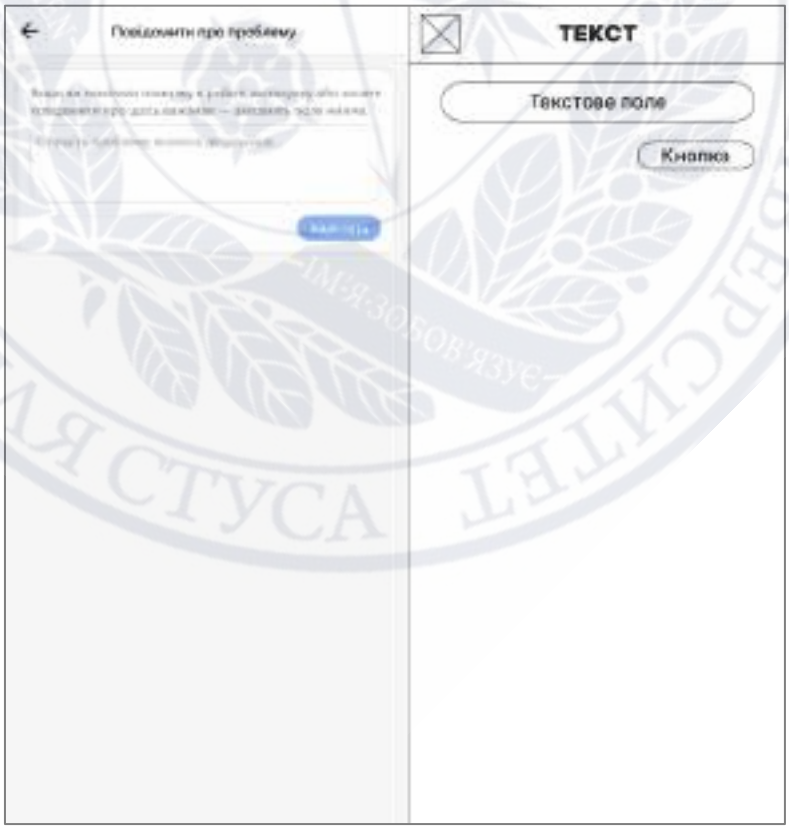

Рисунок А.12 – Налаштування профілю «Повідомити про проблему»

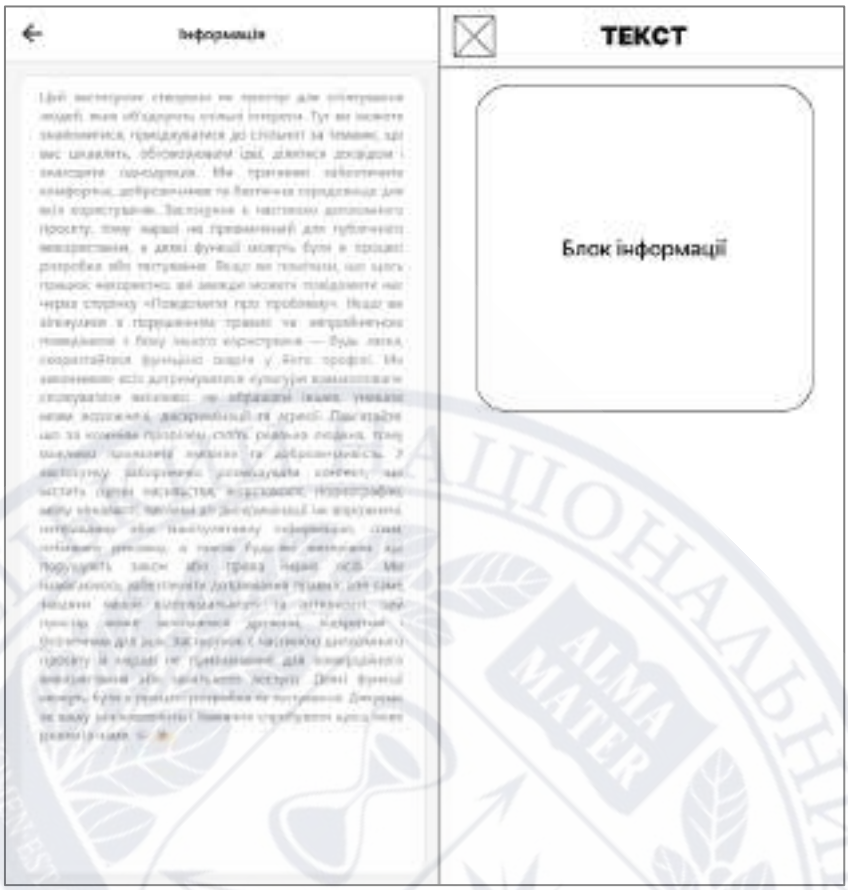

Рисунок А.13 – Налаштування профілю «Інформація»

ДОДАТОК Б

Реалізований інтерфейс

Рисунок Б.1 – Вхід в особистий акаунт

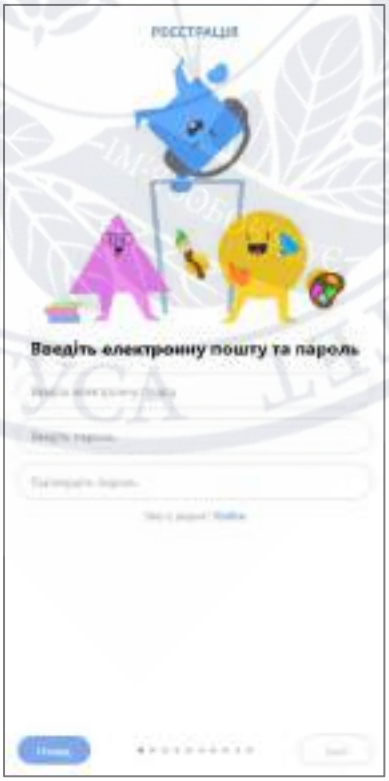

Рисунок Б.2 – Введення даних при реєстрації: логін та пароль

Рисунок Б.3 – Введення нікнейму

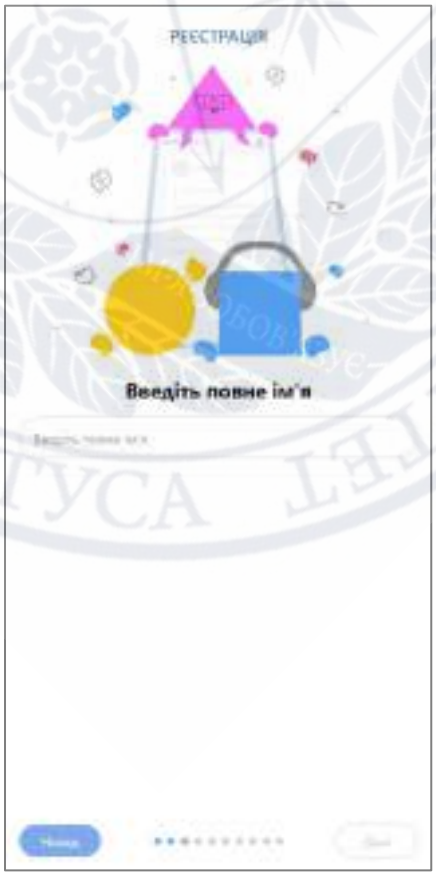

Рисунок Б.4 – Введення повного імені

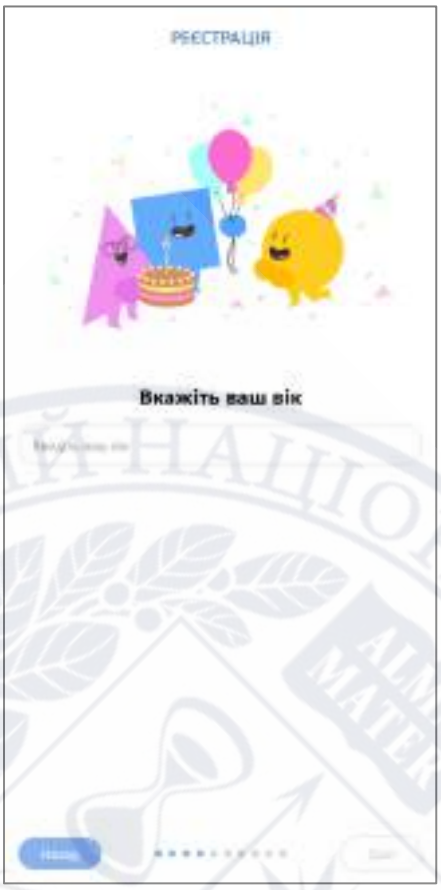

Рисунок Б.5 – Введення віку

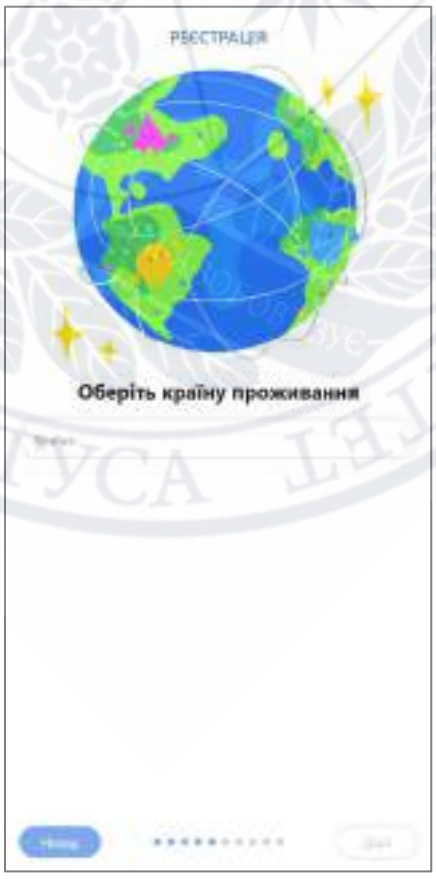

Рисунок Б.6 – Вибір країни проживання

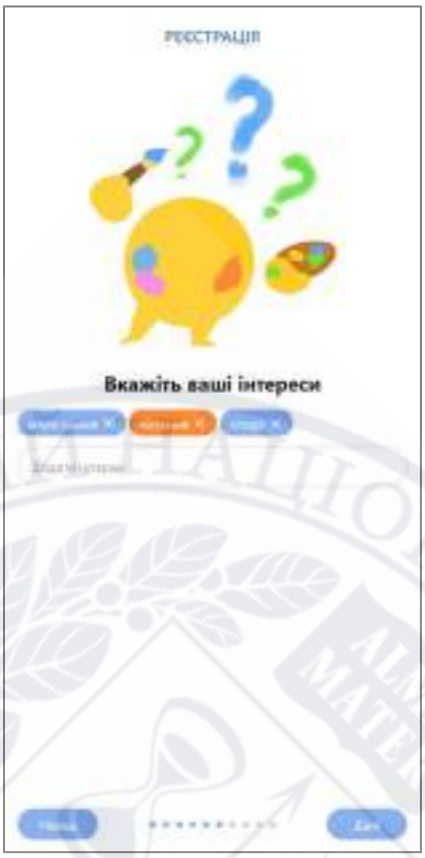

Рисунок Б.7 – Вибір інтересів

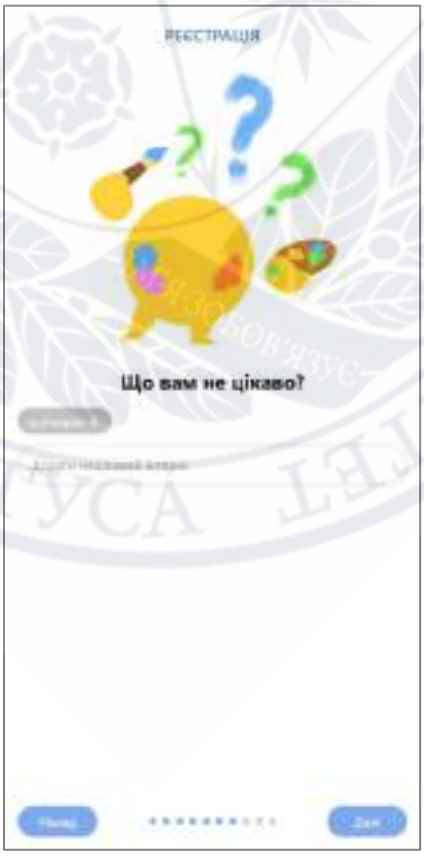

Рисунок Б.8 – Вибір тем, що не цікавлять

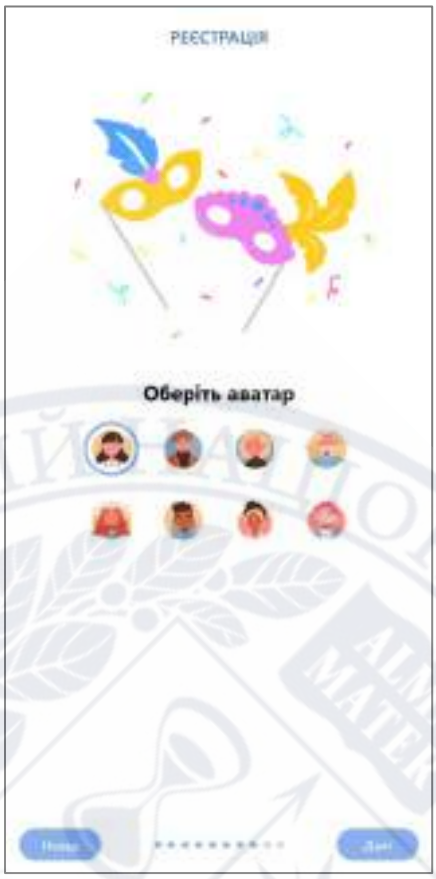

Рисунок Б.9 – Вибір зображення профілю

Рисунок Б.10 – Ознайомлення із правилами

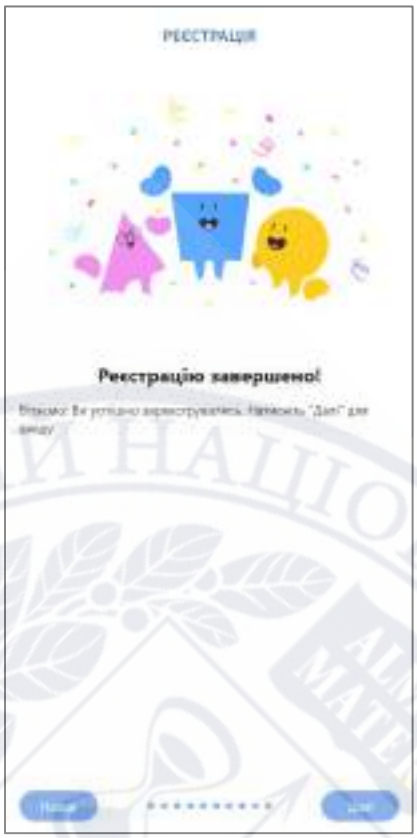

Рисунок Б.11 – Повідомлення про успішну реєстрацію

Рисунок Б.12 – Сторінка «Стрічка»

Рисунок Б.13 – Сторінка «Пошук»

Рисунок Б.14 – Сторінка «Чати»

Рисунок Б.15 – Сторінка «Особистий профіль користувача»

Рисунок Б.16 – Сторінка іншого користувача

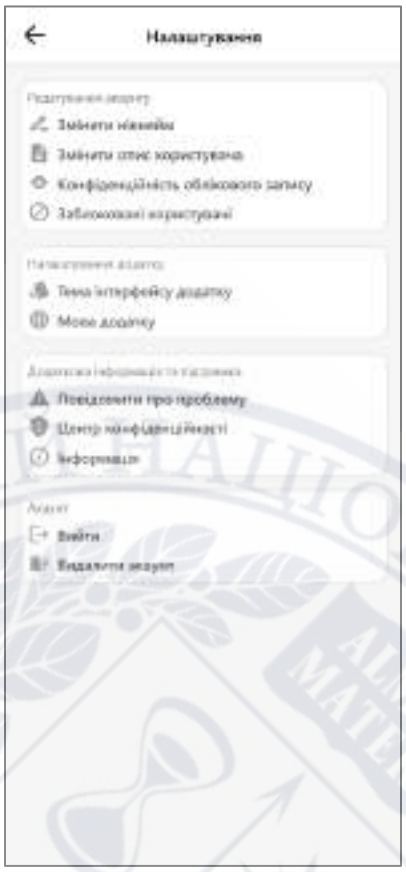

Рисунок Б.17 – Налаштування

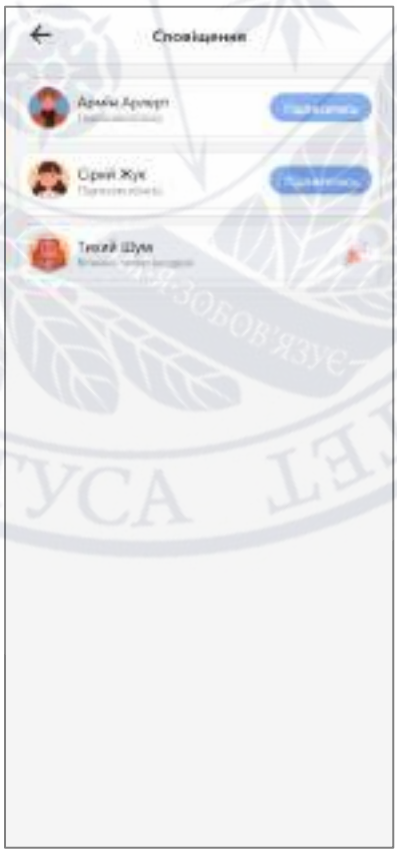

Рисунок Б.18 – Сповіщення

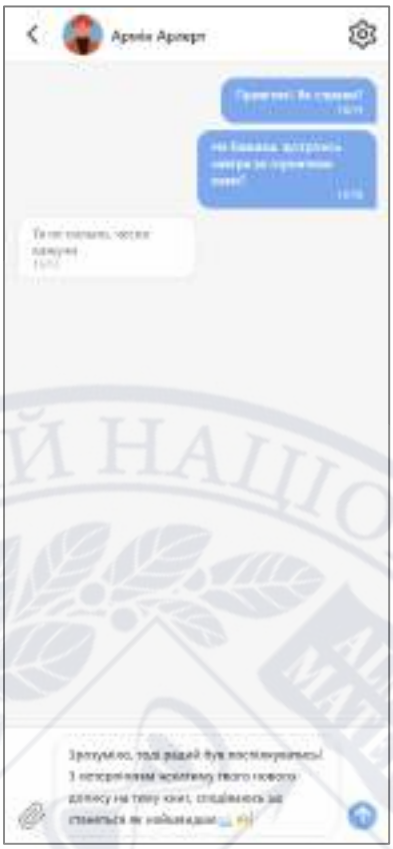

Рисунок Б.19 – Листування з іншим користувачем

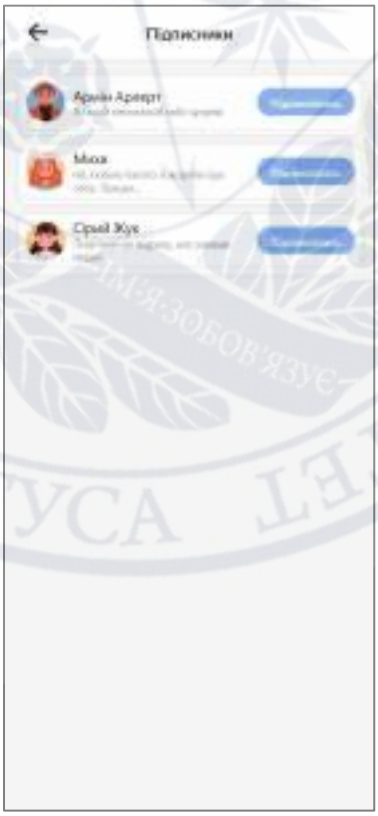

Рисунок Б.20 – Список Друзів/Підписників/Підписок

Рисунок Б.21 – Створення нового допису

ДОДАТОК В

Лістинг коду

Post.tsx:

```
import { Octicons } from "@expo/vector-icons";
import React, { useState, useEffect, useContext } from "react";
import {
  View,
  Text,
  StyleSheet,
  TouchableOpacity,
  Image,
  ScrollView,
} from "react-native";
import { PostData } from "@/constants/types";
import getUserAvatar from "@/constants/user";
import { router } from "expo-router";
import { ContextMenuContext } from "@/contexts/ContextMenuContext";
import { deleteRemoteData, getRemoteData, postRemoteData } from
"@/utils/api";
import { PostSkeleton } from "./PostSkeleton";
import { getData } from "@/utils/storage";

interface PostProps {
  postId: number;
  fixedSize?: boolean;
  onClose?: () => void;
  onDelete?: () => void;
}

function AttachmentsGrid({ attachments }: { attachments: string[]
}) {
  if (!attachments || attachments.length === 0) return null;
  const count = attachments.length;
  if (count === 1) {
    return (
      <View style={styles.attachmentsGridRow}>
        <Image
          source={{ uri: attachments[0] }}
          style={styles.attachmentSingle}
          resizeMode="cover"
        />
      </View>
    );
  }
  if (count === 2) {
    return (
      <View style={styles.attachmentsGridRow}>
        <Image
```

```
          source={{ uri: attachments[0] }}
          style={styles.attachmentHalf}
          resizeMode="cover"
        />
        <Image
          source={{ uri: attachments[1] }}
          style={styles.attachmentHalf}
          resizeMode="cover"
        />
      </View>
    );
  }
  if (count === 3) {
    return (
      <View style={styles.attachmentsGridRow}>
        <Image
          source={{ uri: attachments[0] }}
          style={styles.attachmentHalf}
          resizeMode="cover"
        />
        <View
          style={{
            width: 0,
            flexGrow: 1,
            flexShrink: 1,
            flexDirection: "column",
            gap: 4,
          }}
        >
          <Image
            source={{ uri: attachments[1] }}
            style={styles.attachmentQuarter}
            resizeMode="cover"
          />
          <Image
            source={{ uri: attachments[2] }}
            style={styles.attachmentQuarter}
            resizeMode="cover"
          />
        </View>
      </View>
    );
  }
  if (count >= 4) {
    const showCount = Math.min(4, count);
    return (
      <View style={styles.attachmentsGrid}>
        <View style={styles.attachmentsGridRow}>
          <Image
            source={{ uri: attachments[0] }}
            style={styles.attachmentQuarter}
            resizeMode="cover"
```

```
          />
          <Image
            source={{ uri: attachments[1] }}
            style={styles.attachmentQuarter}
            resizeMode="cover"
          />
        </View>
        <View style={styles.attachmentsGridRow}>
          <Image
            source={{ uri: attachments[2] }}
            style={styles.attachmentQuarter}
            resizeMode="cover"
          />
          {showCount > 4 ? (
            <View style={styles.attachmentQuarter}>
              <Image
                source={{ uri: attachments[3] }}
                style={[
                  styles.attachmentQuarter,
                  { position: "absolute", top: 0, left: 0 },
                ]}
                resizeMode="cover"
              />
              <View style={styles.overlay}>
               <Text style={styles.overlayText}>+{count - 4}</Text>
              </View>
            </View>
          ) : (
            <Image
              source={{ uri: attachments[3] }}
              style={styles.attachmentQuarter}
              resizeMode="cover"
            />
          )}
        </View>
      </View>
    );
  }
  return null;
}

export const Post: React.FC<PostProps> = ({ postId, fixedSize,
onDelete }) => {
  const { showMenu } = useContext(ContextMenuContext);
  const [post, setPost] = useState<PostData | null>(null);
  const [interactions, setInteractions] = useState<string[]>([]);
  const [initialized, setInitialized] = useState(false);
  const [isMine, setIsMine] = React.useState(false);
  const [isLoading, setIsLoading] = useState(true);

  const foreignPostMenuOptions = [
    {
```

```
    label: "Поскаржитись",
    onPress: () => alert("Поскаржитись"),
  },
  {
    label: "Чому я це бачу?",
    onPress: () => alert("Чому я це бачу?"),
  },
  {
    label: "Нецікаво",
    onPress: () => alert("Нецікаво"),
  },
];

const myPostMenuOptions = [
  {
    label: "Редагувати",
    onPress: () => alert("Редагувати пост"),
  },
  {
    label: "Видалити",
    onPress: async () => {
      await deleteRemoteData(`/posts/${postId}`);
      console.log("Post deleted");
      onDelete?.();
    },
  },
];

useEffect(() => {
  if (!postId) return;
  setIsLoading(true);
  getRemoteData<PostData>(`/posts/${postId}`)
    .then((data) => {
      setPost(data);
      setInteractions(data.interactions || []);
      setInitialized(true);
      getData<string>("token").then((token) => {
        if (!token) {
          setIsMine(false);
          return;
        }
        const payload = JSON.parse(atob(token.split(".")[1]));
        const userId = payload.nameid;
        setIsMine(data.creator?.id == userId);
      });
    })
    .finally(() => {
      setIsLoading(false);
    });
}, [postId]);

useEffect(() => {
```

```
    if (!initialized) return;
    if (interactions === (post?.interactions || [])) return;
    postRemoteData(`/posts/${postId}/interactions`, interactions);
  }, [interactions, initialized, post]);

  function handleSaved() {
    setInteractions((prev) =>
      prev.includes("save")
        ? prev.filter((i) => i !== "save")
        : [...prev, "save"]
    );
  }

  function handleLike() {
    setInteractions((prev) => {
      const newInteractions = prev.includes("like")
        ? prev.filter((i) => i !== "like")
        : [...prev, "like"];
      if (newInteractions.includes("dislike")) {
        return newInteractions.filter((i) => i !== "dislike");
      }
      return newInteractions;
    });
  }

  function handleDislike() {
    setInteractions((prev) => {
      const newInteractions = prev.includes("dislike")
        ? prev.filter((i) => i !== "dislike")
        : [...prev, "dislike"];
      if (newInteractions.includes("like")) {
        return newInteractions.filter((i) => i !== "like");
      }
      return newInteractions;
    });
  }

  if (isLoading) {
    return <PostSkeleton />;
  }

  function handleMenuPress(event: any) {
    const { pageX, pageY } = event.nativeEvent || {};

        const   options   =   isMine   ?   myPostMenuOptions   :
foreignPostMenuOptions;

    showMenu({
      x: pageX ?? 0,
      y: pageY ?? 0,
      menuOptions: options,
    });
```

```
  }

  if (!post) {
    return (
      <View style={styles.container}>
        <Text>Завантаження...</Text>
      </View>
    );
  }

  return (
    <View
      style={[
        styles.container,
        fixedSize ? { height: "100%" } : { maxHeight: 648 },
      ]}
    >
      <View style={styles.header}>
        <View style={styles.userInfo}>
                          <TouchableOpacity    onPress={()    =>
router.push(`/${post.creator.id}`)}>
            <Image
             source={getUserAvatar(post?.creator?.avatarUrl || "")}
              style={styles.avatar}
            />
          </TouchableOpacity>
          <Text style={styles.nickname}>
            {post?.creator?.fullName || "Full Name"}
          </Text>
        </View>
                       <TouchableOpacity    style={styles.options}
onPress={handleMenuPress}>
          <Octicons name="kebab-horizontal" size={24} color="#999"
/>
        </TouchableOpacity>
      </View>
      <View style={styles.body}>
        <ScrollView
          contentContainerStyle={[
            styles.postBody,
            fixedSize
              ? { justifyContent: "space-between", height: "100%" }
              : {},
          ]}
          showsVerticalScrollIndicator={false}
          showsHorizontalScrollIndicator={false}
        >
          <View>
            <View style={styles.titleContainer}>
              <View style={styles.verticalLine} />
                      <Text  style={styles.title}>{post?.title  ||
"Title"}</Text>
```

```
            </View>
                                          {post.text     ?     <Text
style={styles.text}>{post.text}</Text> : null}
          </View>
          {post.attachments && post.attachments.length > 0 && (
            <AttachmentsGrid attachments={post.attachments} />
          )}
        </ScrollView>
      </View>
      <View style={styles.footer}>
        <View style={styles.actions}>
                        <TouchableOpacity    style={styles.options}
onPress={handleLike}>
            <Octicons
              name="thumbsup"
              size={24}
                color={interactions.includes("like") ? "#91b49d" :
"#999"}
            />
          </TouchableOpacity>
                         <TouchableOpacity    style={styles.options}
onPress={handleDislike}>
            <Octicons
              name="thumbsdown"
              size={24}
             color={interactions.includes("dislike") ? "#e57373" :
"#999"}
            />
          </TouchableOpacity>
          <TouchableOpacity
            style={styles.options}
            onPress={() => console.log("Direct message pressed")}
          >
            <Octicons name="comment" size={24} color="#999" />
          </TouchableOpacity>
          <TouchableOpacity
            style={styles.options}
            onPress={() => console.log("Discussions pressed")}
          >
                    <Octicons  name="comment-discussion"  size={24}
color="#999" />
          </TouchableOpacity>
        </View>
                       <TouchableOpacity    style={styles.options}
onPress={handleSaved}>
          <Octicons
            name="bookmark"
            size={24}
               color={interactions.includes("save")  ?  "#f8c125"  :
"#999"}
          />
        </TouchableOpacity>
```

```
      </View>
    </View>
  );
};
```

UserWithButton.tsx:

```
import { UserPublic } from "@/constants/types";
import getUserAvatar from "@/constants/user";
import { getRemoteData } from "@/utils/api";
import { router } from "expo-router";
import React, { useEffect, useState } from "react";
import { Image, Text, View, StyleSheet, TouchableOpacity } from
"react-native";

export default function UserWithButton(props: {
  userId: number;
  button?: React.ReactNode;
  description?: string;
}) {
  const { button, description } = props;

  const [user, setUser] = useState<UserPublic | null>(null);

  const avatar = getUserAvatar(user?.avatarUrl || "");

  function getDescription(text: string | undefined) {
    if (text === undefined) {
      return "";
    }
    if (text.trim() === "") {
      return "";
    }
    if (text.length > 40) {
      return text.slice(0, 40).concat("...");
    }
    return text;
  }

  useEffect(() => {
     getRemoteData<UserPublic>(`/users/${props.userId}`).then((data
) => {
      setUser(data);
    });
  }, []);

  return (
    <View style={styles.container}>
      <View style={styles.absoluteContainer}></View>
      <View style={styles.userContainer}>
        <View style={styles.userDataContainer}>
          <TouchableOpacity
```

```
            onPress={() => {
              router.push(`/${user?.id}`);
            }}
          >
            <Image source={avatar} style={styles.imageProfileStyle}
/>
          </TouchableOpacity>
          <View style={styles.textContainer}>
                <Text  style={styles.userName}>{user?.fullName  ||
"user"}</Text>
            <Text style={styles.description}>
                                {description   ?   description   :
getDescription(user?.description)}
            </Text>
          </View>
        </View>
                                    {button       &&       <View
style={styles.buttonContainer}>{button}</View>}
      </View>
    </View>
  );
}
```

Chat.tsx:

```
import React from "react";
import { ScrollView } from "react-native";
import ChatMessage from "./ChatMessage";
import { StyleSheet, View } from "react-native";
import { MessageData } from "@/constants/types";

interface ChatProps {
  messages: MessageData[];
}

export default function Chat(props: ChatProps) {
  const { messages } = props;
  return (
    <ScrollView>
      <View style={styles.messageList}>
        {messages.map((message, index) => (
          <ChatMessage key={index} message={message} />
        ))}
      </View>
    </ScrollView>
  );
}
```

ChatPreview.tsx:

```
import { MessageData } from "@/constants/types";
```

```
import getUserAvatar from "@/constants/user";
import { getRemoteData } from "@/utils/api";
import React, { useEffect, useState } from "react";
import { StyleSheet, Text, View, Image, TouchableOpacity } from
"react-native";

interface ChatItemProps {
  chatId: number;
  avatarUrl: string;
  title: string;
  onPress: () => void;
  onLongPress: (event: any) => void;
  unreadCount?: number;
}

export const ChatItem: React.FC<ChatItemProps> = ({
  chatId,
  avatarUrl,
  title,
  onPress,
  onLongPress,
  unreadCount = 0,
}) => {
  const [message, setMessage] = useState<MessageData | null>(null);
  useEffect(() => {
    getRemoteData(`/chats/${chatId}`).then((data) => {
      console.log(data.messages[data.messages.length - 1]);
      setMessage(data.messages[data.messages.length - 1]);
    });
  }, []);

  function getTime(date: Date | string) {
    if (typeof date === "string") {
      date = new Date(date);
    }
    const hours = date.getHours();
    const minutes = date.getMinutes();
    return `${hours}:${minutes < 10 ? "0" + minutes : minutes}`;
  }
  return (
    <TouchableOpacity
      style={styles.container}
      onPress={onPress}
      onLongPress={onLongPress}
      delayLongPress={300}
    >
      <View style={styles.background}></View>
      <View style={styles.chatItem}>
                        <Image     source={getUserAvatar(avatarUrl)}
style={styles.avatar} />
        <View style={styles.dataContainer}>
          <View style={styles.titleContainer}>
```

```
            <Text style={styles.title} numberOfLines={1}>
              {title}
            </Text>
            {unreadCount > 0 && (
              <View style={styles.unreadCount}>
                                                                <Text
style={styles.unreadCountText}>{unreadCount}</Text>
              </View>
            )}
          </View>
          <View style={styles.messageContainer}>
            <Text style={styles.lastMessage} numberOfLines={1}>
              {message?.text}
            </Text>
            <Text style={styles.date}>
             {message && getTime(message?.createdAt || new Date())}
            </Text>
          </View>
        </View>
      </View>
    </TouchableOpacity>
  );
};
```

api.ts:

```
import { getData } from "./storage";

var API_URL = "";

export function setApiUrl(url: string) {
  API_URL = url;
}

async     function     withAuthHeaders(options?:     RequestInit):
Promise<RequestInit> {
  const token = await getData<string>("token");
  return {
    ...options,
    headers: {
      ...(options?.headers || {}),
      ...(token ? { Authorization: `Bearer ${token}` } : {}),
    },
  };
}

export async function getRemoteData<T = any>(
  url: string,
  options?: RequestInit
): Promise<T> {
  const requestOptions = await withAuthHeaders({ ...options, method:
"GET" });
```

```
  console.log("getRemoteData", API_URL + url, requestOptions);
  const response = await fetch(API_URL + url, requestOptions);
  if (!response.ok) throw new Error(await response.text());
  return response.json();
}

export async function postRemoteData<T = any>(
  url: string,
  body?: any,
  options?: RequestInit
): Promise<T> {
  const requestOptions = await withAuthHeaders({
    ...options,
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      ...(options?.headers || {}),
    },
    body: JSON.stringify(body),
  });
  console.log("postRemoteData", API_URL + url, requestOptions);
  const response = await fetch(API_URL + url, requestOptions);
  if (!response.ok) throw new Error(await response.text());
  return response.json();
}

export async function putRemoteData<T = any>(
  url: string,
  body?: any,
  options?: RequestInit
): Promise<T> {
  const requestOptions = await withAuthHeaders({
    ...options,
    method: "PUT",
    headers: {
      "Content-Type": "application/json",
      ...(options?.headers || {}),
    },
    body: JSON.stringify(body),
  });
  const response = await fetch(API_URL + url, requestOptions);
  if (!response.ok) throw new Error(await response.text());
  return response.json();
}

export async function deleteRemoteData<T = any>(
  url: string,
  options?: RequestInit
): Promise<T> {
  const requestOptions = await withAuthHeaders({
    ...options,
    method: "DELETE",
```

```
  });
  const response = await fetch(API_URL + url, requestOptions);
  if (!response.ok) throw new Error(await response.text());
  return response.json();
}

export async function patchRemoteData<T = any>(
  url: string,
  body?: any,
  options?: RequestInit
): Promise<T> {
  const requestOptions = await withAuthHeaders({
    ...options,
    method: "PATCH",
    headers: {
      "Content-Type": "application/json",
      ...(options?.headers || {}),
    },
    body: JSON.stringify(body),
  });
  const response = await fetch(API_URL + url, requestOptions);
  if (!response.ok) throw new Error(await response.text());
  return response.json();
}
```

profile/index.tsx:

```
import React, { useCallback, useEffect, useState } from "react";
import {
  StyleSheet,
  Text,
  View,
  Image,
  TouchableOpacity,
  ScrollView,
  ActivityIndicator,
} from "react-native";
import { AntDesign, Octicons } from "@expo/vector-icons";
import { Post } from "@/components/Post";
import { deleteRemoteData, getRemoteData } from "@/utils/api";
import { useIsFocused } from "@react-navigation/native";
import getUserAvatar from "@/constants/user";
import { TabView, SceneMap, TabBar } from "react-native-tab-view";
import { UserPublic } from "@/constants/types";
import { PostData } from "@/constants/types";
import { router, useLocalSearchParams } from "expo-router";

const categories = [
  { key: "created", icon: "three-bars", label: "Створені" },
  { key: "liked", icon: "heart", label: "Вподобані" },
  { key: "saved", icon: "bookmark", label: "Збережені" },
];
```

```
export default function ProfileScreen() {
  const [userData, setUserData] = useState<UserPublic | null>(null);
  const [createdPosts, setCreatedPosts] = useState<PostData[]>([]);
  const [likedPosts, setLikedPosts] = useState<PostData[]>([]);
  const [savedPosts, setSavedPosts] = useState<PostData[]>([]);
  const [friendsCount, setFriendsCount] = useState(0);
  const [followingCount, setFollowingCount] = useState(0);
  const [followersCount, setFollowersCount] = useState(0);
  const [index, setIndex] = useState(0);
  const [routes] = useState(categories);
  const [userLoading, setUserLoading] = useState(true);
  const isFocused = useIsFocused();
  var updatePage = useLocalSearchParams().updatePage;
  const [refreshKey, setRefreshKey] = useState(0);

  useEffect(() => {
    console.log("User data:", userData);
    fetchData();
  }, [isFocused, updatePage, refreshKey]);

  async function fetchData() {
    setUserLoading(true);
    await Promise.all([
      getRemoteData("/me").then((data) => setUserData(data)),
       getRemoteData("/me/interactions?type=create").then(setCreate
dPosts),
       getRemoteData("/me/interactions?type=like").then(setLikedPos
ts),
       getRemoteData("/me/interactions?type=save").then(setSavedPos
ts),
      getRemoteData("/me/relations?type=friends").then((data) => {
        setFriendsCount(data.length);
      }),
       getRemoteData("/me/relations?type=following").then((data) =>
{
        setFollowingCount(data.length);
      }),
       getRemoteData("/me/relations?type=followers").then((data) =>
{
        setFollowersCount(data.length);
      }),
    ]).then(() => {
      setTimeout(() => {
        setUserLoading(false);
      }, 500);
    });
  }

  React.useEffect(() => {
    if (index === 0) {
```

```
      getRemoteData("/me/interactions?type=create").then(setCreate
dPosts);
    } else if (index === 1) {
      getRemoteData("/me/interactions?type=like").then(setLikedPos
ts);
    } else if (index === 2) {
      getRemoteData("/me/interactions?type=save").then(setSavedPos
ts);
    }
  }, [index]);

  const renderScene = SceneMap({
    created: () => <TabPostsList posts={createdPosts} />,
    liked: () => <TabPostsList posts={likedPosts} />,
    saved: () => <TabPostsList posts={savedPosts} />,
  });

  function TabPostsList({ posts }: { posts: PostData[] }) {
    return (
      <ScrollView
        style={styles.postsContainer}
        showsVerticalScrollIndicator={false}
      >
        {posts.length === 0 ? (
          <View style={styles.empty}>
            <Text style={{ color: "#7b7b7b" }}>Постів немає</Text>
          </View>
        ) : (
          posts.map((item) => (
            <View style={styles.postWrapper} key={item.id}>
              <Post
                postId={item.id}
                key={item.id}
                onDelete={async () => {
                  setCreatedPosts((prev) =>
                    prev.filter((post) => post.id !== item.id)
                  );
                  setLikedPosts((prev) =>
                    prev.filter((post) => post.id !== item.id)
                  );
                  setSavedPosts((prev) =>
                    prev.filter((post) => post.id !== item.id)
                  );
                  setRefreshKey((k) => k + 1);
                }}
              />
            </View>
          ))
        )}
      </ScrollView>
    );
  }
```

```
  if (userLoading) {
    return (
      <View
        style={{
          flex: 1,
          justifyContent: "center",
          alignItems: "center",
          backgroundColor: "#f6f6f6",
        }}
      >
        <ActivityIndicator size="large" color="#cecece" />
      </View>
    );
  }

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <Text style={styles.nickname}>{userData?.nickname}</Text>
        <View style={styles.actions}>
          <TouchableOpacity
            style={styles.actionIcon}
            onPress={() => router.push("/profile/create")}
          >
            <AntDesign name="pluscircleo" size={32} color="#333" />
          </TouchableOpacity>
          <TouchableOpacity
            style={styles.actionIcon}
            onPress={() => router.push("/profile/settings")}
          >
            <Octicons name="gear" size={32} color="#333" />
          </TouchableOpacity>
        </View>
      </View>
      <View style={styles.body}>
        <View style={styles.userInfo}>
          <Image
            source={getUserAvatar(userData?.avatarUrl || "")}
            resizeMode="cover"
            style={styles.profileImage}
          />
          <View style={styles.infoBlock}>
              <Text style={styles.userName}>{userData?.fullName ||
"User"}</Text>
            <View style={styles.relations}>
              <TouchableOpacity
                style={styles.statItem}
                onPress={() => {
                  router.push("/profile/friends");
                }}
              >
```

```
              <Text style={styles.statNumber}>{friendsCount}</Text>
                <Text style={styles.statLabel}>Друзі</Text>
              </TouchableOpacity>
              <TouchableOpacity
                style={styles.statItem}
                onPress={() => {
                  router.push("/profile/followers");
                }}
              >
                                                                  <Text
style={styles.statNumber}>{followersCount}</Text>
                <Text style={styles.statLabel}>Підписники</Text>
              </TouchableOpacity>
              <TouchableOpacity
                style={styles.statItem}
                onPress={() => {
                  router.push("/profile/following");
                }}
              >
                                                                  <Text
style={styles.statNumber}>{followingCount}</Text>
                <Text style={styles.statLabel}>Підписки</Text>
              </TouchableOpacity>
            </View>
            {userData?.description && (
              <View style={styles.biographyContainer}>
                <Text style={styles.biographyText}>
                  {userData?.description}
                </Text>
              </View>
            )}
          </View>
        </View>
        <TabView
          commonOptions={{
            icon: ({ route, focused }) => (
              <Octicons
                // @ts-ignore
                name={route.icon}
                size={24}
                color={focused ? "#000000" : "#808080"}
              />
            ),
          }}
          navigationState={{ index, routes }}
          renderScene={renderScene}
          onIndexChange={setIndex}
          renderTabBar={(props) => (
            <TabBar
              {...props}
              indicatorStyle={{ height: 0 }}
              style={{
```

```
                backgroundColor: "transparent",
                shadowColor: "transparent",
                borderBottomWidth: 1,
                borderBottomColor: "#cecece",
                borderTopWidth: 1,
                borderTopColor: "#cecece",
              }}
            />
          )}
        />
      </View>
    </View>
  );
}
```

app/[userId].tsx:

```
import React, { useEffect, useState } from "react";
import {
  StyleSheet,
  Text,
  View,
  Image,
  TouchableOpacity,
  FlatList,
  ScrollView,
  ActivityIndicator,
} from "react-native";
import { useLocalSearchParams } from "expo-router";
import { Octicons, Ionicons } from "@expo/vector-icons";
import { useRouter, Stack } from "expo-router";
import { Post } from "@/components/Post";
import { getRemoteData, patchRemoteData } from "@/utils/api";
import { PostData, UserPublic, UserPublicFull } from
"@/constants/types";
import getUserAvatar from "@/constants/user";
import { useIsFocused } from "@react-navigation/native";

export default function ProfileScreen() {
  const { userId } = useLocalSearchParams();
  const router = useRouter();
  const [user, setUser] = useState<UserPublicFull | null>(null);
  const [posts, setPosts] = useState<PostData[]>([]);
  const [isSubscribed, setIsSubscribed] = useState(false);
  const [userLoading, setUserLoading] = useState(true);
  const isFocused = useIsFocused();

  useEffect(() => {
    fetchData();
  }, [isFocused]);
```

```
  async function fetchData() {
    setUserLoading(true);
    await Promise.all([
      getRemoteData(`/users/${userId}`).then((data) => {
        setUser(data);
        console.log("User data:", data);
      }),
       getRemoteData(`/me/relations?type=following`).then((followin
gs) => {
         const followingIds = followings.map((user: UserPublic) =>
user.id);
         getRemoteData(`/me/relations?type=friends`).then((friends)
=> {
            const friendIds = friends.map((user: UserPublic) =>
user.id);
          setIsSubscribed(
            followingIds.includes(Number(userId)) ||
              friendIds.includes(Number(userId))
          );
        });
      }),
       getRemoteData<PostData[]>(`/users/${userId}/posts`).then((da
ta) => {
        setPosts(data);
      }),
    ]).then(() => {
      setTimeout(() => {
        setUserLoading(false);
      }, 500);
    });
  }

  useEffect(() => {}, []);

  function toggleSubscriptionState(state: boolean) {
    setIsSubscribed(!state);
    patchRemoteData(`/users/${userId}/follow`, !state);
  }

  if (userLoading) {
    return (
      <View
        style={{
          flex: 1,
          justifyContent: "center",
          alignItems: "center",
          backgroundColor: "#f6f6f6",
        }}
      >
        <ActivityIndicator size="large" color="#cecece" />
      </View>
    );
```

```
  }

  return (
    <>
      <View style={styles.container}>
        <View style={styles.header}>
          <TouchableOpacity
            style={styles.actionIcon}
            onPress={() => {
              if (router.canGoBack()) {
                router.back();
              } else {
                router.push("/");
              }
            }}
          >
            <Octicons name="chevron-left" size={32} color="#4d4d4d"
/>
          </TouchableOpacity>

          <View style={styles.profileProps}>
            <Text style={styles.nickname}>
              {user?.nickname || "Користувач"}
            </Text>
            <TouchableOpacity style={styles.actionIcon}>
                      <Ionicons  name="ellipsis-vertical"  size={32}
color="#4d4d4d" />
            </TouchableOpacity>
          </View>
        </View>
        <View style={styles.userInfo}>
          <Image
            source={getUserAvatar(user?.avatarUrl || "")}
            style={styles.profileImage}
          />
          <View style={styles.infoBlock}>
                <Text  style={styles.userName}>{user?.fullName  ||
"User"}</Text>
            {user?.description ? (
              <View style={styles.biographyContainer}>
                                                                <Text
style={styles.biographyText}>{user?.description}</Text>
              </View>
            ) : null}
          </View>
        </View>
        <View style={styles.buttonHolder}>
          <TouchableOpacity
            onPress={() => toggleSubscriptionState(isSubscribed)}
            style={[
              styles.buttonContainer,
              isSubscribed && styles.buttonInactiveContainer,
```

```
            ]}
          >
            <Text
              style={[
                styles.buttonText,
                isSubscribed && styles.buttonInactiveText,
              ]}
            >
              {isSubscribed ? "Відписатися" : "Підписатися"}
            </Text>
          </TouchableOpacity>
        </View>
        <ScrollView
          style={styles.scrollView}
          showsVerticalScrollIndicator={false}
          showsHorizontalScrollIndicator={false}
          contentContainerStyle={styles.postsContainer}
        >
          {posts?.length > 0 && posts.map((item) => <Post
postId={item.id} />)}
        </ScrollView>
      </View>
    </>
  );
}
```

friends.tsx:

```
import { Ionicons } from "@expo/vector-icons";
import UserWithButton from "@/components/UserWithButton";
import { router } from "expo-router";
import React, { useEffect, useState } from "react";
import { View, StyleSheet, TouchableOpacity, Text } from "react-
native";
import { getRemoteData } from "@/utils/api";
import { UserPublic } from "@/constants/types";

export default function FriendsScreen() {
  const [users, setUsers] = useState<UserPublic[] | null>(null);

  useEffect(() => {
    getRemoteData(`/me/relations?type=friends`).then(setUsers);
  }, []);
  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <TouchableOpacity
          style={styles.actionIcon}
          onPress={() => {
            router.back();
          }}
        >
```

```
          <Ionicons name="arrow-back" size={32} color="#222" />
        </TouchableOpacity>
        <Text style={styles.headerTitle}>Друзі</Text>
        <View style={styles.actionIcon}></View>
      </View>
      <View style={styles.body}>
        {users && users.length > 0 ? (
          users.map((user) => (
            <UserWithButton
              key={user.id}
              userId={user.id}
              button={
                <TouchableOpacity
                  style={styles.button}
                  onPress={() => {
                    router.push(`/${user.id}`);
                  }}
                >
                  <Text style={styles.buttonText}>Перейти</Text>
                </TouchableOpacity>
              }
            />
          ))
        ) : (
          <View style={{ padding: 16, alignItems: "center" }}>
            <Text style={{ color: "#7b7b7b" }}>Друзів немає</Text>
          </View>
        )}
      </View>
    </View>
  );
}
```

followers.tsx:

```
import React, { useEffect, useState } from "react";
import { View, StyleSheet, TouchableOpacity, Text } from "react-
native";
import { router } from "expo-router";
import { Ionicons } from "@expo/vector-icons";
import UserWithButton from "@/components/UserWithButton";
import { getRemoteData, patchRemoteData } from "@/utils/api";
import { UserPublic } from "@/constants/types";

export default function FollowersScreen() {
  const [users, setUsers] = useState<UserPublic[] | null>(null);
  const [subscribedIds, setSubscribedIds] = useState<number[]>([]);

  useEffect(() => {
    getRemoteData(`/me/relations?type=followers`).then(setUsers);
     getRemoteData<UserPublic[]>(`/me/relations?type=following`).th
en(
```

```
      (users) => {
        const ids = users.map((user) => user.id);
         getRemoteData<UserPublic[]>(`/me/relations?type=friends`).
then(
          (friends) => {
            const friendIds = friends.map((user) => user.id);
            const allIds = [...ids, ...friendIds];
            setSubscribedIds(allIds);
          }
        );
      }
    );
  }, []);

  const handleSubscribe = (userId: number) => {
    patchRemoteData(`/users/${userId}/follow`, true);
    setSubscribedIds((prev) => [...prev, userId]);
  };

  const handleUnsubscribe = (userId: number) => {
    patchRemoteData(`/users/${userId}/follow`, false);
    setSubscribedIds((prev) => prev.filter((id) => id !== userId));
  };

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <TouchableOpacity
          style={styles.actionIcon}
          onPress={() => {
            router.back();
          }}
        >
          <Ionicons name="arrow-back" size={32} color="#222" />
        </TouchableOpacity>
        <Text style={styles.headerTitle}>Підписники</Text>
        <View style={styles.actionIcon}></View>
      </View>
      <View style={styles.body}>
        {users && users.length > 0 ? (
          users.map((user) => {
            const isSubscribed = subscribedIds.includes(user.id);
            return (
              <UserWithButton
                key={user.id}
                userId={user.id}
                button={
                  isSubscribed ? (
                    <TouchableOpacity
                     style={[styles.button, styles.buttonInactive]}
                      onPress={() => handleUnsubscribe(user.id)}
                    >
```

```
                    <Text
                                        style={[styles.buttonText,
styles.buttonInactiveText]}
                    >
                      Відписатись
                    </Text>
                  </TouchableOpacity>
                ) : (
                  <TouchableOpacity
                    style={styles.button}
                    onPress={() => handleSubscribe(user.id)}
                  >
                                                          <Text
style={styles.buttonText}>Підписатись</Text>
                  </TouchableOpacity>
                )
              }
            />
          );
        })
      ) : (
        <View style={{ padding: 16, alignItems: "center" }}>
              <Text  style={{  color:  "#7b7b7b"  }}>Підписників
немає</Text>
        </View>
      )}
    </View>
  </View>
 );
}
```

following.tsx:

```
import { Ionicons } from "@expo/vector-icons";
import { router } from "expo-router";
import React, { useEffect, useState } from "react";
import { View, StyleSheet, TouchableOpacity, Text } from "react-
native";
import UserWithButton from "@/components/UserWithButton";
import { UserPublic } from "@/constants/types";
import { getRemoteData, patchRemoteData, postRemoteData } from
"@/utils/api";

export default function FollowingScreen() {
  const [users, setUsers] = useState<UserPublic[] | null>(null);
      const      [unsubscribedIds,      setUnsubscribedIds]      =
useState<number[]>([]);

  useEffect(() => {
    getRemoteData(`/me/relations?type=following`).then(setUsers);
  }, []);
```

```
  async function handleUnsubscribe(userId: number) {
    patchRemoteData(`/users/${userId}/follow`, false);
    setUnsubscribedIds((prev) => [...prev, userId]);
  }

  async function handleSubscribe(userId: number) {
    patchRemoteData(`/users/${userId}/follow`, true);
      setUnsubscribedIds((prev)  =>  prev.filter((id)  =>  id  !==
userId));
  }

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <TouchableOpacity
          style={styles.actionIcon}
          onPress={() => {
            router.back();
          }}
        >
          <Ionicons name="arrow-back" size={32} color="#222" />
        </TouchableOpacity>
        <Text style={styles.headerTitle}>Підписки</Text>
        <View style={styles.actionIcon}></View>
      </View>
      <View style={styles.body}>
        {users && users.length > 0 ? (
          users.map((user) => {
            const isSubscribed = !unsubscribedIds.includes(user.id);
            return (
              <UserWithButton
                key={user.id}
                userId={user.id}
                button={
                  isSubscribed ? (
                    <TouchableOpacity
                      style={styles.button}
                      onPress={() => handleUnsubscribe(user.id)}
                    >
                                                                <Text
style={styles.buttonText}>Відписатись</Text>
                    </TouchableOpacity>
                  ) : (
                    <TouchableOpacity
                      style={[styles.button, styles.buttonInactive]}
                      onPress={() => handleSubscribe(user.id)}
                    >
                      <Text
                                              style={[styles.buttonText,
styles.buttonInactiveText]}
                      >
                        Підписатись
```

```
                    </Text>
                  </TouchableOpacity>
                )
              }
            />
          );
        })
      ) : (
        <View style={{ padding: 16, alignItems: "center" }}>
          <Text style={{ color: "#7b7b7b" }}>Підписок немає</Text>
        </View>
      )}
    </View>
  </View>
  );
}
```

chats/index.tsx:

```
import React, { useState, useContext, useEffect } from "react";
import {
  StyleSheet,
  FlatList,
  View,
  Text,
  TouchableOpacity,
  Pressable,
  TextInput,
} from "react-native";
import { useRouter } from "expo-router";
import { ChatItem } from "@/components/ChatPreview";
import { Octicons, AntDesign } from "@expo/vector-icons";
import { ContextMenuContext } from "@/contexts/ContextMenuContext";
import { useFocusEffect } from "@react-navigation/native";
import { getRemoteData } from "@/utils/api";
import { ChatPreview } from "@/constants/types";

const tabs = [
  {
    label: "Всі чати",
    key: "all",
  },
  {
    label: "Друзі",
    key: "private",
  },
  { label: "Групові чати", key: "group" },
  { label: "Дискусії", key: "discussion" },
];

export default function ChatsScreen() {
  const router = useRouter();
```

```
  const { showMenu } = useContext(ContextMenuContext);

  const [activeTab, setActiveTab] = useState(tabs[0]);
  const [chats, setChats] = useState<ChatPreview[]>([]);
        const       [categoryChats,       setCategoryChats]       =
useState<ChatPreview[]>([]);
  const [inputValue, setInputValue] = useState("");

  useFocusEffect(
    React.useCallback(() => {
      getRemoteData(`/chats`).then((data) => {
        setChats(data);
      });
    }, [])
  );
  useEffect(() => {
    setCategoryChats(
      chats.filter((chat) => {
        if (activeTab.key === "all") return true;
        return chat.type === activeTab.key;
      })
    );
  }, [activeTab, chats]);

  const handleDeleteChat = (chatId: number) => {
    alert(`Видалити чат ${chatId}`);
  };
  const handleMuteChat = (chatId: number) => {
    alert(`Вимкнути сповіщення для чату ${chatId}`);
  };
  const handlePinChat = (chatId: number) => {
    alert(`Прикріпити чат ${chatId}`);
  };

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <View style={styles.searchBar}>
          <Octicons name="search" size={24} color="#808080" />
          <TextInput
            value={inputValue}
            onChangeText={setInputValue}
            style={{ outlineColor: "none", outline: "none" }}
          />
        </View>

                            <TouchableOpacity     onPress={()     =>
router.push("/chats/create")}>
           <AntDesign name="pluscircleo" size={32} color="#313131"
/>
        </TouchableOpacity>
      </View>
```

```
      <View style={styles.tabsContainer}>
        {tabs.map((tab) => (
          <Pressable
            key={tab.key}
                    style={[styles.tab,  activeTab  ===  tab  &&
styles.activeTab]}
            onPress={() => setActiveTab(tab)}
          >
            <Text
              style={[
                styles.tabText,
                activeTab === tab && styles.activeTabText,
              ]}
            >
              {tab.label}
            </Text>
          </Pressable>
        ))}
      </View>
      <FlatList
        data={categoryChats}
        keyExtractor={(item) => item.id.toString()}
        renderItem={({ item }) => {
          return (
            <ChatItem
              unreadCount={0}
              chatId={item.id}
              avatarUrl={item?.avatarUrl || ""}
              title={item?.title || "Користувач"}
              onPress={() => router.push(`/chats/${item.id}`)}
              onLongPress={(event: any) => {
                const { pageX, pageY } = event.nativeEvent;
                showMenu({
                  x: pageX,
                  y: pageY,
                  menuOptions: [
                    {
                      label: "Видалити чат",
                      onPress: () => handleDeleteChat(item.id),
                      style: { color: "#ff0000" },
                    },
                    {
                      label: "Вимкнути сповіщення",
                      onPress: () => handleMuteChat(item.id),
                    },
                    {
                      label: "Прикріпити",
                      onPress: () => handlePinChat(item.id),
                    },
                  ],
                });
              }}
```

```
            />
          );
        }}
      />
    </View>
  );
}
```

chats/create.tsx:

```
import { getRemoteData } from "@/utils/api";
import { Octicons } from "@expo/vector-icons";
import React, { useEffect, useState } from "react";
import {
  StyleSheet,
  View,
  TextInput,
  Text,
  TouchableOpacity,
} from "react-native";
import { UserPublic } from "@/constants/types";
import UserWithButton from "@/components/UserWithButton";
import { FlatList } from "react-native-gesture-handler";
import { router } from "expo-router";

export default function CreateChatScreen() {
  const [inputValue, setInputValue] = useState("");

  const handleInputChange = (text: string) => {
    setInputValue(text);
  };

  const [users, setUsers] = useState<UserPublic[] | null>(null);

  useEffect(() => {
    getRemoteData(`/me/relations?type=followers`).then(setUsers);
  }, []);

  function openChat(userId: number) {
    alert(`Ви відкрили чат з ${userId}`);
  }

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <TouchableOpacity
          style={styles.actionIcon}
          onPress={() => {
            router.back();
          }}
        >
          <Octicons name="arrow-left" size={32} color="#222" />
```

```
        </TouchableOpacity>
        <View style={styles.searchBar}>
          <Octicons name="search" size={24} color="#808080" />
          <TextInput
            value={inputValue}
            onChangeText={handleInputChange}
            style={{ outlineColor: "none", outline: "none" }}
          />
        </View>
      </View>
      <FlatList
        data={users}
        renderItem={({ item }) => (
          <UserWithButton
            userId={item.id}
            button={
              <TouchableOpacity
                style={styles.buttonContainer}
                onPress={() => {
                  openChat(item.id);
                }}
              >
                <Text style={styles.buttonText}>Відкрити чат</Text>
              </TouchableOpacity>
            }
          />
        )}
        keyExtractor={(item) => item.id.toString()}
        contentContainerStyle={styles.listContainer}
        showsVerticalScrollIndicator={false}
        showsHorizontalScrollIndicator={false}
        ListEmptyComponent={
          <View style={{ padding: 16, alignItems: "center" }}>
            <Text style={{ color: "#7b7b7b" }}>
              Немає підписників з якими ви могли б поспілкуватись
            </Text>
          </View>
        }
      />
    </View>
  );
}
```

chats/[chatId]/index.tsx:

```
import React, { useEffect, useState } from "react";
import {
  View,
  StyleSheet,
  TouchableOpacity,
  Image,
  Text,
```

```
  TextInput,
} from "react-native";
import { useLocalSearchParams, useRouter } from "expo-router";
import { Feather, Octicons } from "@expo/vector-icons";
import Chat from "@/components/Chat";
import { getRemoteData } from "@/utils/api";
import {
  ChatData,
  GroupChatData,
  PrivateChatData,
  MessageData,
} from "@/constants/types";
import getUserAvatar from "@/constants/user";
import { postRemoteData } from "@/utils/api";

interface ChatProps {
  info: ChatData;
  additional: PrivateChatData | GroupChatData;
  messages: MessageData[];
}

export default function ChatView() {
  const [chat, setChat] = useState<ChatProps | null>(null);
  const { chatId } = useLocalSearchParams();
  const router = useRouter();
  const [inputHeight, setInputHeight] = useState(32);
  const [avatar, setAvatar] = useState<any>();
  const [title, setTitle] = useState<string>("");
  const [message, setMessage] = useState<string>("");

  useEffect(() => {
    getRemoteData(`/chats/${chatId}`).then((data) => {
      setChat(data);
    });
  }, []);

  useEffect(() => {
    if (chat) {
      if (chat.info.type === "group") {
        setAvatar(chat?.additional.avatarUrl);
      } else if (chat.info.type === "private") {
        setAvatar(getUserAvatar(chat?.additional.user.avatarUrl));
      }
                            setTitle(chat?.additional.title        ||
chat?.additional.user.fullName || "");
    }
  }, [chat]);

  const handleSend = async () => {
    if (message.trim() === "") return;

    try {
```

```
      await postRemoteData(`/messages/${chatId}`, { text: message
});
      setMessage("");
    } catch (error) {
      console.error("Error sending message:", error);
    }

    getRemoteData(`/chats/${chatId}`).then((data) => {
      setChat(data);
    });
  };

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <TouchableOpacity
          style={styles.actionIcon}
          onPress={() => {
            if (router.canGoBack()) {
              router.back();
            } else {
              router.replace("/chats");
            }
          }}
        >
           <Octicons name="chevron-left" size={32} color="#4d4d4d"
/>
        </TouchableOpacity>
        <View style={styles.chatProps}>
          <View style={styles.chatInfo}>
            <TouchableOpacity
              onPress={() => {
                if (chat?.info.type === "group") {
                  router.push(`/chats/${chatId}/info`);
                } else if (chat?.info.type === "private") {
                  router.push(`/${chat?.additional.user.id}`);
                }
              }}
            >
              <Image style={styles.avatar} source={avatar} />
            </TouchableOpacity>
            <Text style={styles.title}>{title}</Text>
          </View>
          <TouchableOpacity style={styles.actionIcon}>
            <Octicons name="gear" size={32} color="#4d4d4d" />
          </TouchableOpacity>
        </View>
      </View>
      <View style={styles.body}>
        <Chat messages={chat?.messages ?? []} />
      </View>
      <View style={styles.footer}>
```

```
        <TouchableOpacity style={styles.actionIcon}>
          <Feather name="paperclip" size={32} color="#999999" />
        </TouchableOpacity>
        <View style={styles.input}>
          <TextInput
            style={[
              styles.inputField,
              {
                outlineColor: "none",
                outline: "none",
                minHeight: 32,
                maxHeight: 120,
                height: inputHeight,
              },
            ]}
            placeholder="Повідомлення"
            placeholderTextColor={"#999999"}
            multiline={true}
            onContentSizeChange={(e) => {
              const newHeight = Math.min(
                Math.max(32, e.nativeEvent.contentSize.height),
                120
              );
              setInputHeight(newHeight);
            }}
            value={message}
            onChangeText={(text) => setMessage(text)}
          />
        </View>
                      <TouchableOpacity    style={styles.actionIcon}
onPress={handleSend}>
          <View style={styles.sendIcon}>
            <Feather name="arrow-up" size={28} color="#ffffff" />
          </View>
        </TouchableOpacity>
      </View>
    </View>
  );
}
```

login.tsx:

```
import React, { useState } from "react";
import {
  View,
  Text,
  TextInput,
  TouchableOpacity,
  StyleSheet,
  Image,
} from "react-native";
import { useRouter } from "expo-router";
```

```
import { getRemoteData, postRemoteData } from "@/utils/api";
import { saveData } from "@/utils/storage";

export default function LoginScreen() {
  const [login, setLogin] = useState("");
  const [password, setPassword] = useState("");
  const router = useRouter();

  function loginRequest() {
    postRemoteData("/authorization/login", {
      login: login,
      password: password,
    }).then((response) => {
      if (response.token) {
        saveData("token", response.token);
        router.replace("/home");
      } else {
        console.error("Registration failed:", response);
      }
    });
  }

  return (
    <View style={styles.container}>
      <View style={{ flex: 1, justifyContent: "space-around" }}>
        <View style={{ gap: 28 }}>
          <Text style={styles.appTitle}>Delagram</Text>
          <View style={styles.imageContainer}>
            <Image
              source={require("../../assets/images/login.png")}
              style={styles.image}
            />
          </View>
        </View>
        <View>
          <Text style={styles.title}>УВІЙТИ В АКАУНТ</Text>
        </View>
        <View style={styles.loginContainer}>
          <View style={styles.inputFields}>
            <View style={styles.inputContainer}>
              <TextInput
                style={
                  (styles.inputField,
                  {
                    outlineColor: "none",
                    outline: "none",
                  })
                }
                placeholder="Нікнейм або e-mail"
                placeholderTextColor="#999999"
                value={login}
                onChangeText={setLogin}
```

```
              keyboardType="default"
            />
          </View>
          <View style={styles.inputContainer}>
            <TextInput
              style={
                (styles.inputField,
                {
                  outlineColor: "none",
                  outline: "none",
                })
              }
              placeholder="Пароль"
              placeholderTextColor="#999999"
              value={password}
              onChangeText={setPassword}
              secureTextEntry={true}
            />
          </View>
          <TouchableOpacity>
                    <Text  style={styles.forgotPassword}>Забули
пароль?</Text>
          </TouchableOpacity>
        </View>
        <View style={styles.loginButtonContainer}>
                  <TouchableOpacity  style={styles.loginButton}
onPress={loginRequest}>
            <Text style={styles.loginButtonText}>Увійти</Text>
          </TouchableOpacity>
          <View style={styles.registerContainer}>
              <Text  style={styles.registerText}>Немає  акаунту?
</Text>
            <TouchableOpacity
                                                  onPress={()     =>
router.replace("/authorization/register")}
            >
                                                          <Text
style={styles.registerLink}>Зареєструватися</Text>
            </TouchableOpacity>
          </View>
        </View>
      </View>
    </View>
  );
}
```

register.tsx:

```
import                  EmailAndPasswordStep                  from
"@/components/registration/Steps/EmailAndPasswordStep";
```

```
import NicknameStep from "@/components/registration/Steps/NicknameStep";
import FullNameStep from "@/components/registration/Steps/FullNameStep";
import AgeStep from "@/components/registration/Steps/AgeStep";
import CountryStep from "@/components/registration/Steps/CountryStep";
import InterestsStep from "@/components/registration/Steps/InterestsStep";
import AvatarStep from "@/components/registration/Steps/AvatarStep";
import NotInterestedStep from "@/components/registration/Steps/NotInterestedStep";
import TermsStep from "@/components/registration/Steps/TermsStep";
import FinishStep from "@/components/registration/Steps/FinishStep";
import React, { useState } from "react";
import { View, Text, TouchableOpacity, StyleSheet } from "react-native";
import { clamp } from "react-native-reanimated";
import { router } from "expo-router";
import { RegistrationFormData, RegistrationRequest } from "@/constants/types";
import { getRemoteData, postRemoteData } from "@/utils/api";
import { saveData } from "@/utils/storage";

export default function RegisterScreen() {
  const [step, setStep] = useState(1);
  const nextStep = () =>
    setStep((prev) => clamp(prev + 1, 1, registrationSteps.length));
  const prevStep = () =>
    setStep((prev) => clamp(prev - 1, 1, registrationSteps.length));

  const [stepValid, setStepValid] = useState(false);

  const [registrationData, setRegistrationData] =
    useState<RegistrationFormData>({
      email: "",
      password: "",
      confirmPassword: "",
      nickname: "",
      fullName: "",
      age: "",
      country: "",
      interests: [],
      notInterested: [],
      avatar: "",
    });

  function updateRegistrationData(field: string, value: any) {
    setRegistrationData((prev) => ({ ...prev, [field]: value }));
  }
```

```
const registrationSteps = [
  <EmailAndPasswordStep
    key={1}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <NicknameStep
    key={2}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <FullNameStep
    key={3}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <AgeStep
    key={4}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <CountryStep
    key={5}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <InterestsStep
    key={6}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <NotInterestedStep
    key={7}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <AvatarStep
    key={8}
    onStepValidChange={setStepValid}
    registrationData={registrationData}
    updateRegistrationData={updateRegistrationData}
  />,
  <TermsStep
    key={9}
```

```
      onStepValidChange={setStepValid}
      registrationData={registrationData}
      updateRegistrationData={updateRegistrationData}
    />,
    <FinishStep
      key={10}
      onStepValidChange={setStepValid}
      registrationData={registrationData}
      updateRegistrationData={updateRegistrationData}
    />,
  ];

  async function submitRegistration() {
    const request: RegistrationRequest = {
      email: registrationData.email,
      password: registrationData.password,
      nickname: registrationData.nickname,
      fullName: registrationData.fullName,
      avatar: registrationData.avatar,
      interests: registrationData.interests,
      notInterested: registrationData.notInterested,
    };

                          postRemoteData("/authorization/register",
request).then((response) => {
      if (response.token) {
        saveData("token", response.token);
        router.replace("/home");
      } else {
        console.error("Registration failed:", response);
      }
    });
  }

  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <Text style={styles.mainTitle}>РЕЄСТРАЦІЯ</Text>
      </View>
     <View style={styles.body}>{registrationSteps[step - 1]}</View>
      <View style={styles.footer}>
        <TouchableOpacity style={styles.button} onPress={prevStep}>
          <Text style={styles.buttonText}>Назад</Text>
        </TouchableOpacity>
        <View style={styles.stepsContainer}>
          {Array.from({ length: registrationSteps.length }).map((_,
index) => (
            <View
              key={index}
              style={[
                styles.step,
```

```
                          index < step ? styles.activeStep : styles.inactiveStep,
                ]}
              />
            ))}
          </View>
          <TouchableOpacity
                          style={[styles.button, !stepValid && styles.buttonDisabled]}
            onPress={
              step === registrationSteps.length
                ? submitRegistration
                : stepValid
                ? nextStep
                : undefined
            }
          >
            <Text
                        style={[styles.buttonText, !stepValid && styles.buttonTextDisabled]}
            >
              Далі
            </Text>
          </TouchableOpacity>
        </View>
      </View>
    );
}
```

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»; що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації; що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів; що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)