

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Мобільний додаток для знайомств та пошуку
друзів»**

Виконав студент групи ІПЗк-20-1
КЛЮЄВ Ілля Олександрович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
КРАВЧЕНКО Світлана Миколаївна

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення

Андрій МОРОЗОВ

«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КЛЮЄВ Ілля Олександрович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Мобільний додаток для знайомств та пошуку друзів»**,
затверджена Наказом закладу вищої освіти від **«14» лютого 2023 р., №70с**

Вихідні дані для роботи: Методи та засоби розробки веб-додатків, аналіз проблематики онлайн-знайомств та розробка додатку за темою роботи.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Власенко О.В.	16.02.2023р.	16.02.2023р.
2	Власенко О.В.	16.02.2023р.	16.02.2023р.
3	Власенко О.В.	16.02.2023р.	16.02.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз проблематики онлайн-знайомств	09 січня 2023 – 11 січня 2023	Виконано
2	Аналіз існуючих програмних продуктів	11 січня 2023 – 18 січня 2023	Виконано
3	Складання вимог до програмного продукту	18 січня 2023 – 20 січня 2023	Виконано
4	Створення дизайн-концепту основних екранів додатку	20 січня 2023 – 22 січня 2023	Виконано
5	Створення дизайн системи, основних компонентів та іконок	22 січня 2023 – 1 лютого 2023	Виконано
6	Створення дизайну екранів додатку	1 лютого 2023 – 7 лютого 2023	Виконано
7	Розробка прототипу	7 лютого 2023 – 10 лютого 2023	Виконано
8	Тестування прототипу на цільовій аудиторії	10 лютого 2023 – 14 лютого 2023	Виконано
9	Розробка прототипу клієнтської частини	14 лютого 2023 – 7 березня 2023	Виконано
10	Тестування прототипу клієнтської частини на цільовій аудиторії	7 березня 2023 – 14 березня 2023	Виконано
11	Проектування архітектури бази даних	14 березня 2023 – 21 березня 2023	Виконано
12	Розробка серверної частини для MVP	21 березня 2023 – 1 квітня 2023	Виконано
13	Розробка клієнтської частини для MVP	1 квітня 2023 – 14 квітня 2023	Виконано
14	Налаштування серверу, Phusion Passenger, CI/CD та PWA можливостей	14 квітня 2023 – 16 квітня 2023	Виконано
15	Запуск альфа-тестування MVP на цільовій аудиторії	16 квітня 2023 – 19 квітня 2023	Виконано
16	Розробка решти функціоналу додатку	19 квітня 2023 – 7 травня 2023	Виконано
17	Повне ручне тестування додатку	7 травня 2023 – 10 травня 2023	Виконано
18	Запуск бета-тестування на цільовій аудиторії	10 травня 2023 – 14 травня 2023	Виконано
19	Підготовка до релізу	14 травня 2023 – 19 травня 2023	Виконано
20	Реліз та впровадження додатку	19 травня 2023 – 21 травня 2023	Виконано
21	Аналіз одержаних результатів	21 травня 2023 – 1 червня 2023	Виконано
22	Оформлення пояснювальної записки	1 червня 2023 – 5 червня 2023	Виконано
23	Попередній захист	6 червня 2023	Виконано

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «мобільний додаток для знайомств та пошуку друзів» складається з трьох розділів, висновків, списку використаної літератури та додатків.

Метою кваліфікаційної роботи є аналіз існуючих програмних додатків у сфері інтернет знайомств та створення мобільного додатку, що реалізує функціонал, відсутній у існуючих програмних системах.

Текстова частина викладена на 110 сторінках друкованого тексту, має 1 сторінку додатків.

Список використаних джерел містить 45 найменувань і займає 5 сторінок. В роботі наведено 100 рисунків. Загальний обсяг роботи – 127 сторінок.

У першому розділі було проаналізовано актуальність теми інтернет знайомств, проведено дослідження думок та побажань цільової аудиторії, проаналізовано існуючі програмні продукти, виявлено їх сильні та слабкі сторони, розроблено концепцію програмного продукту, виконано аналіз вимог до програмного забезпечення, та обґрунтовано вибір засобів реалізації та використовуваних технологічних процесів.

У другому розділі було розроблено дизайн та виконано прототипування екранів додатку на платформі Figma, розроблено прототип клієнтської частини для опробування його на вибірці з цільової аудиторії, проаналізовано їх відгук, розроблено архітектуру бази даних, створено серверну частину, налаштовано CI/CD конвеєр для безперервного постачання оновлень, використано налаштування серверу, налаштовано систему Phusion Passenger, реалізовано функціонал Progressive Web Application.

					ІІЗ.КР.Б – 121 – 23 – ІІЗ						
Зм.	Арк.	№ докум.	Підпис	Дата	Мобільний додаток для знайомств та пошуку друзів Пояснювальна записка			Літ.	Арк.	Аркушів	
Розроб.		Клюєв І.О.									
Керівник		Власенко О.В.								4	127
н. контр.											
Зав. каф.		Морозов А.В.									
					Житомирська політехніка, група ІІІЗК-20-1						

У третьому розділі було детально описано інтерфейс та функціонал створеного додатку та вбудованої панелі адміністрування, а також додана інформація з підтримки системи та налаштоване cron-завдання для автоматичного нічного резервного копіювання бази даних.

Висновок містить в собі огляд виконаної роботи, оцінку результатів реального впровадження отриманого програмного забезпечення, аналіз статистики, поведінки користувачів та опис планів подальшого розвитку системи.

Ключові слова: ЗНАЙОМСТВА, ДОДАТОК, VUE.JS, PINIA, VITE, PHUSION PASSENGER, PUSH NOTIFICATIONS, WEB SOCKETS, NEST.JS, TYPESCRIPT, PWA, ODM, MONGO DB, MONGOOSE, TYPEGOOSE, KINDAGOOSE

ABSTRACT

The explanatory note to the qualification work on the topic "mobile application for dating and finding friends" consists of three chapters, conclusions, a list of references and appendixes.

The purpose of the qualification work is to analyze existing software applications in the field of online dating and create a mobile application that implements functionality that is not available in existing software systems.

The text part is set out on 110 pages of printed text, with 1 page of appendixes.

The list of references contains 45 titles and covers 5 pages. The paper contains 100 figures. The total volume of the paper is 127 pages.

The first chapter analyzed the relevance of the topic of online dating, conducted a study of the opinions and wishes of the target audience, analyzed existing software products, identified their strengths and weaknesses, developed a software product concept, analyzed software requirements, and justified the choice of implementation tools and technological processes used.

In the second section, we designed and prototyped the application screens on the Figma platform, developed a prototype of the client side to test it on a sample of the target audience, analyzed their feedback, developed the database architecture, created the server side, configured the CI/CD pipeline for continuous delivery of updates, used server settings, configured the Phusion Passenger system, and implemented the Progressive Web Application functionality.

The third section described in detail the interface and functionality of the created application and the built-in Admin Dashboard, as well as added information on system support and configured a cron job for automatic nightly database backup.

The conclusion contains an overview of the work performed, an assessment of the results of the actual implementation of the software, an analysis of statistics and user behavior, and a description of plans for further development of the system.

					III3.KP.Б – 121 – 23 – II3	Арк.
						6

Keywords: DATING, APPLICATION, VUE.JS, PINIA, VITE, PHUSION
PASSENGER, PUSH NOTIFICATIONS, WEB SOCKETS, NEST.JS,
TYPESCRIPT, PWA, ODM, MONGO DB, MONGOOSE, TYPEGOOSE,
KINDAGOOSE

					ПЗ.КР.Б – 121 – 23 – ПЗ	Арк.
						7

ЗМІСТ

ВСТУП.....	11
РОЗДІЛ 1 ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ	13
1.1 Аналіз особливостей вже існуючих на ринку додатків знайомств.....	13
1.1.1 Романтичні знайомства.....	13
1.1.2 Дружні знайомства.....	16
1.1.3 Порівняльна таблиця.....	21
1.2. Постановка задачі.....	24
1.3 Обґрунтування вибору засобів реалізації	26
Висновок до першого розділу	29
РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ	30
2.1 Проектування графічного інтерфейсу.....	30
2.1.1 Вибір кольорів та шрифтів.....	30
2.1.2 Розробка компонентів та ui-kit.....	31
2.1.3 Проектування дизайну екранів додатку.....	34
2.1.4 Прототипування та анімації	39
2.2 Розробка клієнтської частини	41
2.2.1 Локалізація	44
2.2.2 Анімації переходу між сторінками додатку	47
2.2.3 Pinch Zoom	49
2.2.4 Кадрування фотографій.....	51
2.2.5 Розпізнавання складних емоїї	54
2.3 Проектування архітектури бази даних.....	57
2.4 Розробка серверної частини	61

2.4.1 Авторизація.....	63
2.4.2 Дерево інтересів	66
2.4.3 Перевірка фотографій на пристойність за допомогою ІІІ	67
2.4.4 Сповіщення за допомогою Telegram боту	69
2.5 Створення Swagger документації	71
2.6 Адаптація клієнтської частини під роботу з сервером.....	72
2.6.1 Робота з Арі.....	72
2.6.2 Спільні типи для клієнта та сервера.....	75
2.7 Налаштування Web Sockets.....	77
2.8 Налаштування CI/CD	79
2.9 Налаштування серверу та бази даних	82
2.10 Налаштування кешування	83
2.11 Налаштування Phusion Passenger.....	85
2.12 Progressive Web Application.....	86
2.12.1 Web Manifest	87
2.12.2 Кешування.....	88
2.12.3 Власний плагін для Vite.....	89
2.12.4 Push Notifications	91
Висновок до другого розділу	94
РОЗДІЛ 3. ОГЛЯД ТА СУПРОВІД ГОТОВОГО ПРОДУКТУ	95
3.1 Огляд функціоналу готового додатку	95
3.2 Панель адміністрування	109
3.3 Аналітика поведінки користувачів за допомогою Amplitude.....	113
3.4 Відстеження помилок за допомогою Sentry	115
3.6 Резервне копіювання бази даних	116

Висновок до третього розділу	116
ВИСНОВКИ.....	117
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	121
ДОДАТКИ.....	126

ВСТУП

В наш час додатки для знайомств стали однією з основних можливостей знайомства з новими людьми у своєму житті, адже все більше людей, навчені пандемією COVID-19, сидять вдома та навіть для покупки продуктів користуються сервісами доставки. Все більше і більше людей не виходять за межі своєї бульбашки знайомих, хоча всі знають, що нові знайомства це не тільки нові враження, а ще й нові можливості. Саме тому, у останній час помітно значне зростання попиту на різні соціальні мережі та сайти для знайомств в режимі онлайн.

На даний момент існує безліч сайтів та додатків для знайомств, як романтичних, так і дружніх, проте я не знайшов жодного із них, які б могли вивчити особистість людини та підбирати схожих кандидатів і бути при цьому зручними та популярними. Саме тому я обрав це завдання для виконання кваліфікаційної роботи.

Головна мета дипломної роботи полягає в розробці та реалізації додатку знайомств, що буде базуватися на аналізі потреб користувачів та забезпечувати ефективний підбір сумісних кандидатів.

Завдання:

- Аналіз існуючих сайтів та додатків для знайомств з метою виявлення їхніх переваг та недоліків.
- Розробка додатку для онлайн знайомств та пошуку друзів

Об'єктом дослідження є процес знайомства та взаємодії людей в онлайн-середовищі, зокрема в контексті сайтів та додатків для знайомств.

Предметом дослідження є розробка PWA-додатку, що реалізує функціонал, необхідний для онлайн-знайомств.

Методи дослідження включають аналіз літературних джерел, існуючих сайтів та додатків для знайомств, розробку алгоритмів підбору людей,

проведення соціологічних досліджень, опитувань та статистичного аналізу для вивчення попиту.

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	Арк.
						12

РОЗДІЛ 1 ТЕОРЕТИЧНИЙ АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ТА ОСОБЛИВОСТЕЙ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз особливостей вже існуючих на ринку додатків знайомств

Перед тим, як почати проектування власного додатку, необхідно детально дослідити предметну область, переглянути програми-аналоги та скласти перелік можливостей, які повинен мати додаток.

1.1.1 Романтичні знайомства

Найбільш близькими до ідеї дипломного проекту є такі відомі мобільні додатки як Tinder, Badoo та Bumble, вони займають перші місця у категорії «Знайомства» у магазинах Google Play та Apple Store.

Розглянемо ці додатки та виділимо типові елементи. Типовий додаток для романтичних знайомств складається з трьох основних частин: власний профіль користувача, перегляд профілів інших користувачів та чати, розглянемо їх детальніше.

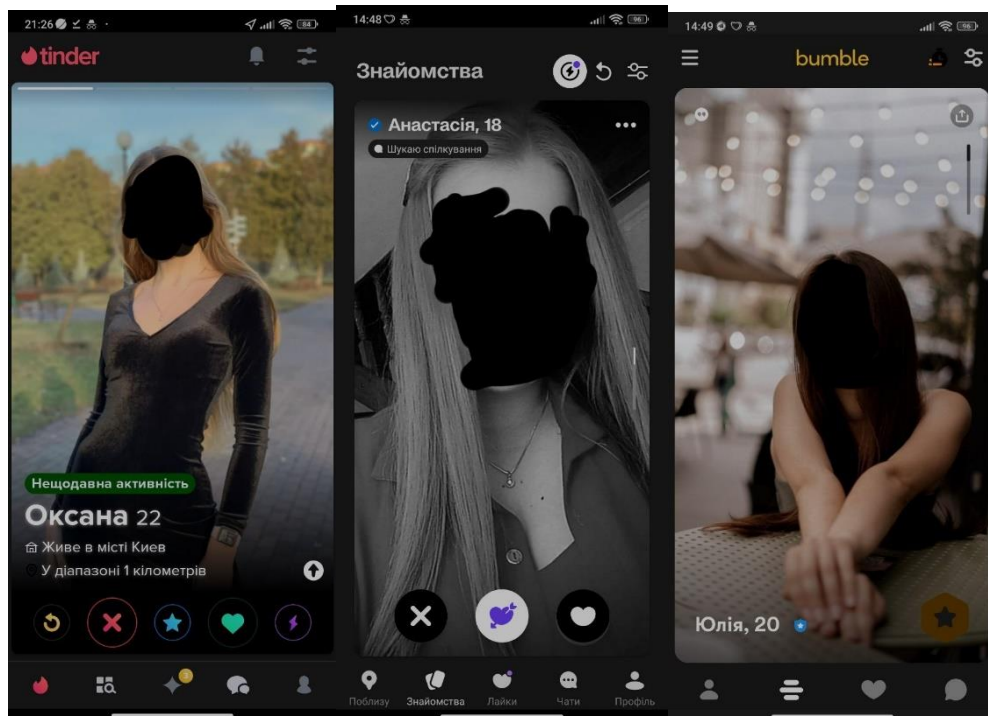


Рис. 1. Перегляд профілів користувачів у Tinder, Badoo та Bumble

Як зображено на Рис. 1, екран перегляду користувачів є найважливішою частиною додатку, адже саме на цьому екрані відбувається пошук людей. Він складається з трьох основних елементів: фотографії, інформація за анкети та кнопки дій, також, більшість додатків надають можливість відфільтрувати людей за статтю, віком, та іншими категоріями, проте, ці фільтри значно обмежені та у деяких додатках вимагають преміум підписки.

Кнопки дій у різних додатках різні, проте основними є лише дві: Like та Dislike. Після того, як обидва користувача вподобали профілі один одного, в них з'являється можливість почати діалог. Після натискання на одну з цих кнопок, користувач побачить наступний профіль. Також, в деяких додатках є можливість замість натискання на кнопки, робити Swіre анкети. У лівому-верхньому кутку, зазвичай, розташована кнопка відміни останньої дії, проте ця можливість теж вимагає преміум підписки.

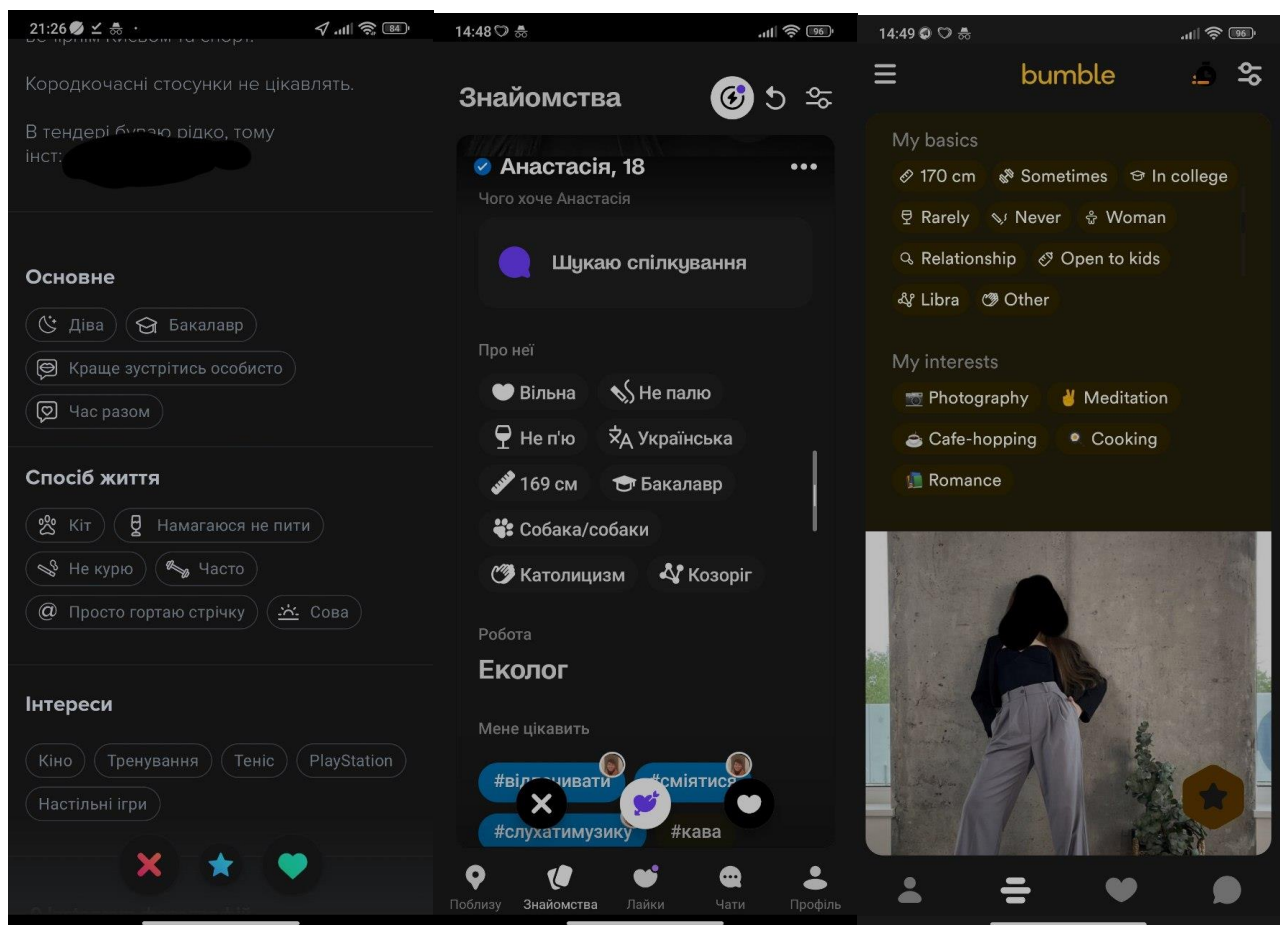


Рис. 2. Перегляд деталей профіля у Tinder, Badoo та Bumble

Під фото (Рис. 2), зазвичай, розміщується поле тексту «Про себе», основна інформація у вигляді тегів та інтереси. Інтереси зазвичай обмежені п'ятьма штуками та є дуже загальними.

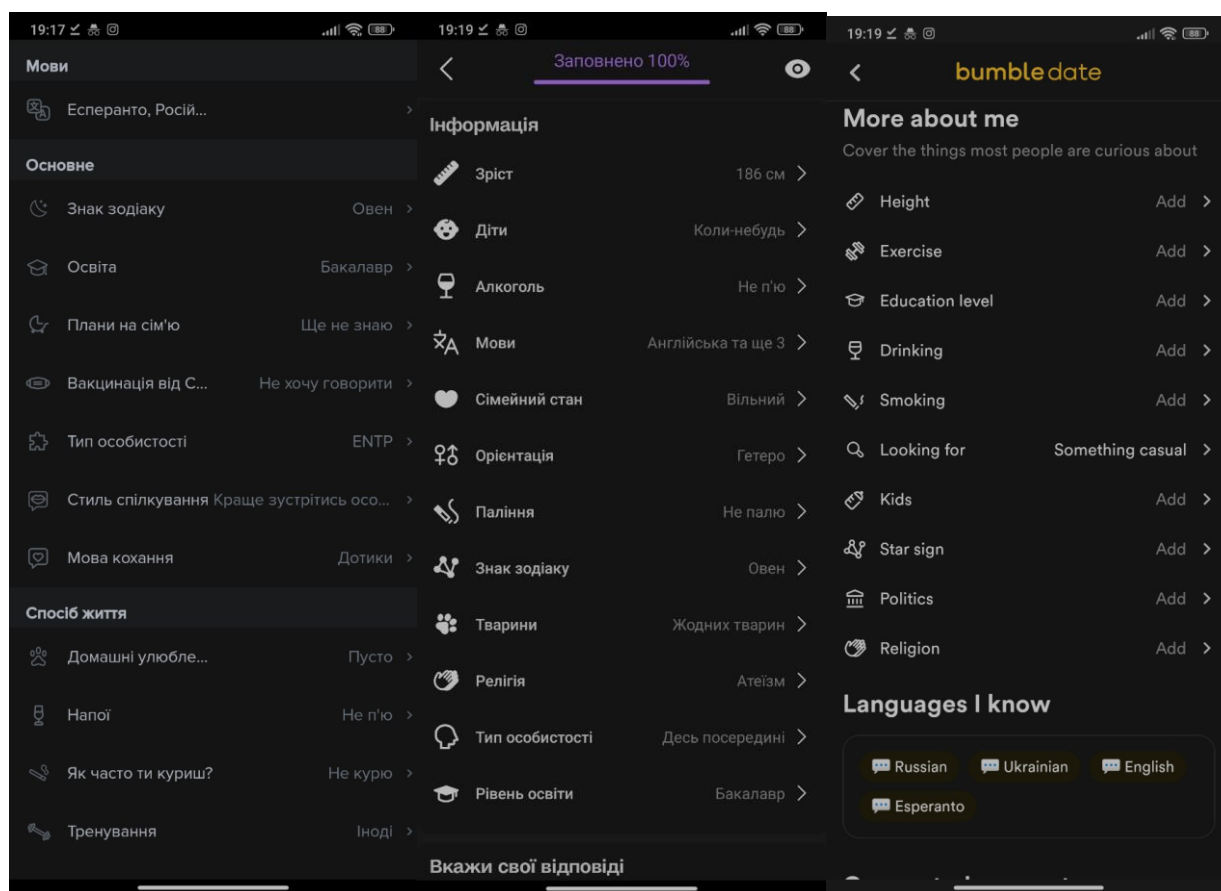


Рис. 3. Заповнення профілю у Tinder, Badoo та Bumble

На екрані профілю (Рис. 3) є можливість заповнити інформацію про себе та завантажити фотографії.

Зазвичай в усіх додатках запитується загальна інформація, така як орієнтація, сімейний стан, знання мов, зріст, відношення до паління та алкоголю, бажання мати дітей. Додаток Tinder пішов трохи далі та просить вказати «Тип особистості», проте не пояснює, що це таке та що з цим робити.

Деякі додатки (Badoo, Bumble), також, дають можливість відповісти на типові запитання, щоб показати своє відношення до життя. Проте, під час аналізу анкет на цих ресурсах було помічено, що ці поля у більшості випадків заповнюються типовими відповідями, що, майже повністю, знецінює їх.

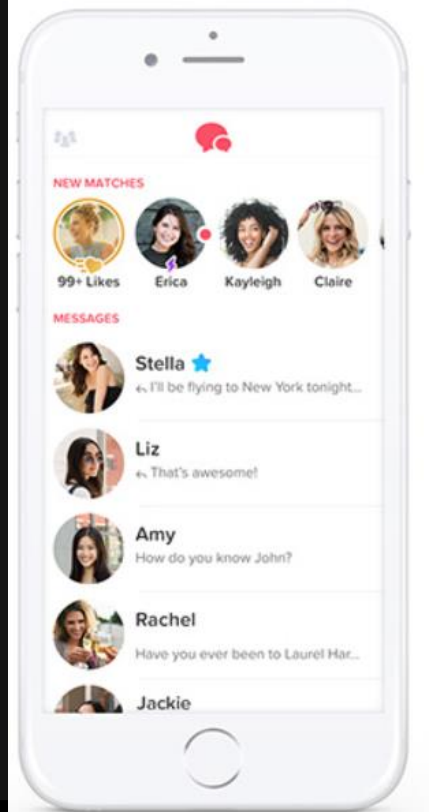
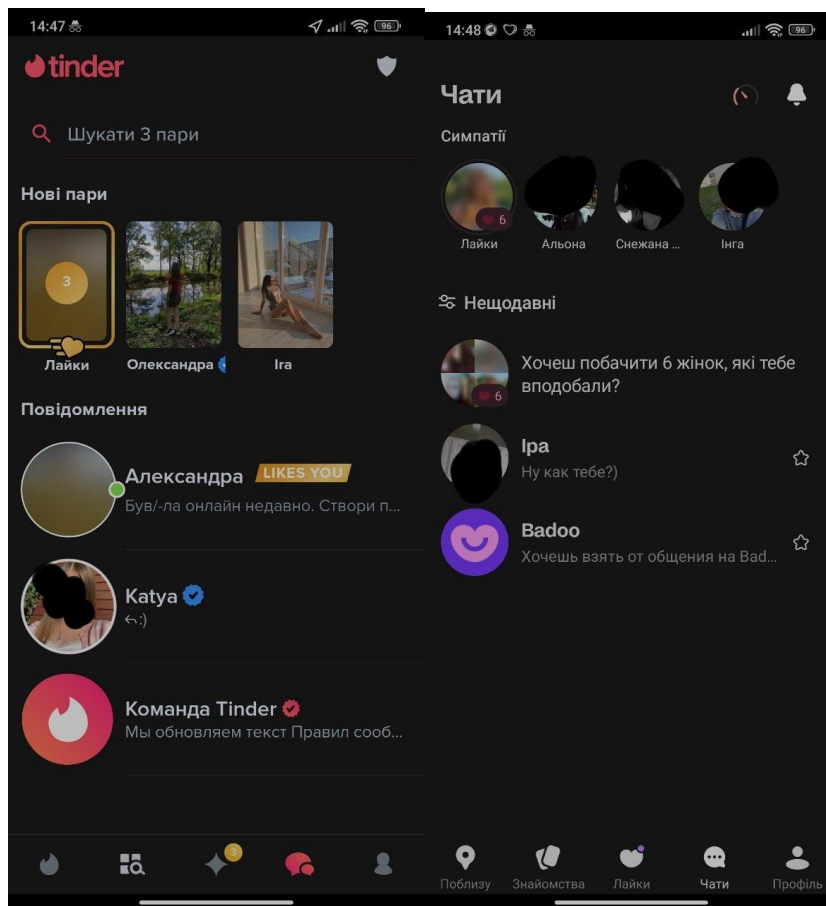


Рис. 4. Перелік чатів у Tinder, Badoo та невідомому додатку

На екрані чатів (Рис. 4) відбувається спілкування між людьми, а також відображаються невикористані симпатії (взаємне вподобання без створених діалогів). Самі чати, зазвичай, дуже обмежені за функціоналом та мають лише такі можливості як відправка повідомлень, додавання фотографій (не у всіх додатках), відправка голосових повідомлень (у деяких додатках) та відправка GIF зображень із сервісів типу Giphy та Tumblr). Відображення, чи прочитав партнер повідомлення користувача, зазвичай, потребує преміум підписки.

1.1.2 Дружні знайомства

У сфері дружніх знайомств можна розглянути такі додатки як Bumble (у BFF режимі), Yubo, Boo та бот Дайвінчик. Розглянемо їх більш детально за зростанням можливостей.

Дайвінчик. Цей Telegram бот являє собою щось на кшталт публічної дошки оголошень, де будь-яка людина може створити анкету, вказавши ім'я, вік, місто

проживання, фотографію (до 3-х штук) та, за бажанням, прикріпити коротенький текст. Потім у користувача є можливість переглядати анкети інших людей, виконуючи з ними такі дії як «Вподобання», «Повідомлення» та «Пропуск» (Рис. 5). Із недоліків можна зазначити те, що дуже мало людей вказують супровідний текст, а також, величезну кількість підроблених анкет, тому, щоб знайти цікаву людину потрібно перегорнути велику кількість анкет. Крім того, при виставленні кожного вподобання, користувачу, якому його поставили, приходить повідомлення та вподобання одному користувачу відправляти можна необмежену кількість разів.



Рис. 5. Приклад анкети у боті Дайвінчик

Bumble BFF. Це режим дружніх знайомств у додатку Bumble. Єдиною відмінністю від звичайного режиму Date є те, що показуються лише люди своєї статі. Також, відсутнє звичайне обмеження на перше повідомлення (у Bumble Date перший крок можуть робити тільки дівчата). Одним із головних недоліків цього режиму є те, що кількість людей у BFF режимі у два рази менша, адже показуються люди лише такої самої статі, як і стать користувача, проте це є проблемою, адже дружба можлива і серед представників різних статей.

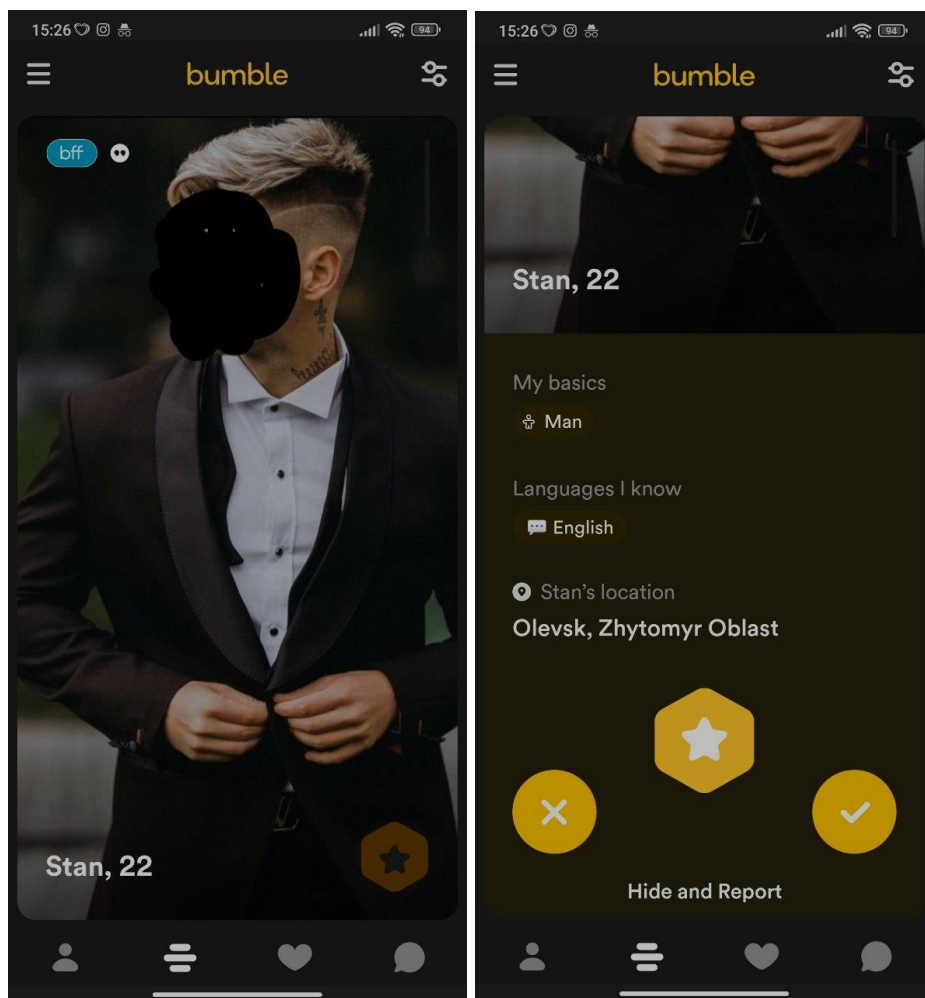


Рис. 6. Стрічка у додатку Bumble BFF

Наступними проаналізованими додатками були додатки Yubo та Boo. Додаток Yubo позиціонується як додаток для пошуку друзів за спільними інтересами та інтересів там дійсно багато (декілька сотень), вони організовані у вигляді тегів та розбиті по категоріях з можливістю пошуку. У такій організації інтересів є як свої плюси, так і мінуси. До плюсів відноситься зручність пошуку тегів у випадку, якщо користувач знає, що саме шукає. Проте, за результатами спостережень, у наш час таке трапляється не часто, та з цього впливає недолік, а саме, складність заповнення через надто велику кількість тегів. Також, у додатку Yubo помічено серйозні проблеми із пошуком людей поблизу, адже часто показуються люди з інших куточків світу та, навіть, ті, що розмовляють на інших мовах, проте, мають спільні інтереси з користувачем.

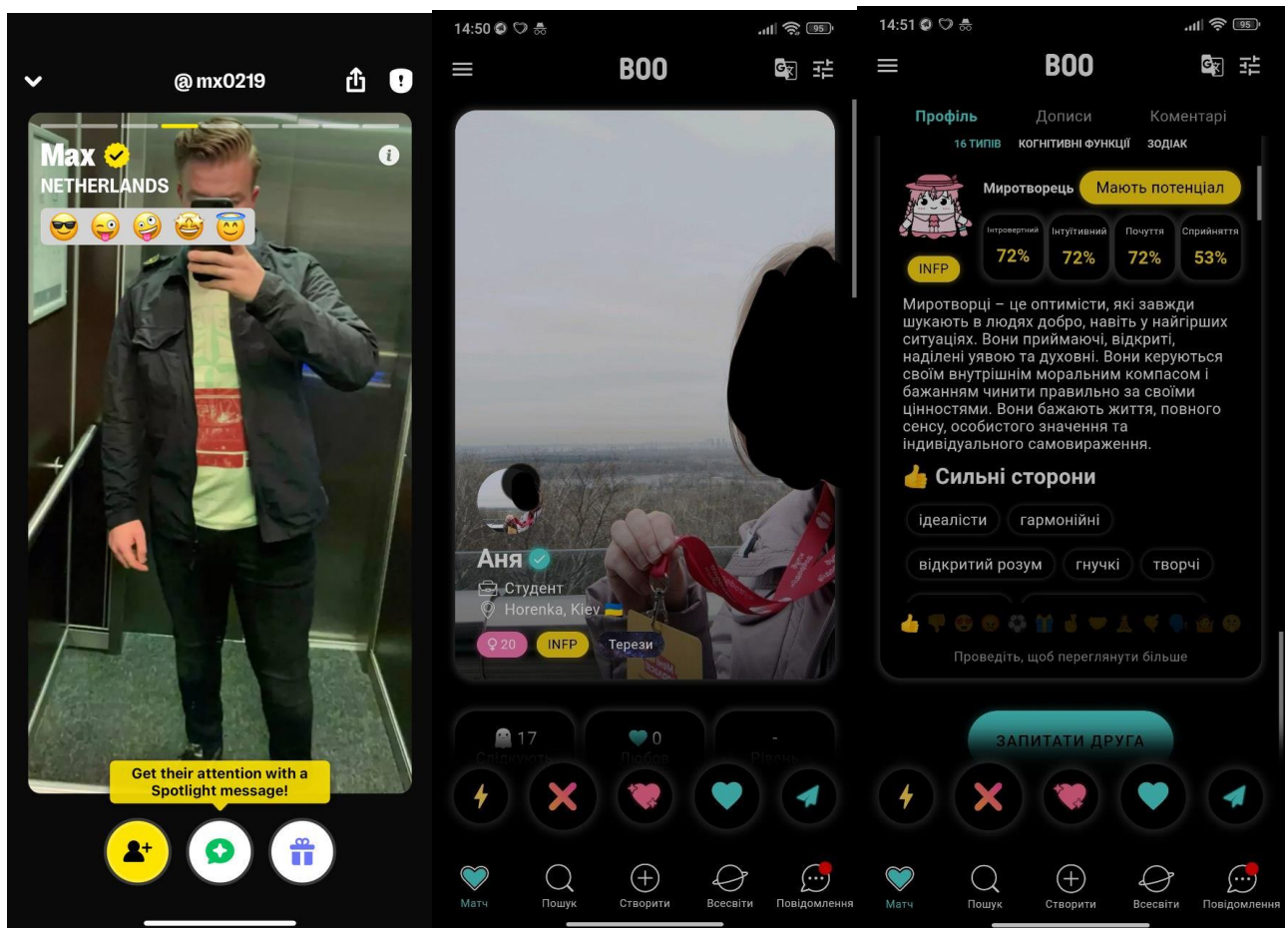


Рис. 7. Стрічка у додатках Yubo (1) та Boo (2, 3)

Додаток Boo декларує себе як додаток для пошуку кохання та друзів. Цей проект за концепцією найбільш схожий на тему даної кваліфікаційної роботи, проте має і відмінності.

Перша відмінність Boo (на відміну від Bumble) у користувача є всього один профіль, де він вказує, що хоче знайти (кохання, друзів або обидва варіанти). В залежності від цього, його анкети буде показуватися людям із такими самими налаштуваннями, проте у такій схемі є значні недоліки. По-перше, відображаються лише люди із такими самими налаштуваннями пошуку (наприклад, якщо користувач А шукає тільки друзів, а користувач Б шукає і кохання, і друзів – вони не будуть показані один одному). По-друге, для пошуку кохання та друзів часто потрібно використовувати різні фотографії та опис анкети, показувати себе з різних сторін, а при використанні одного профілю для двох режимів це зробити неможливо.

Друга відмінність полягає у тому, що Boo це перший додаток, що просуває поняття сумісності. Проте, сумісність вираховується лише за психотипом по системі MBTI [1], не враховуючи інші фактори.

Також, додаток надає детальні рекомендації щодо психотипу людей, гарно перекладені українською мовою, проте ця інформація більше схожа на гороскоп та скоріше відволікає, аніж допомагає.

Щодо алгоритмів підбору у додатку Boo також є питання, адже не зовсім зрозумілі критерії підбору. У самому додатку часто говориться про сумісність, проте, люди, що відображаються у стрічці часто мають «не сумісні» психотипи та чергуються з людьми із «сумісними» психотипами.

Також, дивну поведінку має пошук людей із того самого міста. Адже при виборі локального пошуку (тільки люди з того самого міста) у стрічці можуть відображатися люди із сусідніх регіонів, при цьому, в хаотичному порядку з'являються і люди з Житомира.

Також, слід зазначити незвичайну систему внутрішньої валюти додатку, яку, на відміну від інших додатків, можна заробити, виконуючи різні корисні дії (наприклад, заходити кожен день, заповнити профіль на 100%, поставити хорошу оцінку додатку та ін.). Це цікавий хід, який може значно підвищити активність користувачів у додатку. Також, цю валюту можна купити за реальні гроші.

В цілому, додатки для дружніх знайомств намагаються підбирати людей за системою тегів, які додають самі користувачі. У деяких додатках (наприклад, Boo), для додавання нового тегу потрібно пройти модерацію. Також, було помічено різні способи організації тегів: з локалізацією та без. У випадку тегів з локалізацією (Yubo), для однієї сутності існує лише один тег, що перекладається на різні мови, проте у іншому випадку (Boo), теги «Музика» та «Music» це різні теги, що призводить до значного погіршення результатів підбору людей.

1.1.3 Порівняльна таблиця

Таблиця 1

Порівняльна таблиця існуючих додатків

Сервіс	Tinder	Badoo	Bumble	Дайвінчик	Boo
Знайомства	Романтичні	Романтичні	Романтичні + Дружні	?	Романтичні + Дружні
Формат	Додаток + PWA	Додаток + Сайт	Додаток	Telegram Бот	Додаток
Інтереси	1	1	1	0	3
Психотип	1	0	0	0	4
Обмеження	2	3	1	5	4
Фільтр	1	3	3 (1)	0	4
Сумісність	0	1	0	0	3
Популярність в Україні	5	5	3	5	2

На таблиці 1 зображено порівняння раніше проаналізованих додатків за деякими найбільш важливими критеріями, розберемо їх детальніше:

- **Знайомства:** Режими знайомств, які підтримує даний сервіс. Для бота Дайвінчик стоїть знак питання, адже цей сервіс не має конкретного режиму та люди самі вказують (або не вказують), кого вони шукають.
- **Формат:** У якому вигляді надається сервіс. Під словом «додаток» маються на увазі окремі додатки для Android та iOS.
- **Інтереси:** Можливості пошуку людей за інтересами, оцінюється по шкалі від 0 до 5. Під оцінкою 0 розуміється повна відсутність роботи з інтересами. Під оцінкою 1 розуміється можливість вказати до 5 загальних

інтересів із списку без можливості пошуку або фільтрації людей по ним. Під оцінкою 3 розуміється можливість завдання необмеженої кількості інтересів у вигляді тегів та можливість пошуку людей за ними. Під оцінкою 5 розуміється зручне та детальне завдання інтересів, пошук або фільтрація людей за ними та використання інтересів людини для автоматичного підбору людей зі схожими інтересами.

- **Психотип:** Підтримка роботи з психотипами (по системі MBTI [1] або іншій). Під оцінкою 0 розуміється відсутність можливості вказати психотип (класифікація інтроверсія/екстраверсія за психотип не вважаються). Під оцінкою 1 розуміється можливість вказати психотип, проте, відсутність будь-якої взаємодії з цим полем (пошук, фільтрація, порівняння). Під оцінкою 4 розуміється усі попередні вимоги плюс пошук або фільтрація людей за психотипом, розрахунок сумісності та можливість пройти тест. Під оцінкою 5 розуміється виконання усіх попередніх вимог плюс використання сумісності за психотипом для автоматичного підбору сумісних людей.
- **Обмеження:** Оцінка, наскільки обмежені можливості у користувачів без преміум підписки, чим вища оцінка, тим обмеження менші. Під оцінкою 5 розуміється відсутність обмежень функціоналу, платна підписка надає лише косметичні або додаткові можливості. Під оцінкою 4 розуміється обмеження деяких не основних можливостей з використанням внутрішньої валюти, яку, у свою чергу, можна отримати без вкладання валюти реальної. Під оцінкою 3 розуміється часткове обмеження можливостей, таких як перегляд людей, яким користувач вже подобається, перегляд, чи людина вже прочитала повідомлення та часткове обмеження використання фільтрів. Оцінка 2 аналогічна оцінці 3, проте додаток не має додаткових фільтрів, то ж його неможливо віднести ані до оцінки 3, ані до оцінки 1. Оцінка 1 включає в себе усі попередні обмеження та використання додаткових фільтрів можливе лише за платною підпискою.

- **Фільтр:** Можливості фільтрації людей за вказаними критеріями. Оцінка 0 вказує на відсутність будь-якої фільтрації або сортування, люди просто показуються у випадковому порідку. Під оцінкою 1 розуміється наявність базових фільтрів за віком та статтю. Під оцінкою 3 розуміється можливість фільтрації за деякими полями профілю, проте ці поля мало деталізовані. У додатку Vumble в дужках вказана оцінка 1, адже без преміум підписки детальний фільтр недоступний. Оцінка 4 аналогічна оцінці 3, проте, полів більше та вони більш деталізовані. Оцінка 5, аналогічна оцінці 4, але також включає в себе також жорстку фільтрацію за конкретною локацією.
- **Сумісність:** Чи є у додатку розрахунок сумісності з показаною людиною. Оцінка 0 вказує на повну відсутність такого функціоналу. Під оцінкою 1 розуміються мінімальні прояви оцінки сумісності, наприклад, у додатку Badoo, якщо є, хоча б один, спільний інтерес пропонується відправити «супер-вподобання». Під оцінкою 3 розуміється розрахунок сумісності або за спільними цінностями, або за психотипом, або за інтересами, проте, ця інформація не використовується для сортування людей, що відображаються. Під оцінкою 5 розуміється розрахунок сумісності як за цінностями та психотипом, так і за інтересами та використання цих характеристик для відображення найбільш сумісних людей.
- **Популярність в Україні:** Суб'єктивна оцінка популярності в Україні сервісу від 1 до 5 на основі власного досвіду, усних опитувань, оцінки кількості згадувань у популярних статтях та списках кращих додатків українською мовою, та кількості коментарів у Google Play від людей із України.

Згідно таблиці 1, найбільш слабкими місцями у найпопулярніших додатках для знайомств є робота з інтересами, психотипом та персоналізованим розрахунком сумісності. Саме тому, при створенні власного додатку для знайомств було прийняте рішення акцентувати увагу саме на ці можливості.

1.2. Постановка задачі

Після детального аналізу існуючих додатків та опитування їх користувачів, було сформовано список можливостей, що потрібно реалізувати для того, щоб додаток знайомств був повноцінним та функціональним.

Перш за все, це основні частини: заповнення профіля, пошук людей з можливістю фільтрації та чати. Побачивши значний недолік у існуючих додатках стосовно аналізу особистості та підбору сумісних партнерів (анкети у існуючих додатках, зазвичай, показуються за своєю популярністю та виводять у перші рядки саме преміум-користувачів, а не підбирають сумісних людей для конкретного користувача), було вирішено основну увагу приділити саме детальному заповненню профіля та алгоритму розрахунку сумісності.

Пошук романтичних партнерів та друзів було вирішено повністю розділити на окремі режими: Романтичний та Дружній. Кожен користувач матиме окремі профілі для цих режимів, окремі симпатії та окремі діалоги. Також, користувач за бажанням може відключити один з режимів, або, навіть, видалити один з профілів, наприклад, якщо вже знайшов романтичну пару та все ще шукає друзів. Також, при реєстрації планується додати можливість обрати, що саме (пошук романтичного партнера, друзів, чи обидва варіанти) шукає користувач та створювати профілі саме для обраних режимів. Також, для чіткого візуального розділення, у якому режимі зараз знаходиться користувач, було вирішено розробити для кожного з режимів свою кольорову схему та перемикати їх, в залежності від режиму додатку.

Проаналізувавши існуючі рішення, було вирішено перейняти систему симпатій у бота Дайвінчик із деякими модифікаціями.

По-перше, кількість вподобань на день обмежена. Це додасть їм цінності та люди намагатимуться детальніше вивчати профілі перед тим як поставити вподобання.

По-друге, користувачі будуть бачити у анкеті, що цей користувач їх вже вподобав та вподобання у відповідь буде безкоштовним. Це спонукатиме людей частіше ставити взаємні вподобання, навіть коли, на перший погляд, людина здається не дуже цікавою (йде розрахунок на те, що алгоритми додатку можуть підібрати людей краще, ніж користувач може сподіватися).

По-третє, якщо користувач пропускає деякий профіль, він побачить його знову через деякий час, але якщо ставить вподобання, цей профіль більше не буде йому показуватися. Цей механізм враховує таку особливість психіки, як завищенні очікування напочатку та їх поступове спадання до реальності, а також, бажання людини спочатку вивчити існуючі варіанти, а лише потім обирати. Якщо ж користувач точно впевнений, що йому не подобається ця людина, було вирішено додати кнопку «Не показувати більше» у самий кінець профілю, яка автоматично додає цього користувача у чорний список.

Також, було вирішено перейняти модель реферальної системи, коли додаткові вподобання можна отримати лише шляхом запрошення нових користувачів.

Щодо місцезнаходження, було прийняте рішення не використовувати GPS, а надати людині можливість обрати своє поточне місцезнаходження із списку. Окрім можливих проблем із виявленням місцеположення (такі є у додатку Bumble, коли при наявності людей з Житомира, все одно постійно відображаються люди з Києва та це неможливо ніяк змінити, адже вибір місцезнаходження вручну доступний лише у преміум підписці), це вирішить ще і життєві ситуації, коли людина готується до переїзду та вже може почати шукати друзів у новій локації. Також, було вирішено розділити місце народження та місце проживання, адже походження із одної країни за кордоном може бути додатковим фактором, що зміцнює зв'язки.

На рахунок чатів, на даному етапі було вирішено реалізувати лише текстовий функціонал, на це існує декілька причин. По-перше, на відміну від анкети, де кількість зображень обмежена, у приватних повідомленнях

користувачі можуть відправляти сотні зображень на день, які при цьому потрібно зберігати. По-друге, можливість відправляти фотографії у приватних повідомленнях значно ускладнює модерацію. У великих компаніях існують окремі відділи, що обробляють скарги користувачів на непристойні фотографії у приватних повідомленнях. По-третє, в цьому немає необхідності, адже мета додатку – знайти максимально сумісних людей. Для подальшого спілкування краще підходять різноманітні вже існуючі месенджери. Саме тому було прийняте рішення обмежитися лише текстовими повідомленнями із можливістю додавання голосових записів у майбутньому, адже голос людини це вагома складова сприйняття образу.

За переліченими вимогами було створено діаграму варіантів використання, її знайти можна у додатку А на Рис. А.1.

1.3 Обґрунтування вибору засобів реалізації

Для розробки системи було прийняте рішення використовувати найновітніші технології та практики, це необхідно не тільки для навчання та оновлення своїх знань, а ще і для максимального прискорення як процесу розробки, так і швидкодії готового продукту.

1.3.1 Клієнтська частина

Для створення мобільного додатку було обрано технологію Progressive Web Application (PWA) [2], адже вона дає змогу розробляти одночасно сайт, десктопний додаток, та мобільний додаток для всіх сучасних платформ. Використання PWA є допустимим адже постановка задачі не вимагає додаткових дозволів для більш глибокої взаємодії із цільовою платформою.

У якості UI-бібліотеки, для створення додатку було обрано Vue.js версії 3.2.45 [3]. Для маршрутизації використовується пакет Vue Router [4]. Для управління станом додатку – бібліотека Pinia [5] (сучасна заміна Vuex).

Замість об'єктно-орієнтованого підходу до створення компонентів, який був загальноприйнятим у Vue 2, було обрано новітній підхід, що з'явився у Vue

3, а саме, Composition API [6], що дає змогу створювати логіку компонентів, використовуючи функціональний підхід, а також значно покращує швидкодію кінцевого продукту через уникнення роботи з повільною системою класів JavaScript.

Для побудови графічного інтерфейсу не було обрано жодного фреймворку, адже під час виконання даної роботи, буде розроблено власний фреймворк на основі дизайн системи та компонентів, що буде описано у розділі 2.1 цього звіту.

У якості системи збирання, замість WebPack, що є загальноприйнятим стандартом веб-розробці, було обрано новітній збиральник Vite [7], що підтримує живу заміну модулів, zero-conf підключення плагінів та, на відміну від WebPack, запускається всього за декілька секунд, навіть на дуже великих проектах.

Для спрощення написання таблиць стилів, було обрано препроцесор SASS [8]. При використанні Vite ніякої додаткової конфігурації не потребується, достатньо просто встановити пакет sass.

1.3.2 Серверна частина

Для організації серверної частини додатку було обрано сучасний framework – Nest.js [9], адже він надає можливості для швидкого, зручного та надійного проектування REST API.

Підхід REST API було обрано через його простоту, зручність, та можливості легкого масштабування. Більш розширені можливості, які надає GraphQL [10] були б надмірними для даної системи, адже, замість спрощення, навпаки, ускладнили би її.

У якості бази даних на початку планувалося обрати PostgreSQL [11], адже для більшої частини системи добре підходить реляційна модель, а PostgreSQL, також, підтримує можливості працювати із форматом JSON. Проте, після більш детального вивчення PostgreSQL, можливостей її інтеграції із Nest.js та створення прототипу системи було прийняте рішення відмовитися від

PostgreSQL на користь MongoDB [12]. На це рішення вплинуло декілька факторів. По-перше, необхідна нормалізація даних значно ускладнювала систему. На етапі проектування прототипу бази даних було виділено більше 25 таблиць, що значно ускладнювало подальший розвиток та підтримку такої системи. По-друге, можливості PostgreSQL по роботі з JSONB та JsonPath [13] значно обмежені та ускладнені, адже мова запитів SQL історично не була спрямована на підтримку роботи із структурами типу JSON, тому виконання подібних запитів більш схоже на милиці, ніж на хороше рішення. По-третє, на етапі прототипування виникли проблеми із інтеграцією з Nest.js. На момент написання роботи, на ринку існувало декілька ORM, що підтримують Nest.js, проте, всі вони мають значні недоліки [14] [15], що змусило відмовитися від їх використання у даній кваліфікаційній роботі. Розглядалися такі ORM як TypeORM, Sequelize та Prisma.

Проблеми з використанням SQL баз даних у даній кваліфікаційній роботі навело на думку щодо використання NoSQL підходу, а саме, найпопулярнішої NoSQL бази даних – MongoDB. Виявилося, що використання такого підходу значно спрощує архітектуру бази даних та, при використанні ODM, що забезпечує валідацію даних та перевірку зв'язків, значно спрощує подальший розвиток системи та її підтримку.

У якості ODM було обрано Mongoose, який зарекомендував себе як найкраща ODM для MongoDB. Для покращеної підтримки типів та використання декораторів для декларативного оголошення полів схеми було обрано бібліотеку Typegoose [16]. Для інтеграції Typegoose із Nest.js спочатку розглядався стандартний пакет @nestjs/mongoose [17], проте, під час розробки прототипу із ним виникли проблеми, тому було обрано замінити його на неофіційний та непопулярний fork від користувача GrapeoffJS під назвою Kindagoose [18].

Для API-тестування та створення документації було обрано бібліотеку Swagger, що має гарну інтеграцію з Nest.js та дозволяє виконувати описання API запитів за допомогою зручних декораторів. Більш детально використання та налагодження бібліотеки Swagger [19] буде описано у розділі 2.5 цієї записки.

1.3.3 Інші інструменти та технології

Для клієнт-серверного спілкування у реальному часі було обрано технологію Web Sockets, а саме, бібліотеку Socket.io [20].

Для перевірки та стандартизації форматування коду використовуються ESLint та Prettier.

У якості IDE використовується Microsoft Visual Studio Code (VSCode) із встановленим розширенням GitLens для зручної роботи з Git та розширенням GitHub Copilot [21] для спрощення рутинних задач.

У якості IDE для використання на сервері використовується NeoVim з конфігурацією AstronVim, використанням LSP серверів для TypeScript, JavaScript, JSON, Yaml та PHP, а також, розширенням GitHub Copilot, аналогічно до VSCode. Взаємодія виконується через підключення віддаленого терміналу за допомогою протоколу SSH.

У якості пакетного менеджера – Yarn.

У якості системи контролю версій – Git.

У якості платформи збереження репозиторію – GitLab.

Для іменування комітів використовується Conventional Commits.

Також, для тестування, перевірки на вразливості та розгортання було обрано GitLab CI.

Для контролю процесу виконання було використано онлайн сервіс Jira від Atlassian у режимі Kanban дошки.

Для створення дизайну та прототипу додатку було обрано онлайн-сервіс Figma, що є сучасним стандартом при роботі з веб дизайном.

Висновок до першого розділу

Оцінивши актуальність онлайн знайомств, проаналізувавши існуючі додатки та обравши засоби реалізації додатку на тему кваліфікаційної роботи, можна приступати до самого проектування та розробки додатку, чому буде присвячено наступний розділ пояснювальної записки.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ

2.1 Проектування графічного інтерфейсу

2.1.1 Вибір кольорів та шрифтів

Перед початком розробки дизайну, потрібно обрати палітру кольорів (Рис. 8), шрифтів (Рис. 9) та компонентів (Рис. 10, Рис. 11), з яких цей дизайн буде створюватись – це називається дизайн-системою.



Рис. 8. Основні кольори додатку

Для романтичної кольорової схеми додатку було обрано поєднання помаранчевого та чорного, з вкрапленнями інших кольорів для деяких елементів, а основною темою додатку буде темна.

Дружня тема поєднуватиме фіолетові та сині відтінки для того, щоб контрастувати з романтичною.

Для стилізації іконок було обрано яскраві неонові кольори, що забезпечують високу контрастність та швидке сприйняття.

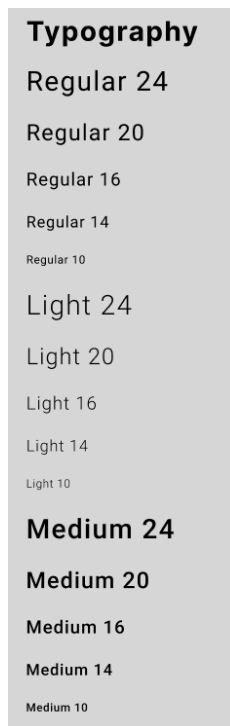


Рис. 9. Основні шрифти

Для текстів було обрано шрифт Roboto у варіантах Regular, Light та Medium. Розміри 24pt, 20pt, 16pt, 14pt та 10pt.

Ці кольори та набори шрифтів були збережені у вигляді Figma стилів та згодом будуть оголошені у вигляді CSS змінних [22].

2.1.2 Розробка компонентів та ui-kit

Наступним кроком слідує розробка основних елементів, які будуть часто використовуватися у додатку, а саме, стилізація деяких звичайних HTML елементів, таких як, кнопки та прапорці.

На Рис. 10 зображені деякі з цих компонентів, проте, у кінцевому результаті лише кнопок було створено 10 видів. Усі елементи оформлені у вигляді Figma компонентів, що значно спростить наступне прототипування та надасть можливість взаємодіяти з цими компонентами у самій програмі Figma. При додаванні компонентів на екрани у налаштуваннях вказується, який саме стан компонент має мати, також, ці стани можуть змінюватися динамічно, наприклад, коли користувач натискає клавішу миші на компоненті.

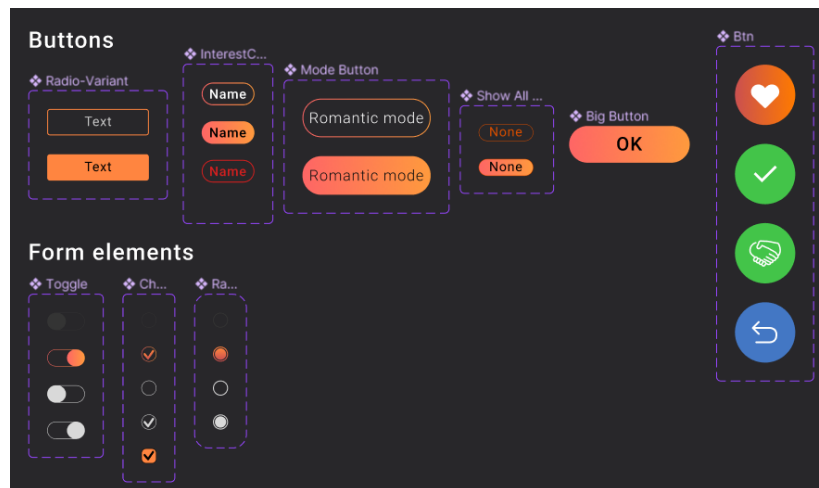


Рис. 10. Основні елементи

Після створення елементів, потрібно розробити шаблонні компоненти системи, а саме, типові елементи меню та форм, з яких потім будуватиметься інтерфейс додатку.

Додаток проектується для мобільних пристроїв, тому основною шириною компонентів було обрано 360px.

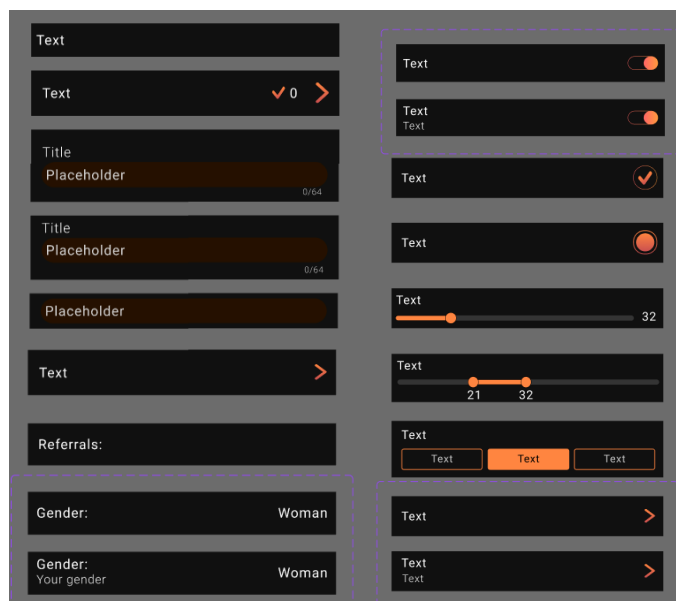


Рис. 11. Основні компоненти системи

Також, у профілі користувачів планується зробити брифінг-секцію, один погляд на яку повинен давати максимально повну інформацію про людину. Це є альтернативним варіантом зображення інформації на відміну від вигляду тегів, що використовуються у популярних додатках знайомств.

З цією метою було розроблено іконки, що відображають певну характеристику людини, їх зображено на Рис. 12.

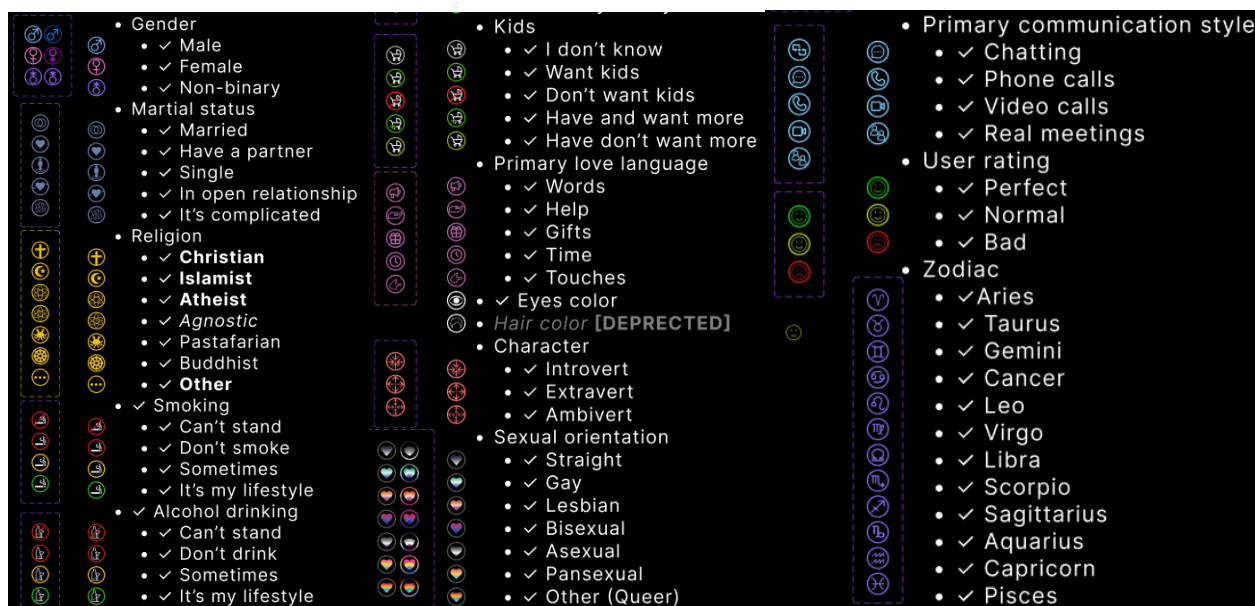


Рис. 12. Іконки для додатку

Кожна іконка виконана у SVG форматі та використовує стандартні кольори для їх чіткої класифікації. Окрім невеликого об'єму зображення та високої чіткості, використання SVG дає можливість динамічно змінювати колір, розмір та інші характеристики окремих частин, що гарним чином впливає на оптимізацію компонентів, що будуть їх відображати у майбутньому.

Хоча використання окремих іконок для окремих станів характеристик профіля може здаватися складним та недоцільним, дослідження показало, що через невеликий проміжок часу (зазвичай один-два дні), користувач звикає до цих іконок та починає орієнтуватися у них значно швидше, ніж у звичайних тегах, що використовуються у сучасних додатках для знайомств. Це призводить до того, що швидкого погляду на брифінг-секцію достатньо для того, щоб скласти уяву про анкету на екрані. Через тиждень користувачі, на яких проводилося тестування іконок, говорили, що їм зручніше користуватися вище створеними іконками, аніж тегами інших додатків знайомств, розглянутих у розділі 1.1.

2.1.3 Проектування дизайну екранів додатку

Першим для розробки було обрано екран профілю (Рис. 13), він же є головною сторінкою та повинен відображати основну інформацію для користувача.

На ньому розташувалися показники активності, аватар користувача, та головне меню, з якого можна зайти у налаштування, переглянути свої інтереси, отримати допомогу або вийти з акаунту.

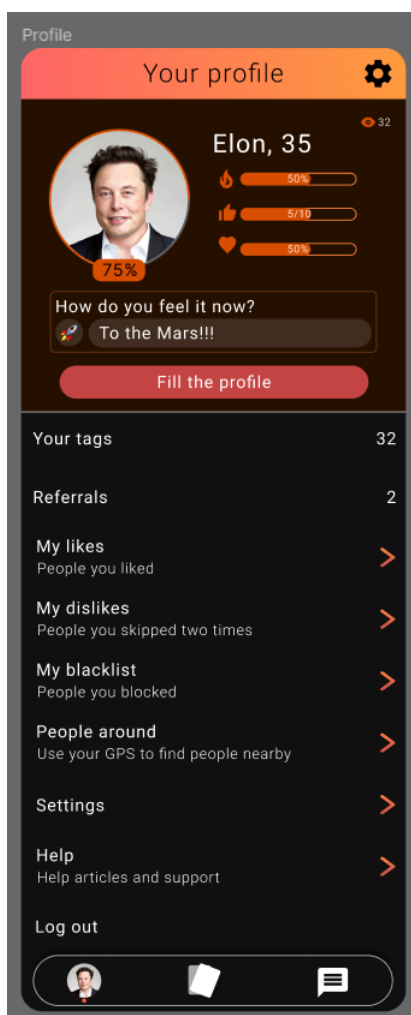


Рис. 13. Дизайн головної сторінки

Наступною сторінкою для розробки була сторінка пошуку людей. Вона повинна включати в себе фотографію, брифінг-секцію з іконками, секцію дерева інтересів людини з можливістю відображати всі інтереси, лише спільні та приховати цю секцію, а також інша інформація, що не може бути виражена ані

іконками, ані деревом інтересів. Крім того, якщо ця людина вже виразила симпатію користувачеві, біля фото повинен бути зображений значок вподобання та прикріплене повідомлення, якщо ця людина його залишила.

Проаналізувавши досвід існуючих додатків для знайомств, було вирішено зробити три кнопки дій: вподобання, пропуск та вподобання із прикріпленням повідомлення. Остання кнопка даватиме змогу прикріпити повідомлення до вподобання, та яке, у випадку взаємної симпатії, стане першим повідомленням у діалозі. Цей механізм вирішує проблему коли при взаємній симпатії обидві сторони чекають, що інша людина зробить перший крок. За результатами усного опитування, це доволі часта проблема під час онлайн знайомств.

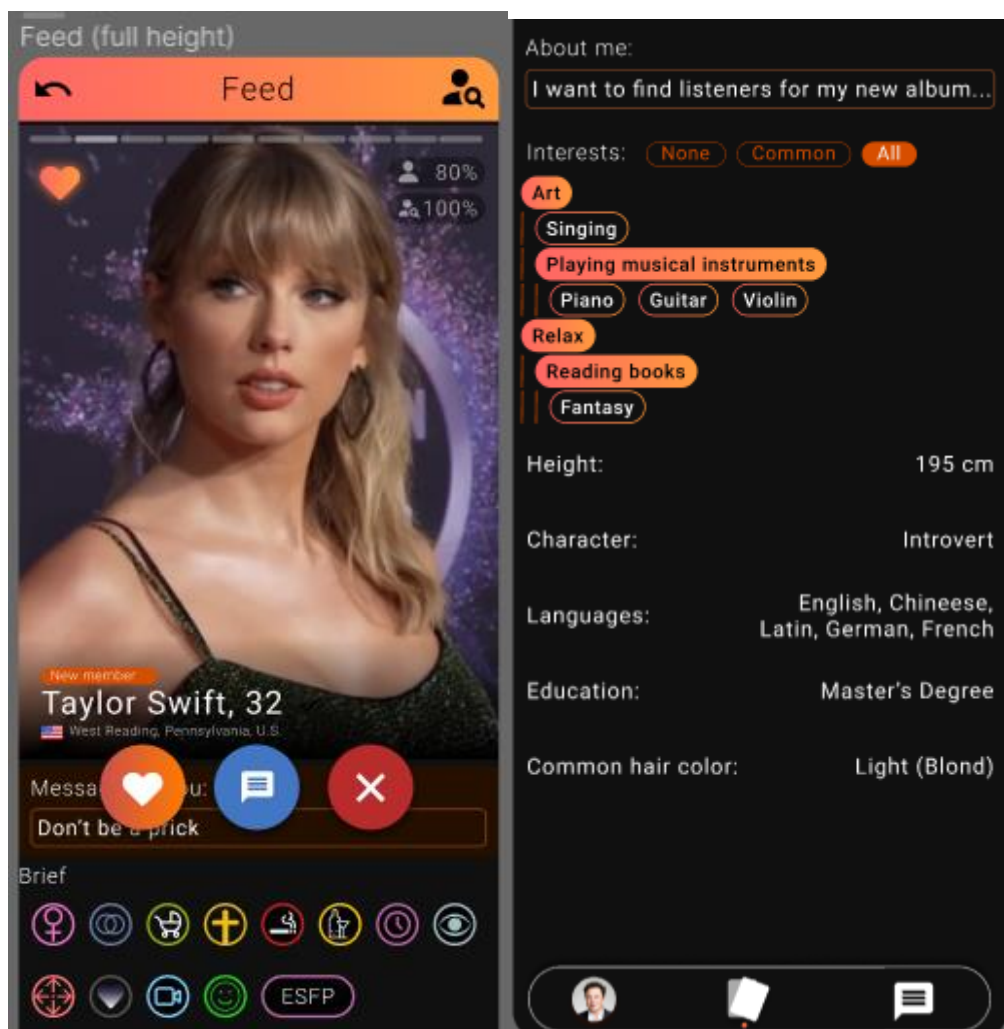


Рис. 14. Дизайн сторінки пошуку людей

Наступною сторінкою для розробки була сторінка діалогів. На основній сторінці повинні розташовуватися діалоги у вигляді списку з позначенням, чия черга зараз відповідати. Також, зверху повинна розташовуватися панель невикористаних симпатій – там зберігаються люди, з якими у користувача були взаємні симпатії, але ніхто ще не написав перше повідомлення.

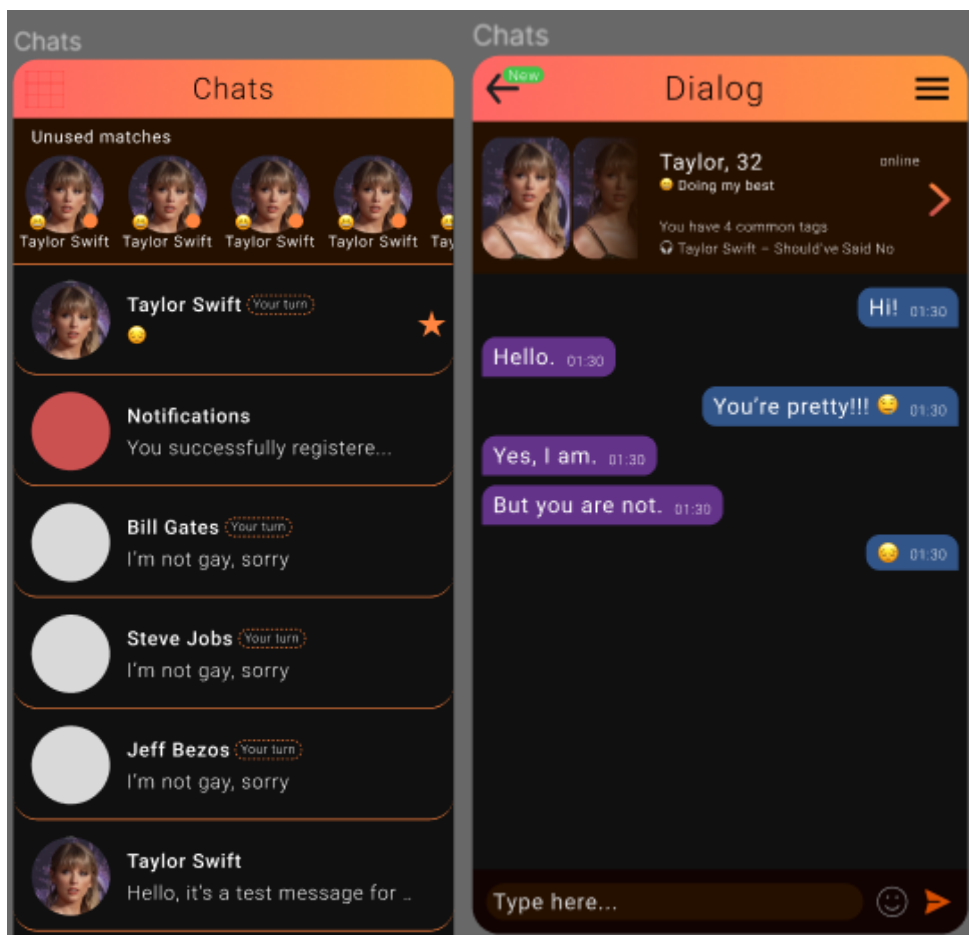


Рис. 15. Сторінка діалогів та відкритого діалогу

На сторінці відкритого діалогу повинні розташовуватися такі елементи як шапка діалогу, не розташована коротка інформація про людину, з якою ведеться діалог та її фотографії, нижче знаходяться повідомлення діалогу та ще нижче – стрічка відправки повідомлення.

Також, є безліч додаткових екранів, таких як редагування та перегляд профіля, налаштування фільтрації (опис ідеального кандидату для пошуку), та налаштування додатку, зображені на Рис. 16.

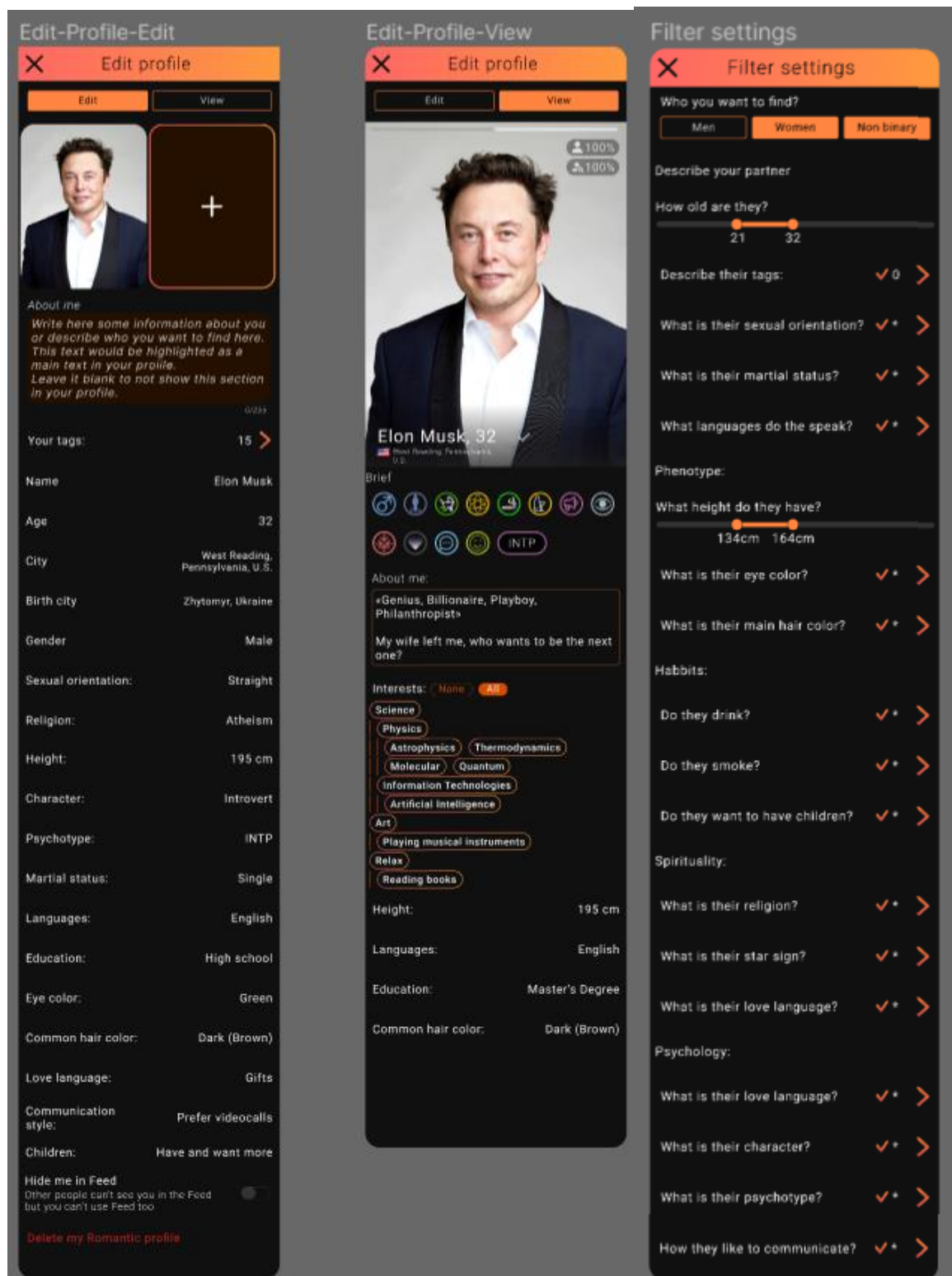


Рис. 16. Редагування профілю, перегляд профілю, налаштування фільтру

Порядок розміщення та зовнішній вигляд компонентів ще будуть корегуватися на етапі демонстрації прототипу цільовій аудиторії та аналізу зворотного зв'язку.

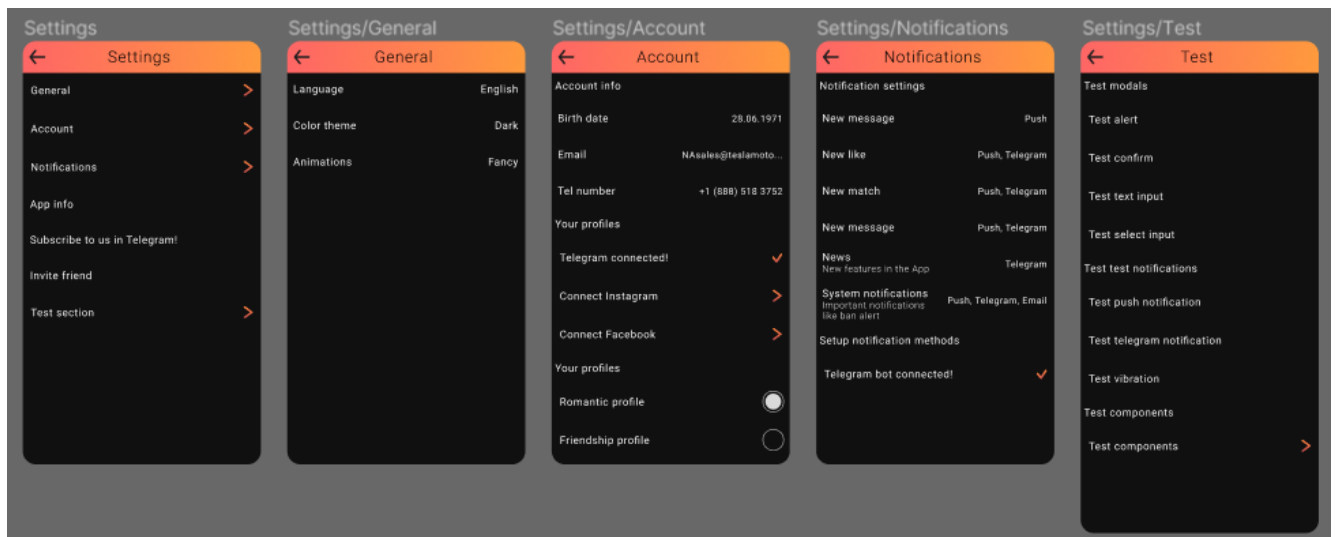


Рис. 17. Налаштування додатку

На Рис. 17 зображено нарис екранів налаштувань додатку.

Також, дизайн передбачає декілька спливаючих вікон (зображені на Рис. 18), таких як панель переключення акаунту (розгортається при натисканні на верхню панель), спливаюче вікно прикріплення повідомлення до вподобання (спливає при натисненні на синю кнопку з іконкою чату), спливаюче вікно вибору (відкривається при кліку на деякі поля в редагуванні профілю та є контекстно-залежним) та параметри діалогу.

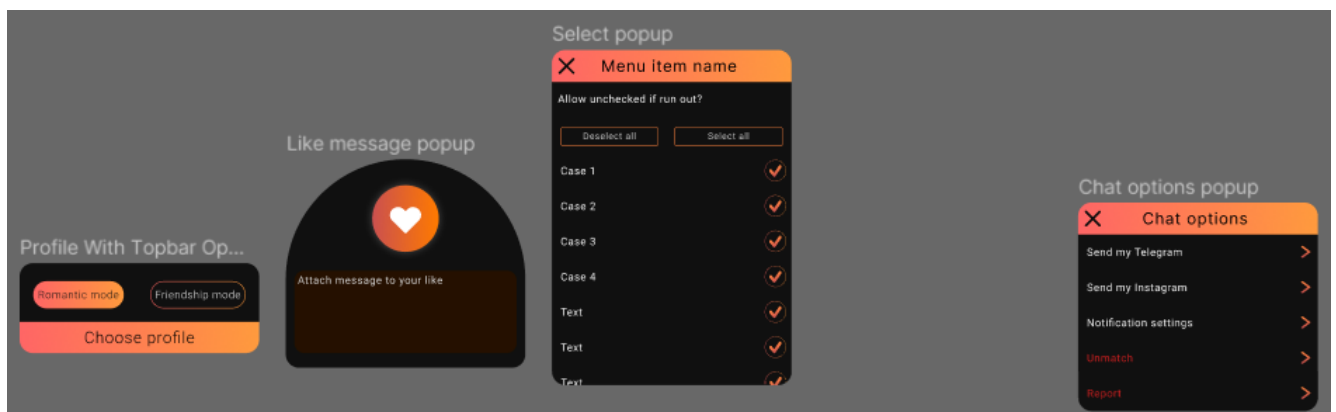


Рис. 18. Спливаючі вікна додатку

Також, є окрема сторінка для налаштування власних інтересів при редагуванні профілю (Рис. 19), вона являє собою деревовидну структуру рекомендованих інтересів, з яких користувач може обрати ті, що цікавлять саме його. При виборі вложеного елемента, повинні вибиратися також всі його

батьківські елементи (наприклад, при виборі інтересу Molecular, також автоматично обираються інтереси Physics та Science). Чим більше за вкладеністю інтереси співпадають, тим вища степінь співпадіння та тим вищий пріоритет планується надавати цій людині.

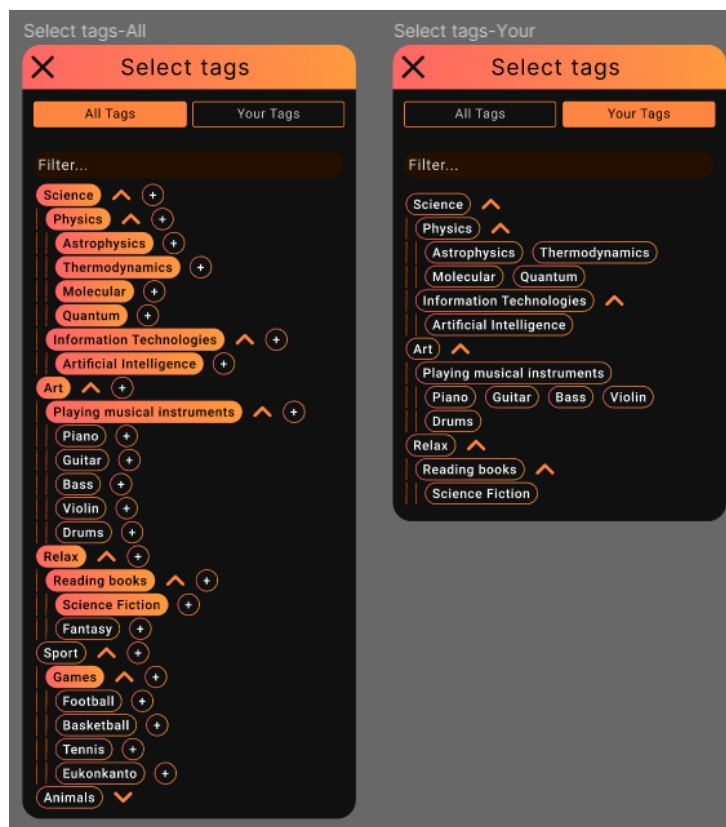


Рис. 19. Налаштування власних інтересів

2.1.4 Прототипування та анімації

Для того, щоб мати можливість перевірити зручність додатку до написання першої стрічки коду, можна використати можливість програми Figma, а саме – прототипування.

Прототипування являє собою налаштування переходів між екранами та відображення спливаючих вікон в залежності від дій користувача. Головною дією є клік на елементи. На Рис. 20 зображено зовнішній вигляд проекту дизайну при увімкненому режимі прототипування, на ньому можна побачити кількість прописаних взаємодій.

Крім того, для переходів можна обрати одну з декількох видів влаштованих анімацій, що робить досвід використання прототипу більш приближеним до досвіду користування реальним додатком.

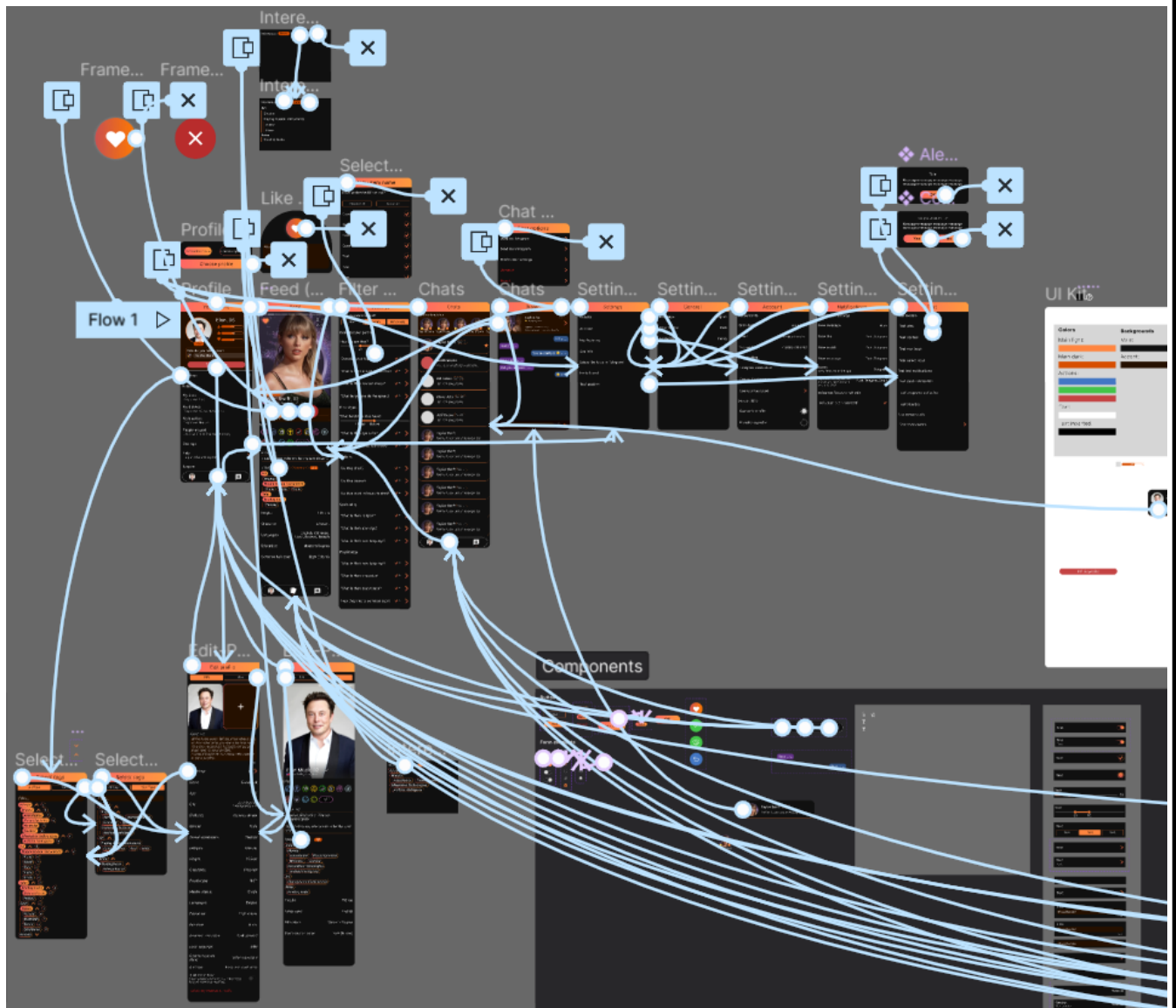


Рис. 20. Figma проект в режимі прототипування

Наприклад, в режимі перегляду прототипу можна перемикатися між екранами додатку, при кліку на деякі елементи будуть відображати спливаючі вікна, будуть працювати анімації переходів, а також, вертикальна та горизонтальна прокрутка екранів та окремих частин, що це підтримують (наприклад, стрічка невикористаних симпатій та фотографій в шапці діалогу мають горизонтальну прокрутку). Приклади деяких екранів зображені на Рис. 21, проте картинки не можуть передати реальний досвід перегляду прототипу.

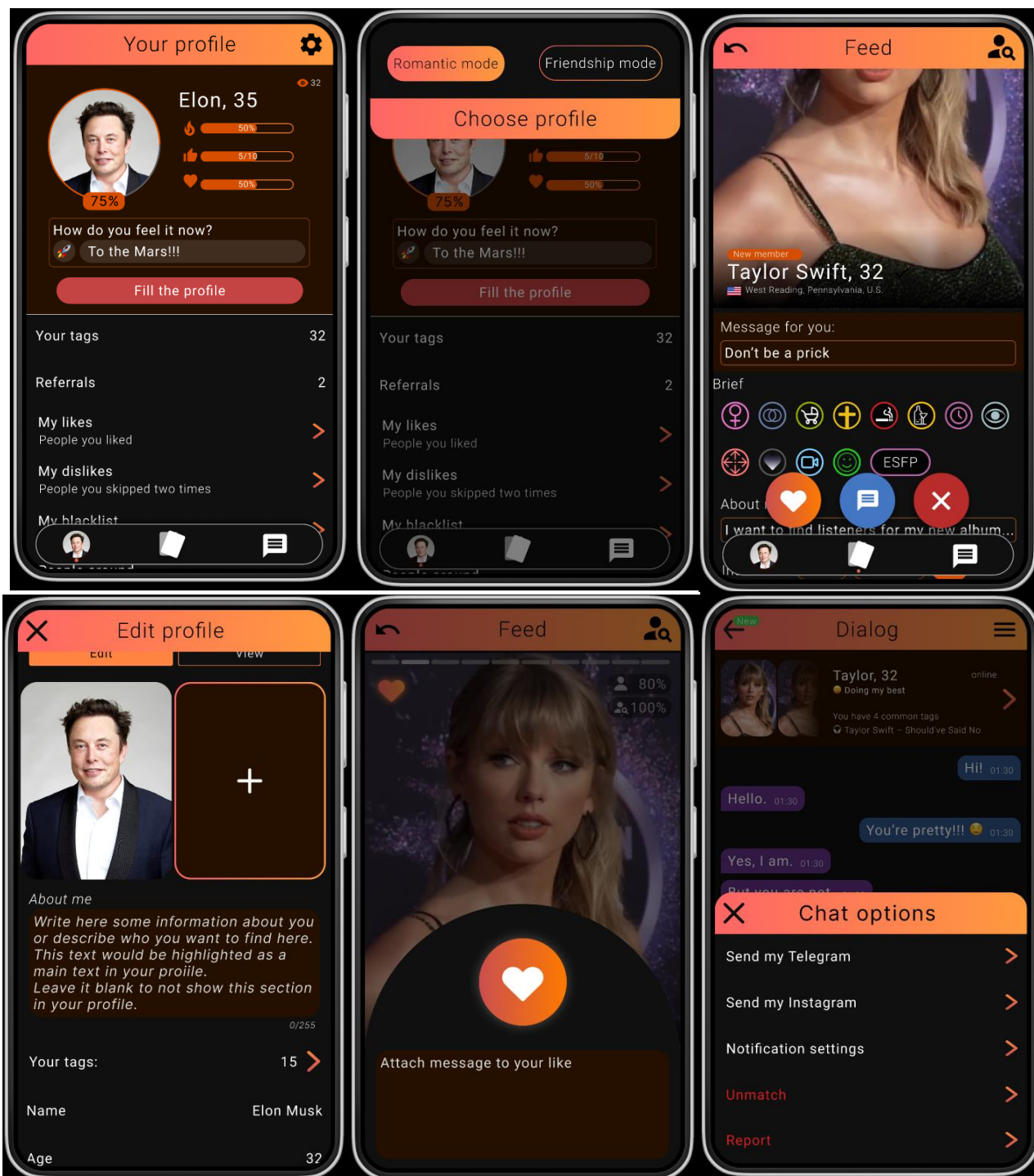


Рис. 21. Деякі екрани в режимі прототипу

2.2 Розробка клієнтської частини

Після завершення розробки прототипу, його було показано декільком представникам цільової аудиторії. Вони відмітили незвичний та цікавий дизайн, а також хороші ідеї використання різних кольорів оформлення для романтичного та дружнього режимів. Також, було відмічено незвичайний підхід до подання інформації про людину у вигляді іконок. Люди поділилися враженнями, що

спочатку це було складно та збивало з пантелику, проте через деякий час вони настільки звикали до цих іконок, що могли скласти враження про людину лише за одним поглядом на них. Крім того, вони підмітили деякі проблеми у дизайні та запропонували покращення, наприклад, відображати повідомлення, прикріплене до вподобання як звичайне повідомлення у чаті, адже це більш інтуїтивно та зрозуміло.

Після аналізу зворотного зв'язку та внесення змін до Figma проекту, було почато розробку клієнтської частини за вже створеним дизайном та прототипом.

Для організації серверної та клієнтської частини було прийняте рішення використовувати різні репозиторії, адже це чітко розділить зони відповідальності, а також, стане у нагоді у випадку, якщо виникне необхідність розробити native додаток з використанням того самого серверу.

Першим кроком для створення клієнтської частини виступала ініціалізація проекту. Для уникнення необхідності власноруч створювати структуру проекту та прописувати усі конфігураційні файли, необхідні для збирання та запуску проекту, було використано утиліту `vue-cli` [23], що дає можливість налаштувати початковий стан для проекту виконанням всього однієї команди та вибором необхідних інструментів у зручному TUI (Text User Interface) вигляді. При ініціалізації було обрано використання мови TypeScript, бібліотеки Vue.js версії 3, Composition API для створення компонентів-прикладів, для форматування коду обрано Prettier, для статичного аналізу якості коду обрано ESLint із стандартними налаштуваннями від Airbnb та використання бібліотеки Vue Router для маршрутизації всередині додатку. Від автоматичного налаштування бібліотеки Vuex було відмовлено на користь подальшого налаштування бібліотеки Pinia.

Структуру папок отриманого проекту зображено на Рис. 22.

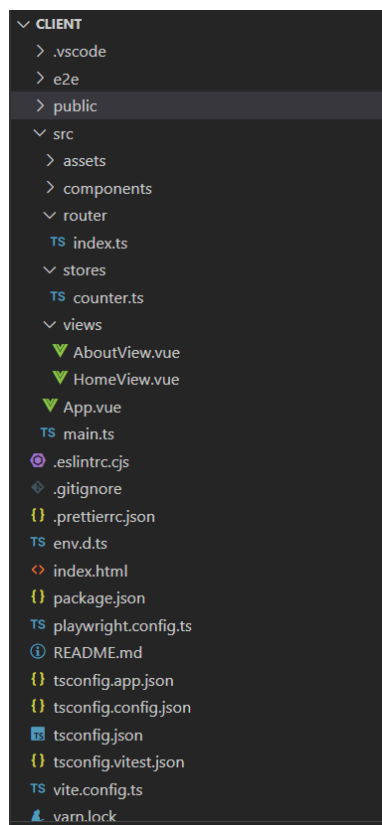


Рис. 22. Структура проекту після ініціалізації

Розглянемо деякі елементи більш детально:

- Assets – папка, що зберігатиме медіа-дані, що будуть імпортуватися у компоненти.
- Components – папка, що міститиме загальні компоненти, що використовуватимуться на різних сторінках додатку, а також, елементи ui-kit.
- Router – папка, що міститиме налаштування умовної та безумовної маршрутизації.
- Stores – папка, що міститиме сховища стану бібліотеки Pinia.
- Views – папка, що міститиме окремі сторінки додатку у деревовидному вигляді та компоненти, що використовуватимуться лише на окремих сторінках.
- Vite.config.ts – файл налаштування збиральника Vite, його налаштування нам знадобиться для реалізації PWA-можливостей.

Типовий компонент у Vue3 з використанням Composition API виглядатиме наступним чином:

```
1  <script setup lang="ts">
2  import { computed } from "vue";
3
4  const props = defineProps<{
5    col?: boolean;
6    gap?: number;
7    center?: boolean;
8    wrap?: boolean;
9  }>();
10
11 const gap = computed(() => props.gap ?? 5);
12 </script>
13
14 <template>
15   <div class="flex" :class="{ col, center, wrap }">
16     <slot></slot>
17   </div>
18 </template>
19
20 <style lang="scss" scoped>
21 > .flex { ...
38 }
39 </style>
```

Рис. 23. Типовий Vue.js компонент на прикладі Flex.vue

Спочатку пишеться HTML-тег `<script setup lang="ts">`, що означає, що використовується спрощений вигляд Composition API, де весь вміст цього тегу буде поміщено у функцію `setup()` та автоматично експортовано всі змінні. Атрибут `lang="ts"` означає, що всередині використовується мова TypeScript.

Власноруч реалізовані компоненти

Описувати весь процес написання клієнтської частини недоцільно, адже він складається із рутинних дій, проте, розглянемо деякі цікаві моменти із його реалізації.

2.2.1 Локалізація

Для реалізації локалізації було обрано використовувати окреме сховище Pinia, що дає змогу миттєво перемикатися між мовами додатку, що сприяє гарному досвіду користувача. Для збереження тексту локалізації серед інших форматів було обрано формат CSV, адже він гарно відображається на GitHub та GitLab, підтримується величезною кількістю табличних редакторів, простий для

ручного редагування, а також, за допомогою всього декількох дій можна налаштувати синхронізацію локалізації із Google Docs, а це означає, що відпадає необхідність у використанні дорогих сервісів локалізації на кшталт Lokalize, Transifex та подібних. У якості роздільника CSV файлу використовується символ «;», а у якості віртуального групування ключів використовується формат із розділенням крапкою. Нижче, на Рис. 24, зображено приклад вмісту файлу локалізації:

```
1 common.action.next;Next;Далі
2 common.action.create;Create;Створити
3 common.action.delete;Delete;Видалити
4 common.action.edit;Edit;Редагувати
5 common.action.change;Change;Змінити
6 common.action.ok;Ok;Ok
7 common.action.cancel;Cancel;Відміна
8 common.action.restart;Restart;Перезапустити
9 common.action.reset;Reset;Скинути
0 common.action.tryAgain;Try again;Спробувати знову
1 common.action.enable;Enable;Увімкнути
2 common.action.disable;Disable;Вимкнути
3 common.toast.copied;Copied;Скопійовано
4 common.datePicker.months;Jan/Feb/Mar/Apr/May/Jun/Jul/Aug/Sep/Oct/Nov/Dec;Січ/Лют/Берез/Квіт/Тра/Черв/Лип/Серп/Верес/Жовт/Листоп/Груд
```

Рис. 24. Перші 14 стрічок файлу локалізації

На даному етапі файл локалізації статично вбудовується у фінальний JavaScript файл у сирому CSV вигляді. Це дозволяє миттєво перемикатися між мовами та не потребує додаткового етапу компіляції для оновлення під час розробки. Проте, є і негативні сторони, такі як додаткова витрата ресурсів на завантаження обох мов та розбір CSV файлу при запуску. У рамках даної кваліфікаційної роботи було вирішено залишити поточну реалізацію, адже файл локалізації досить невеликий та це значно пришвидшує процес розробки. Проте, у скорому майбутньому, коли розмір файлу зросте, планується його розбиття на окремі JSON файли для кожної мови під час компіляції та використання динамічного завантаження під час вибору мови, адже це може значно прискорити момент запуску додатку. Також, у випадку додавання підтримки інших мов, розглядається можливість використання сторонніх сервісів для керування локалізацією.

Розглянемо детальніше Pinia сховище, яке взаємодіє із CSV файлом та надає можливості локалізації. Воно називається lang.ts та зображено на Рис. 25.

```

1  import { defineStore } from "pinia";
2  import { useAppStore } from "../app";
3  import csv from "@lang/locale.csv?raw";
4
5  const lines = csv.split("\n");
6
7  const locale = {} as Record<string, { en: string; uk: string }>;
8
9  lines.forEach((line) => {
10     const [key, en, uk] = line.split(",");
11     locale[key] = { en, uk };
12 });
13
14 You, 4 місяця назад | 1 author (You)
15 export interface I$lOptions {
16     returnKey?: boolean;
17     default?: string;
18 }
19
20 export const useLangStore = defineStore("lang", () => {
21     const appStore = useAppStore();
22
23     const $t = (key: string, options?: I$lOptions) => {
24         let defaultValue = key;
25
26         if (options?.default) defaultValue = options.default;
27         else if (options?.returnKey === false) defaultValue = "";
28
29         return (
30             locale?.[key]?.[appStore.locale] || locale?.[key]?.en || defaultValue
31         );
32     };
33
34     return { $t };
35 });

```

Рис. 25. Сховище локалізації

Спочатку імпортується CSV файл за допомогою механізмів пакувальника Vite, далі відбувається його парсинг та збереження вмісту у словник. На стрічці 14 відбувається оголошення інтерфейсу параметрів локалізації, та на стрічці 19 відбувається оголошення самого сховища.

Так як у додатку широко використовується шаблонізована локалізація (зокрема, у випадаяючих списках), система локалізації підтримує декілька стратегій, наприклад, можна вказати, яке значення повертати, якщо вказаний ключ не знайдено. Розглянемо ці стратегії:

- Звичайна стратегія: Якщо ключ не знайдений, повертається його назва. Ця стратегія дуже зручна для налагодження додатку, а також, дає можливість користувачеві побачити, якого типу текст повинен знаходитися на цьому місці, адже ключі, зазвичай, мають зрозумілі назви.

- `returnKey = false`: Ця стратегія використовується коли потрібно мати можливість дізнатися, що такого ключа не існує, адже при відсутності ключа, функція повертатиме пусту стрічку.
- Значення за замовчуванням: Ця стратегія широко використовується, коли локалізація доступна лише для деяких елементів множини значень. У такому випадку для тих елементів, що мають локалізацію буде застосовуватися локалізація, а для інших – значення за замовчуванням, що, зазвичай, являє собою вже перекладений текст.

2.2.2 Анімації переходу між сторінками додатку

Для анімації між сторінками також було розроблено свою систему. Вона надає можливість налаштовувати тип та тривалість анімації у системі маршрутизації як за допомогою мета-параметрів, так і по шаблону.

Система складається із двох частин: частини, що показує та частини, що налаштовує. Перша частина міститься у файлі `App.vue` (Рис. 26) та являє собою вбудований у Vue тег `<transition>` із динамічними параметрами, що беруться із мета-параметрів системи маршрутизації. Якщо перехід на цю сторінку має опис анімації, вона буде виконана при переході через застосування механізму `v-slot` у компонента `router-view`.

```

18     <router-view v-else v-slot="{ Component, route }">
19       <BottomBar v-if="!route.meta.hideBottomBar" />
20       <transition
21         :enter-active-class="(route.meta.enterAnimation as string)"
22         :leave-active-class="(route.meta.leaveAnimation as string)"
23         :mode="(route.meta.transitionMode as any)"
24         :duration="(route.meta.transitionDuration as number)"
25       >
26         <keep-alive :include="['Profile', 'Feed', 'Chats']">
27           <component :is="Component" :key="route.path" />
28         </keep-alive>
29       </transition>
30     </router-view>

```

Рис. 26. Фрагмент файлу `App.vue`

Частина налаштування знаходиться у файлі `router/index.ts`. Так, як у додатку широко використовуються анімації зсуву, важливим є не тільки

сторінка, на яку виконується перехід, а й сторінка, з якої цей перехід виконується. Це унеможливило використання мета-параметрів напряду, адже в такому випадку анімація буде не контекстно-залежною. Наприклад, при переході на сторінку Люди зі сторінки Профіль повинна програватися анімація справа-наліво, а при переході зі сторінки Чати – зліва-направо. Це досягається за допомогою динамічної зміни мета-параметрів перед самим переходом на сторінку у afterEach hook.

Також, transition-анімації у Vue.js можуть бути трьох типів: default, in-out та out-in. При default типі, анімація зникання попереднього елементу та анімація появи наступного виконуються одночасно, при in-out спочатку виконується анімація появи, а потім – зникання, а при out-in – навпаки. Цей параметр також налаштовується у системі динамічних анімацій.

```

364 router.afterEach((to, from) => {
365   const appStore = useAppStore();
366
367   const f = from.path;      You, 3 місяця назад • ci: autodeploy for feat/server branch ...
368   const t = to.path;
369
370   > function animation(...
380   }
381
382   const SLIDE_DURATION = 500;
383
384   const slideLeft = () => animation("animate__fadeInLeft", "animate__slideOutRight", SLIDE_DURATION);
385   const slideRight = () => animation("animate__fadeInRight", "animate__slideOutLeft", SLIDE_DURATION);
386   const fadeLeft = () => animation("animate__fadeInLeft", "animate__fadeOutRight", SLIDE_DURATION);
387   const fadeRight = () => animation("animate__fadeInRight", "animate__fadeOutLeft", SLIDE_DURATION);
388   animation("", "", 0, "out-in");
389
390   if (!appStore.enableAnimations) return;
391
392   if (t.split("/").length > f.split("/").length) fadeRight(); // in-out?
393   if (f.split("/").length > t.split("/").length) fadeLeft(); // out-in?
394
395   if (t === "/profile" && ["/feed", "/chats"].includes(f)) return slideLeft();
396   if (t === "/chats" && ["/feed", "/profile"].includes(f)) return slideRight();
397   if (t === "/feed" && f === "/profile") return slideRight();
398   if (t === "/feed" && f === "/chats") return slideLeft();
399
400   if (window.innerWidth < 1024 && f.startsWith("/chats")) {
401     if (t === "/chats" && f.startsWith("/chats/chat")) fadeLeft();
402     if (f === "/chats" && t.startsWith("/chats/chat")) fadeRight();
403   } else if (f.startsWith("/chats")) {
404     animation("", "", 0, "in-out");
405   }

```

Рис. 27. Фрагмент системи динамічних анімацій

2.2.3 Pinch Zoom

Якщо подивитися, як реалізоване приближення фотографій у існуючих додатках знайомств, то можна побачити два варіанти: або його немає взагалі (Tinder), або при натисканні на фотографію, вона відкривається на весь екран і там її можна масштабувати як у звичайній програмі перегляду зображень. Проте, це не дуже зручно, адже у першому випадку просто немає можливості збільшити зображення, а у другому – потрібно виконати три дії (відкрити, від масштабувати, закрити).

Тому було прийняте рішення реалізувати масштабування за принципом, схожим на приближення фотографій у Instagram та Telegram, а саме, при виконанні жесту масштабування на мобільному пристрої (англ. Pinch zoom), фотографія збільшується та переміщується, а коли користувач забирає пальці з екрану, зображення плавно повертається на місце.

Після пошуку у репозиторії пакетів NPM, необхідної бібліотеки не зайшлося, отож, було прийняте рішення розробити цей функціонал самостійно.

Функціонал Pinch Zoom було розроблено у вигляді користувацької директиви Vue.js, файл якої знаходиться у папці `src/directives/PinchZoom.ts`. Також, слід зазначити, що ця директива працює лише для мобільних пристроїв із сенсорними екранами.

На Рис. 28 зображено короткий огляд файлу, що реалізує функціонал директиви. Директива у Vue 3 являє собою об'єкт, що реалізує інтерфейс Directive з модуля «vue», зокрема, метод `mounted`, що викликається під час монтування елемента, до якого застосовано директиву, у DOM дерево. Усі наступні дії у коді відбуваються із DOM елементами напряму, що дозволяє використовувати звичайні JavaScript оптимізації під час написання коду директиви. Далі ми розглянемо реалізацію окремих обробників подій сенсорного введення.

```

1  import { preventScrolling } from "@stores/app";
2  import { type Directive, ref, onBeforeUnmount } from "vue";
3
4  const directive: Directive = {
5    mounted: (el: HTMLElement) => {
6      let scale = 1;
7      let startDistance = 0;
8      let currentX = 0;
9      let currentY = 0;
10     let startX = 0;
11     let startY = 0;
12     let zIndex = "0";
13     let isAnimating = false;
14
15     const updateTransform = () => {
16       el.style.transform = `translate3d(${currentX}px, ${currentY}px, 0) scale(${scale})`;
17       isAnimating = false;
18     };
19
20 >   const handleTouchStart = (event: TouchEvent) => { ...
38   };
39
40 >   const handleTouchMove = (event: TouchEvent) => { ...
63   };
64
65 >   const handleTouchEnd = () => { ...
85   };
86
87 >   const getDistance = (touch1: Touch, touch2: Touch) => { ...
92   };
93
94   el.addEventListener("touchstart", handleTouchStart);
95   el.addEventListener("touchmove", handleTouchMove);
96   el.addEventListener("touchend", handleTouchEnd);

```

Рис. 28. Короткий огляд файлу PinchZoom.ts

Реалізація функціоналу відбувається за допомогою підписки на події сенсорного екрану (touchstart, touchmove та touchend) елемента, до якого застосували директиву. Далі у обробнику handleTouchStart відбувається перевірка на те, чи користувач торкнувся екрану саме двома пальцями, та, якщо так, зберігає початкові координати та перемикає елемент у режим масштабування. У обробнику handleTouchMove відслідковується зміна відстані між пальцями та їх розміщення на екрані та відбувається відповідне масштабування та переміщення, викликаючи функцію updateTransform, що виконує трансформацію за допомогою inline стилів. Плавність переміщення та трансформації зображення досягається за допомогою CSS властивості «transition: transform», а «translate3d» було використано для того, щоб браузер завжди використовував апаратне прискорення для виконання цієї трансформації.

Після того, як користувач забирає пальці від екрану, спрацьовує обробник `handleTouchEnd`, який повертає позицію та масштаб зображення до початкових значень та перемикає елемент у звичайний режим. Зображення плавно встає на місце за допомогою CSS властивості «`transition: transform 0.3s ease-in-out`». Функція `getDistance` розраховує відстань між двома пальцями користувача за Евклідовою формулою розрахунку відстані між двома точками на площині, ця відстань використовується для масштабування зображення.

2.2.4 Кадрування фотографій

При завантаженні фотографій, користувачеві важливо мати можливість виконати її кадрування, при цьому, зробити це інтуїтивно, як на телефоні, так і на комп'ютері. В ході дослідження існуючих рішень, було проведено аналіз найбільш популярних та часто використовуваних компонентів кадрування. Найбільш підходящою виявилася бібліотека `vuejs-clipper` [24], проте, вона була доступна лише для Vue 2. Тому було прийнято рішення реалізувати свій компонент кадрування з нуля та додати усі необхідні можливості. Вимоги:

- Цільовий розмір кадрування: 640x800px.
- Область кадрування фіксована, а переміщується та масштабується саме зображення.
- Масштабування колесом миші на комп'ютері та двома пальцями на телефоні.
- Зображення не може бути меншим за область кадрування.
- Область кадрування повинна містити підказки за правилами третин та область заокруглення аватару.

Алгоритм компоненту складається двох етапів. Перший етап – позиціонування. На цьому етапі користувач взаємодіє з DOM елементами, відбувається позиціонування та масштабування фотографії відносно області кадрування. Після натискання на кнопку підтвердження кадрування, запускається другий етап – кадрування. Зображення, зазвичай, завантажується користувачем, проте, компонент може працювати і з

зображеннями з інтернету, головним критерієм є підтримка типу зображення браузером.

Принцип дії алгоритму позиціонування та масштабування майже ідентичний до того, як це реалізовано у директиві Pinch Zoom, що було описано у розділі 2.2.3, єдиними відмінностями є те, що при прибиранні пальців з екрану не потрібно повертати масштаб до стандартного значення, а, замість переміщення двох пальців, для позиціонування використовується лише один. Тому, щоб не повторюватись, розглянемо лише другий етап алгоритму, а саме, кадрування.

Процес кадрування зображено на Рис. 29, він складається з двох етапів. Спочатку, зображення відображається на елементі canvas за допомогою методу drawImage, використовуючи позицію та масштаб з етапу позиціонування, а на другому етапі робиться знімок полотна за допомогою методу toBlob. Отримане зображення у JPEG форматі повертається до батьківського компоненту через подію «crop».

```
234 canvas.width = props.selectorWidth;
235 canvas.height = props.selectorHeight;
236
237 context.drawImage(
238   imageEl.value,
239   x,
240   y,
241   sourceWidth,
242   sourceHeight,
243   0,
244   0,
245   props.selectorWidth,
246   props.selectorHeight
247 );
248
249 canvas.toBlob(
250   (blob) => {
251     emit("crop", blob);
252   },
253   "image/jpeg",
254   1.0
255 );
256
```

Рис. 29. Кадрування зображення

Також, розглянемо процес формування візуальних підказок. Їх реалізувати можна було або за допомогою SVG графіки, або з використанням звичайного CSS. У рамках даної кваліфікаційної роботи було обрано другий спосіб через

кращу швидкодію, проте, у випадку, якщо знадобиться додати додаткові елементи, може бути розглянуто перехід на використання SVG. Відображення підказок за правилом третин працює через псевдо-елементи з абсолютним позиціонуванням та CSS властивість border, для додавання підказки заокруглення аватару знадобилося додати ще один div елемент. Також, на зображеному на Рис. 30 коді можна побачити використання функції v-bind для передачі значень із JavaScript у CSS.

```

372 .cropper-selector {
373   position: absolute;
374   border: 2px solid #fff;
375   box-sizing: border-box;
376   pointer-events: none;
377
378   // Add rule of thirds
379   &::before {
380     content: "";
381     position: absolute;
382     width: 33%;
383     height: 100%;
384     border: 1px solid #fff;
385     box-sizing: border-box;
386     pointer-events: none;
387     left: 50%;
388     transform: translateX(-50%);
389   }
390
391   &::after {
392     content: "";
393     position: absolute;
394     width: 100%;
395     height: 33%;
396     border: 1px solid #fff;
397     box-sizing: border-box;
398     pointer-events: none;
399     top: 50%;
400     transform: translateY(-50%);
401   }
402
403   1 reference
404   .avatar {
405     25 references
406     width: v-bind("props.selectorWidth + 'px'");
407     25 references
408     height: v-bind("props.selectorWidth + 'px'");
409     border-radius: 50%;
410     top: 50%;
411     position: absolute;
412     transform: translateY(-50%);
413     border: 1px solid white;
414   }

```

Рис. 30. Додавання підказки правила третин та контуру аватару

Готовий компонент кадрування та результат його роботи зображено на Рис. 31, сам компонент знаходиться у файлі src/components/ui-kit/Cropper.vue.

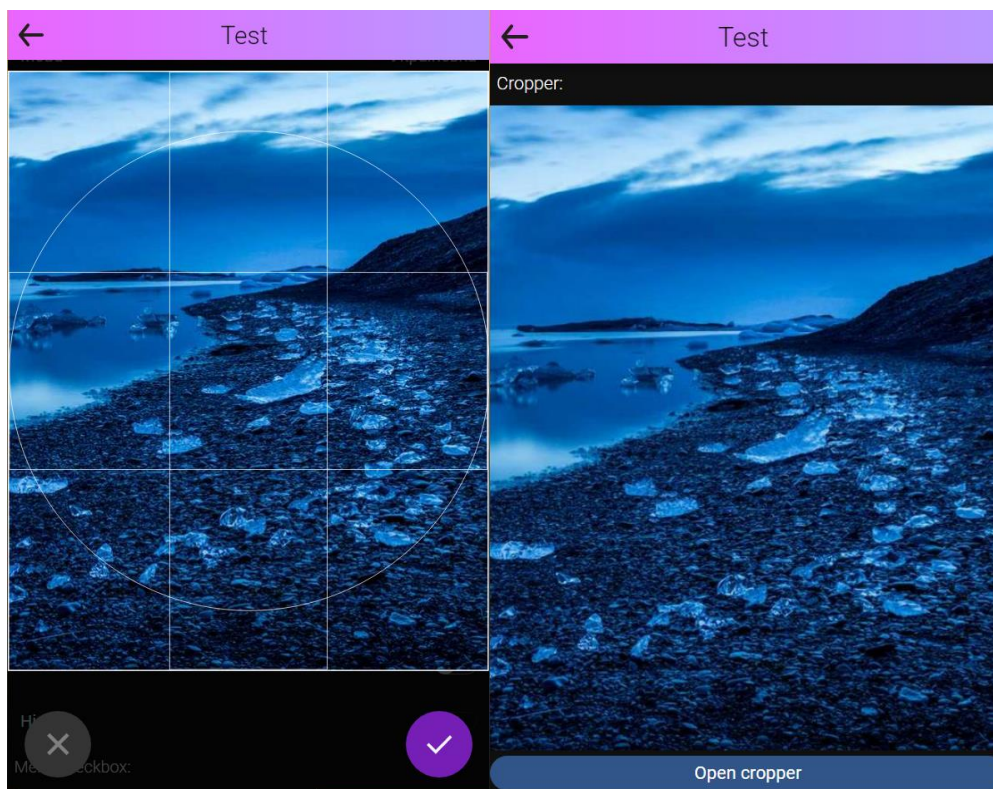


Рис. 31. Вигляд компонента кадрування та результат його роботи

2.2.5 Розпізнавання складних емої

Ще однією особливістю, яку було вирішено додати до функціоналу додатку були стікери. Стікери являють собою звичайні емої, проте, збільшені у 3 рази, та слугують для швидкого вираження емоцій. Так як у додатку використовується набір емої у SVG форматі (набір Twemoji [25] від Twitter), вони можуть бути збільшені без втрати якості. Для того, щоб відправити стікер, необхідно відправити одне емої у повідомленні. Схожий функціонал можна побачити у популярному месенджері Discord, проте, там просто емої трохи збільшуються у випадку відправки їх у кількості не більше трьох штук.

Але, розпізнавання кількості емої, що були відправлені це доволі складна задача, адже, по-перше, самі емої складаються з деякої послідовності Unicode символів (тому звичайна перевірка на `String.length` покаже неправильну кількість). Тут може прийти ідея використати можливості прапорця «u» класу RegExr та скласти регулярний вираз для розділення стрічки за конкретними емої, що виглядатиме, приблизно, так: `«\p{Extended_Pictographic}/gu`. Цей

метод добре працює для прости емоїї, проте, якщо ми спробуємо передати туди, наприклад, емоїї із зображенням сім'ї (як це зображено на Рис. 32), ми побачимо чотири окремих емоїї. У такому випадку, емоїї не перетворилося би на стікер, хоча, за задумкою повинно.

```
> Array.from('👨👩👧👦'.match(/\\p{Extended_Pictographic}/gu))
< ▶ ['👨', '👩', '👧', '👦']

> Array.from('👨👩👧👦'.match(/\\p{Extended_Pictographic}/gu))
< ▶ (4) ['👨', '👩', '👧', '👦']
```

Рис. 32. Спроба розпізнавання кількості емоїї за допомогою регулярного виразу

Щоб розібратися, чому це працює саме так, звернемося до статті «Gender Emoji ZWJ Sequences» від авторів M. Davis, R. Been, and the Unicode Emoji Subcommittee [26]. На Рис. 33 зображено приклад із цієї статті, що показує, яким чином створюються складні емоїї за допомогою комбінування декількох простих та символу 0x200D під назвою «ZWJ».

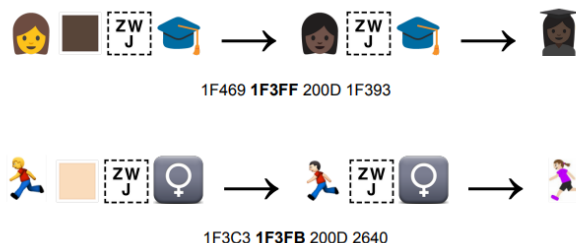


Рис. 33. Приклад складних Емоїї із статті «Gender Emoji ZWJ Sequences»

Тобто, для розпізнавання складних емоїї потрібно шукати інший спосіб, та такий спосіб було знайдено, а саме, модуль Intl [27] та його клас Segmenter. Він дозволяє інтелектуально розділити стрічку на логічні сегменти таким чином, як їх сприймає людина. Клас Segmenter підтримується усіма браузерними крім Mozilla Firefox з листопада 2020 року, тож, для Firefox більш старіших браузерів будемо використовувати спосіб з регулярним виразом..

														
		Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet			WebView Android
Intl		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		24	12	29	15	10	25	56	14	10	1.5	4.4	1.8	0.12.0
Segmenter		✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
		87	87	No	73	14.1	87	No	62	14.5	14.0	87	1.8	16.0.0

Рис. 34. Підтримка Intl.Segmenter браузерами

Після закінчення аналізу розпізнавання складних емоїї було написано наступну функцію (файл src/utlis.ts).

```

105 // Self-made complex emoji parser
106 // Read more about complex emojis: https://www.unicode.org/L2/L2016/16181-gender-zwj-sequences.pdf
107 export const parseEmojis = (str: string) => {
108   if (!("Segmenter" in Intl)) {
109     console.warn("Intl.Segmenter not supported");
110     return [...(str.match(/\p{Extended_Pictographic}/gu) || [])];
111   }
112
113   return [...new Intl.Segmenter().segment(str)]
114     .map((x) => x.segment)
115     .filter((x) => x.length > 1);
116 };

```

Рис. 35. Функція для розпізнавання емоїї

Після перевірки працездатності функції на великій кількості простих та складних емоїї, її було додано до екрану ChatMessage.vue. На Рис. 36 зображено її роботу – коли користувач відправляє повідомлення з одним емоїї (не важливо, простим або складним) та без додаткового тексту, воно автоматично перетворюється на стікер.

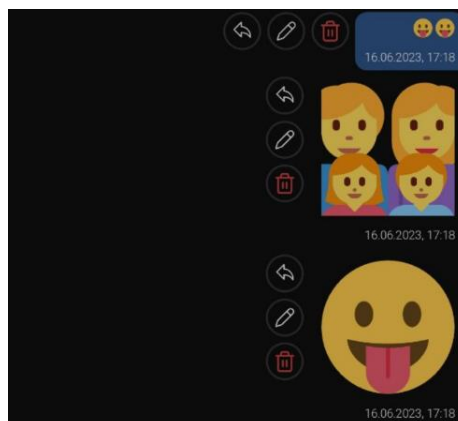


Рис. 36. Демонстрація роботи функції розпізнавання емоїї

2.3 Проектування архітектури бази даних

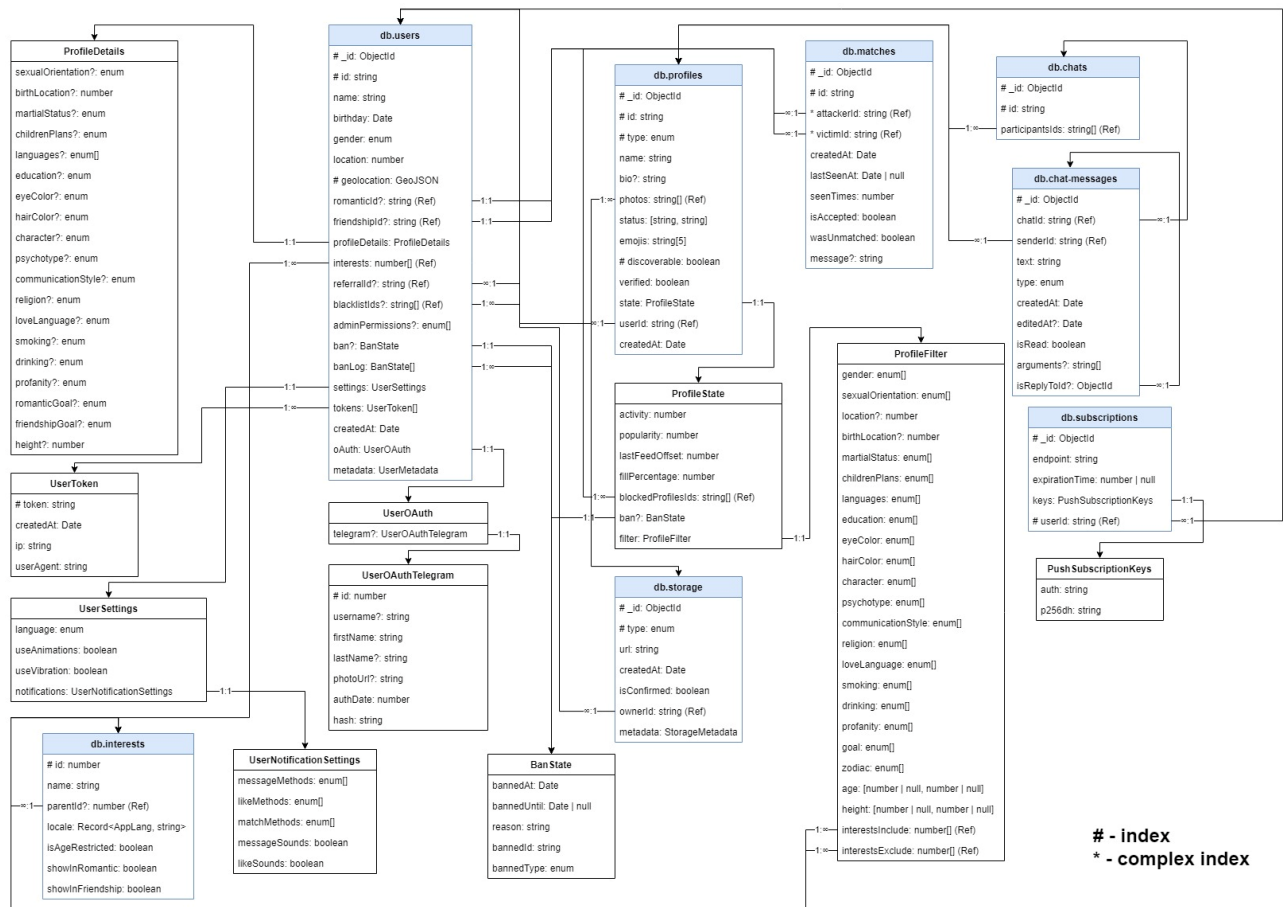


Рис. 37. Схема бази даних

У якості бази даних було обрано СКБД MongoDB [12] через її зручність при роботі зі складними та вкладеними структурами даних без необхідності нормалізації, швидкодію та популярність. У якості драйвера використовується драйвер MongoDB для Node.JS. У якості ORM використовується Mongoose. Для зручної інтеграції Mongoose з TypeScript використовується бібліотека Typegoose. Для інтеграції Typegoose із Nest.JS використовується бібліотека Kindagoose.

Розглянемо більш детально схему бази даних, зображену на Рис. 37. Почнемо з легенди. У прямокутниках зображені окремі класи документів, блакитним кольором позначені кореневі документи колекцій, решіткою позначені прості індекси, зірочками позначені складні індекси (індекс, що складається з декількох полів) [28]. Зв'язки між документами позначені стрілочками, від поля, що посилається на документ до документа, на якого це

поле посилається. Біла поля, що посилається на документ розміщується позначка, що показує тип зв'язку (1:1 – один-до-одного, 1:∞ - один-до-багатьох, ∞:1 – багато-до-одного, та ∞:∞ - багато-до-багатьох). Поля, що містять у собі ID іншого документа для більш складних зв'язків позначені міткою «(Ref)». Для отримання інформації із документів, на які посилається це поле необхідно використати \$lookup стадію агрегації або виконати окремий запит на отримання документа за його ID.

Розглянемо детальніше окремі колекції та розберемо їх призначення:

- **db.users** (модель знаходиться у файлі user.model.ts): Ця колекція містить всю інформацію, що стосується окремого користувача, починаючи від даних авторизації, закінчуючи посиланнями на профілі та інформацію, що є загальною для обох профілів (наприклад, поле profileDetails, що містить інформацію про цінності користувача або поле interests, що містить перелік інтересів користувача). Поле id містить GUID з префіксом «u-» із налаштованим унікальним індексом. Також, на полі geolocation налаштовано 2dsphere індекс для того, щоб мати можливість шукати користувачів із сусідніх міст, або ж, шукати найближчих користувачів від деякої точки, якщо згодом буде реалізоване задання геолокації.
- **db.profiles** (модель знаходиться у файлі profile.model.ts): Ця колекція містить інформацію про конкретний профіль користувача, включаючи посилання на фотографії, налаштування фільтрів пошуку, а також, поточний стан профіля, такий як поточна позиція «курсора» у стрічці, активність та інші параметри. Поле id містить GUID із префіксом «r-» або «f-» (в залежності від типу профілю) із налаштованим унікальним індексом, це дає можливість отримати тип профіля за його ID, а також, відслідковувати помилки, пов'язані із типом профілю. Також, індекс налаштовано на полі type, що дає змогу швидко фільтрувати профілі за типом. Крім того, індекс налаштовано на полі discoverable для того, щоб не

витрачати ресурси на обробку відключених профілів. Поле `emojis` містить масив із 5 елементів, що являють собою Unicode строку з окремими Емої.

- **db.matches** (модель знаходиться у файлі `matches.model.ts`): Ця колекція містить інформацію про симпатії та співпадиння користувачів. Коли користувач натискає на кнопку «Вподобання», автоматично створюється документ у цій колекції, де у полі `attackerId` зберігається `id` профіля користувача, що натиснув кнопку «Вподобання», а у полі `victimId` – профіль на якому цю кнопку натиснули. Коли інший користувач також натискає «Вподобання», стан документа змінюється на `«isAccepted: true»`. Це дає змогу використовувати одну колекцію як для обліку симпатій, так і для обліку співпадинь. Поле `id` містить GUID з префіксом «mr-» або «mf-» (в залежності від режиму), що дає змогу легко відстежувати помилки, пов'язані з утворенням симпатій у різних режимах додатку, а також, реалізувати перевірку на те, щоб обидва профіля та сам документ симпатії були однакового типу. Також, на поле `id` налаштовано унікальний індекс, проте він не використовується та його, при необхідності, можна видалити задля заощадження місця на диску. Також, на цій колекції встановлено складний унікальний індекс на поля `attackerId` та `victimId`, адже пара цих полів являють собою ключ цього документу.
- **db.chats** (модель знаходиться у файлі `chats.model.ts`): Ця колекція зберігає чати користувачів. Поле `id` містить GUID з префіксом «cr-» або «cf-» (в залежності від режиму), що дає змогу легко відстежувати помилки, пов'язані з створенням чатів між різними типами профілів, а також, реалізувати перевірку на те, щоб профілі усіх учасників чату були однакового типу та сам документ був того самого типу. Також, на поле `id` налаштовано унікальний індекс. Поле `participantsIds` являє собою масив `ID` профілів користувачів, що долучені до цього чату. У поточній реалізації у чаті може бути лише два користувачі, проте, залишається можливість для реалізації групових чатів у майбутньому. Можливості MongoDB

працювати із посиланнями дають можливість реалізувати відношення багато-до-багатьох без створення додаткової колекції.

- **db.chat-messages** (модель знаходиться у файлі chat-messages.model.ts): Ця колекція містить окремі повідомлення у чатах користувачів, кожне повідомлення є окремим документом. Поле isLocalized слугує для позначки повідомлень, текст яких має бути пропущений через функцію локалізації, розглянуту у пункті Локалізація цього підрозділу, проте, на даний момент, ця можливість не використовується на користь локалізації повідомлень під час їх відправки на сервері. Поле arguments має містити аргументи для локалізації у випадку використання isLocalized, на даний момент також не використовується. Поле type зарезервоване для реалізації голосових та системних повідомлень у майбутньому. Поле isReplyToId являє собою ObjectId повідомлення, відповіддю на яке є поточне повідомлення. Також, на цій колекції налаштовано складний індекс за полями chatId (прямий порядок) та createdAt (зворотній порядок). Це зроблено для пришвидшення технології Lazy-loading [29] для отримання повідомлень із чату.
- **db.subscriptions**: Ця колекція містить Web Push підписки конкретних пристроїв користувачів. Документи складаються із endpoint строки та двох ключів шифрування. На поле userId налаштовано індекс для швидкого пошуку усіх підписок конкретного користувача, що використовується при відправці повідомлення.
- **db.storage** Ця колекція зберігає інформацію про завантажені файли. На даний момент, файлами можуть бути лише JPEG фотографії, проте ця колекція буде також використовуватися для збереження інформації про завантажені аудіо-файли та інші типи даних. Головна її ціль полягає у тому, щоб пов'язати завантажені на диск файли із користувачем, що їх завантажив. У самому коді сервера вона для читання не використовується, проте, буде корисною у випадку виявлення правопорушень.

- db.interests: У цій колекції зберігається дерево інтересів у вигляді списку. Поле locale містить локалізацію інтересу. Поле id являє собою числове значення із штучним auto-increment, також, на це поле накладено унікальний не незмінний індекс. Поле parentId містить числове значення поля id батьківського інтересу або -1, якщо це інтерес верхнього рівня.

2.4 Розробка серверної частини

Розробка серверної частини виконується за загальними принципами проектування серверних систем з використанням фреймворку Nest.JS, тому в даному розділі не буде описуватися кожен модуль, контролер та сервіс, проте, буде акцентовано увагу на нестандартних рішеннях та системах власного написання, що були створені в процесі розробки серверної частини.

Почнемо із загального огляду модулів, зображеного на Рис. 38.

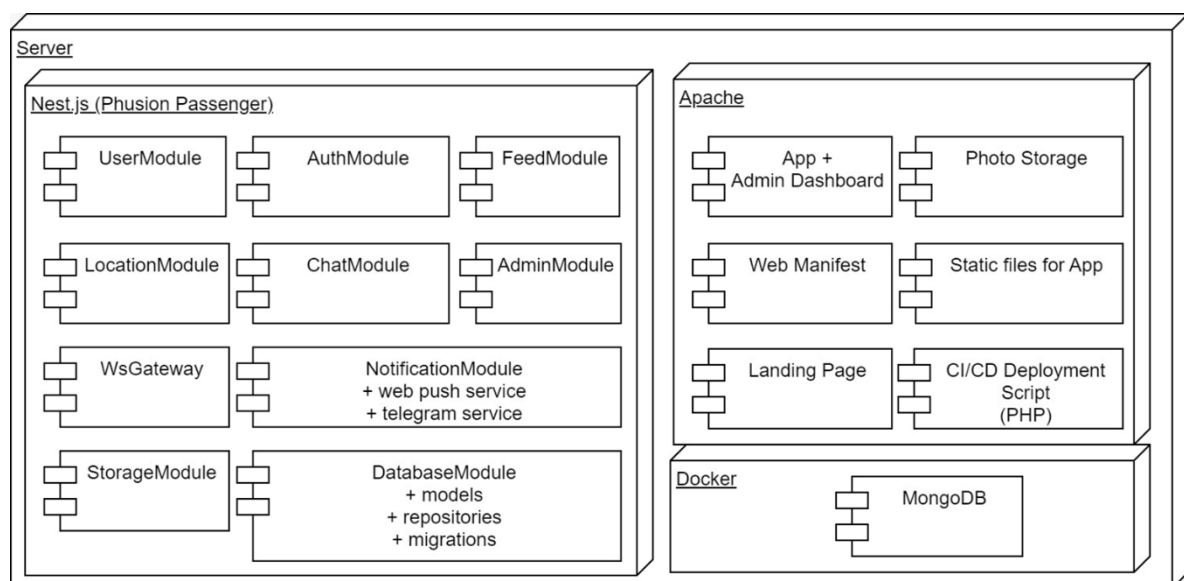


Рис. 38. Спрощена діаграма модулів серверу (зліва)

- AdminModule – Містить закрите API для обслуговування системи. Щоб використовувати це API необхідно мати відповідні ролі адміністрування.
- AuthModule – Відповідає за авторизацію за допомогою Telegram, перевірку прав користувача, а також надає декоратори для зручного застосування різних перевірок у контролерах інших модулів.

- ChatModule – Містить API для роботи з чатами, перегляду та відправки повідомлень, а також іншими діями, пов'язаними із внутрішніми чатами додатку.
- DatabaseModule – Цей великий модуль відповідає за підключення та роботу з базою даних. Він містить в собі усі моделі та репозиторії для роботи з ними, а також, інкапсулює у собі всі операції з базою даних. Таким чином, інші модулі взаємодіють саме з різними елементами DatabaseModule, нічого не знаючи про конкретну реалізацію бази даних. Це дало можливість за весь час розробки додатку змінити 4 бази даних (JavaScript Object, JSON файл, PostgreSQL та MongoDB) у пошуках найкращого рішення без необхідності переписувати решту модулів. Також, цей модуль містить у собі сервіс для запуску стартової міграції, що ініціалізує пусту базу даних необхідною базовою інформацією, такою як системний профіль.
- FeedModule – Цей модуль відповідає за все, що стосується пошуку людей. Він містить алгоритми підбору та фільтрації людей, а також механізм симпатій та співпадінь.
- LocationModule – Цей модуль відповідає за надання інформації про різні міста та селища країн світу.
- NotificationModule – Цей модуль відповідає за всі види зовнішніх сповіщень користувачів додатку. Він містить в собі сервіси для відправки сповіщень за допомогою Telegram боту, а також сервіс для Push-сповіщень. Також, він містить в собі шаблони сповіщень українською та англійською мовами.
- StorageModule – Цей модуль відповідає за збереження медіа-даних користувачів. На даний момент, до таких даних відносяться фотографії профілів. Він відповідає за завантаження фотографій, отримання фотографій, а також, перевірку фотографій на неприпустимий контент за допомогою нейронної мережі та автоматичне цензурування фотографії у випадку позитивного результату перевірки.

- UserModule – Цей модуль надає API для редагування інформації поточного користувача, заповнення профілів, редагування налаштувань а також чорний список користувача, та генерацію персональних QR-кодів запрошень за реферальною програмою. У майбутньому розглядається можливість розбиття цього модуля на декілька окремих.
- WsModule – Цей модуль містить головний шлюз для роботи з технологією Web Sockets. Він реагує на повідомлення користувачів, взаємодіє з іншими модулями та відправляє повідомлення реального часу на під'єднані клієнти.
- AppModule – Це головний модуль додатку, який імпортує усі інші модулі.

Далі розглянемо деякі цікаві елементи, що були реалізовані в процесі написання серверної частини.

2.4.1 Авторизація

Метод авторизації за допомогою логіну та паролю є одним із самих давніх та ефективних, адже він дає можливість увійти до акаунту з будь-якого пристрою. Але, також, він має безліч недоліків, наприклад, їх можна забути, адже надійний пароль повинен бути довгим та використовувати комбінацію великих та малих літер, цифр та спеціальних символів. Також, їх потрібно постійно пам'ятати та, в ідеалі, мати для кожного сервісу окремий пароль. Також, ці дані можна підібрати методом перебору. Для вирішення цих недоліків у наші часи рекомендується використовувати менеджери паролів – це програми, які генерують та зберігають паролі від усіх сервісів в одному місці із одним майстер-паролем та складною системою авторизації та ключів.

Саме тому, було прийняте рішення відмовитися від класичної авторизації за допомогою логіну та паролю у користь більш сучасних способів авторизації, а саме OIDC [30] та OAuth [31]. Технологія OAuth дає можливість зовнішнім додаткам і сервісам отримувати обмежений доступ до інформації користувача без необхідності розкривати свій логін та пароль. Вона дозволяє користувачам надавати обмежені повноваження іншим додаткам, які взаємодіють з їх

обліковим записом. Наприклад, надати можливість зовнішньому додатку записувати інформацію у Google Диск користувача. OAuth використовує токени доступу, які видані відповідним сервером авторизації.

OIDC (OpenID Connect), в свою чергу, є розширенням протоколу OAuth і забезпечує ідентифікацію та аутентифікацію користувача. Використовуючи OIDC, додатки можуть отримувати не лише токени доступу, але і базову інформацію про користувача, яка підтверджує його ідентичність. Це дозволяє забезпечити більш безпечну та довірчу аутентифікацію, зменшуючи ризик шахрайства та підробки ідентичності. OIDC можна розглядати як обмежену реалізацію OAuth, що спрямована саме на реалізацію авторизації.

У якості сервісу авторизації було обрано Telegram, адже це самий популярний месенджер у наш час, для створення Telegram акаунту необхідно підтвердити номер телефону та неможливо створити декілька акаунтів з одного номеру. Таким чином, автоматично ускладнюється можливість створення декількох акаунтів, а також, обходу блокування. Ще одною перевагою авторизації через Telegram є автоматичне надання прив'язаному боту можливості писати авторизованому користувачу. Це знімає необхідність, також, і вказувати адресу електронної пошти, у користь повного переходу на сучасні месенджери.

Таким чином, для користувача процес реєстрації та авторизації виглядає однаково: усе виконується миттєво, натисканням однієї кнопки, більше немає необхідності запам'ятовувати логін, пароль та адресу електронної пошти, які були вказані при реєстрації.

Тепер розглянемо, як це працює всередині. Для NestJS існує стандартизований пакет бібліотек для реалізації різних стратегій авторизації під назвою Passport. Проте, після більше детального вивчення принципу її роботи, було прийняте рішення відмовитися від її використання та реалізувати стратегію авторизації самостійно. На це вплинуло декілька причин: по-перше, велика кількість залежностей, що необхідно привнести до проекту для налаштування

авторизації; по-друге, бібліотека Passport мутує об'єкт Request бібліотеки Express.js, що є дуже поганим тоном та може призвести до помилок; по-третє, реалізація алгоритму авторизації через Telegram не є дуже складною та це єдина стратегія авторизації, що буде використовуватися у додатку.

Розглянемо процес авторизації на спрощеному рівні: спочатку користувач натискає на кнопку авторизації, що запускає Telegram Widget [32] на клієнтській частині, цей віджет відкриває спливаюче вікно, де користувачеві пропонується увійти в Telegram акаунт або підтвердити авторизацію у додатку якщо в акаунт уже виконано вхід. Після підтвердження, користувачеві відправляється запит від внутрішнього бота Telegram, де запитується, чи дійсно користувач хоче авторизуватися у додатку, якщо користувач підтверджує авторизацію, Telegram Widget повертає ID користувача, іншу основну інформацію, а також хеш-суму, підписану токеном боту авторизації. Далі клієнтська частина додатку відправляє ці дані на сервер, який, у свою чергу, виконує хешування отриманих даних (без самого поля хешу) за алгоритмом SHA256, а потім перевіряє достовірність даних, проводячи трансформацію хеш-суми за алгоритмом HMAC з використанням токenu боту у якості секретного ключа. Далі ці строки порівнюються та у випадку співпадиння вважається, що дані правдиві та виконується реєстрація або авторизація користувача в системі.

У вигляді коду процес перевірки достовірності отриманих даних авторизації виглядає як зображено на Рис. 39. Для виконання хешування за алгоритмами SHA256 та HMAC використовується внутрішня бібліотека Node.JS під назвою crypto. Перед хешування дані перетворюються у спеціальну форму, описану в документації. Для цього їх потрібно розмістити у форматі «ключ=значення», відсортованими за ключами у алфавітному порядку, у якості роздільника даних використовується пуста стрічка.

```

33  checkTelegramOAuthData(data: ICUserOAuthTelegramDto) {
34      const { hash, ...dataWithoutHash } = data;
35
36      const dataCheckString = Object.keys(dataWithoutHash)
37          .sort()
38          .map((k) => `${k}=${data[k]}`)
39          .join("\n");
40
41      if (!TELEGRAM_BOT_TOKEN) throw new Error("TELEGRAM_BOT_TOKEN is not set");
42
43      const secretKey = sha256(TELEGRAM_BOT_TOKEN);
44
45      const hmac = hmacSha256Hex(secretKey, dataCheckString);
46
47      return hmac === hash;
48  }
49  }
50
51  function sha256(payload: string) {
52      return createHash("sha256").update(payload).digest();
53  }
54
55  function hmacSha256Hex(key: string | Uint8Array, msg: string) {
56      return createHmac("sha256", key).update(msg).digest("hex");
57  }
58  }

```

Рис. 39. Перевірка достовірності даних авторизації

Після проходження процесу аутентифікації за допомогою Telegram, користувачеві видається токен доступу, який зберігається у localStorage та відправляється при кожному запиті.

2.4.2 Дерево інтересів

Однією з визначних особливостей додатку є дерево інтересів, що дозволяє користувачеві з легкістю описати свої захоплення, поступово заглиблюючись у напрямки, що його цікавлять. Під час аналізу ринку, такої можливості не було знайдено в жодного з існуючих додатків. Проте, реалізація такого дерева додає додаткової складності, тому що кожен інтерес є окремим елементом, що може бути у багатьох користувачів, при цьому потрібно мати можливість легко змінити його. Також, кожен інтерес повинен мати свою глибину та метадані (наприклад, деякі інтереси слід показувати лише у певному режимі, а для інших потрібно використовувати обмеження за віком). Для відображення дерева інтересів на клієнтській частині зручно зберігати усе дерево цілком у JSON вигляді, проте для того, щоб мати можливість посилатися на окремий інтерес у профілі користувача зручно організувати його у вигляді індексованого списку. Для реалізації системи інтересів у додатку було використано обидва варіанти.

Інтереси зберігаються у базі даних у вигляді окремих документів із числовим ID (з автозбільшенням) та ID батьківського елемента. Керування інтересами відбувається через панель адміністрування, а профілі користувачів містять лише масив із числових ID своїх інтересів. Проте, при кожній зміні інтересів через панель адміністрування відбувається генерація дерева інтересів за допомогою рекурсивного алгоритму та отримане дерево зберігається у файл кешу. У майбутньому розглядається можливість зберігання отриманого дерева у базі даних Redis, проте на даному етапі це є надлишковим та використання звичайного JSON файлу повністю покриває потреби системи. Отримане дерево відправляється клієнту за запитом та кешується на стороні клієнта за стратегією Stale-while-revalidate для забезпечення гарного досвіду користувача.

Алгоритм генерації дерева інтересів за списком у колекції зображено нижче.

```

241 async interestsTreeGenerate() {
242   const interests = await this.interestsGetAll();
243
244   const entityToItem = (entity: IIInterestEntity): ISInterestTreeItem => ({
245     ...entity,
246     children: [],
247   });
248
249   const tree: ISInterestTreeItem[] = interests.filter((interest) => !interest.parentId).map(entityToItem);
250
251   const getChildren = (interest: ISInterestTreeItem) => {
252     const children = interests.filter((theInterest) => theInterest.parentId === interest.id);
253
254     if (children.length) {
255       interest.children = children.map(entityToItem);
256       for (const child of interest.children) {
257         getChildren(child);
258       }
259     }
260   };
261
262   for (const interest of tree) {
263     getChildren(interest);
264   }
265
266   return tree;
267 }

```

Рис. 40. Алгоритм генерації дерева інтересів

2.4.3 Перевірка фотографій на пристойність за допомогою ШІ

Модерація є важливою складовою супроводу соціальних мереж, адже контент, розміщений користувачами повинен відповідати правилам сервісу, нормами моралі та закону, зокрема, завантажені фотографії. Проте, не завжди є достатня кількість ресурсів для залучення живих модераторів. Для таких випадків можна використати можливості штучного інтелекту, а саме,

рекурентних та згорткових нейронних мереж. У рамках даної кваліфікаційної роботи для перевірки фотографій на пристойність було використано зовнішню модель, що розташована на ресурсі HuggingFace за адресою `suko-nsfw.hf.space` та має відкритий API.

При завантаженні фотографії, `StorageModule` автоматично відправляє її на перевірку вищевказаній моделі та у випадку виявлення непристойних матеріалів, автоматично цензурує фотографію. Завантаження фотографії до моделі виконується за допомогою читання завантаженого JPEG зображення, кодування його у формат Base64, перетворення у Data-URI строку та відправки POST запиту до API `/run/predict` за допомогою бібліотеки `Axios`.

У випадку виявлення непристойного контенту, `StorageModule` автоматично виконує його цензурування. У якості цензурування було обрано додавання сильного розмиття за алгоритмом Гауса та накладання зверху напису «CENSORED». Під час аналізу можливостей серверної обробки фотографій було розглянуто безліч бібліотек з репозиторію NPM. Деякі з них надають широкі можливості з редагування фотографій, проте містять величезні платформи-залежні модулі, що компілюються за допомогою Node Gyp. Інші бібліотеки надають API для виклику вже встановлених у системі утиліт. Проте, після аналізу існуючих утиліт було прийняте рішення не використовувати їх, а замість цього виконувати виклик програмного пакету `ImageMagick` [33], а саме, утиліти `mogrify`, що виконує обробку фотографії із перезаписом файлу. Таке рішення було прийняте через те, що це єдине місце у додатку, де необхідна обробка фотографій та ця обробка може бути виконана за допомогою однієї шаблонної команди без додаткових ускладнень та збільшення кількості залежностей додатку.

Для цензурування використовується наступна команда: `«mogrify -blur 50x50 -pointsize 50 -fill white -stroke black -strokewidth 2 -gravity center -annotate +0+0 "CENSORED" "${imageFullPath}»`. Повний код методу з цензурування зображено нижче.

```

70 async censorPhoto(id: string) {
71   const isNSFW = await this.checkForNudity(id);
72
73   if (!isNSFW) {
74     this.logger.log("Photo ${id} is safe");
75     return false;
76   }
77
78   this.logger.warn("Photo ${id} contains NSFW content! Censoring...");
79
80   const path = `${STORAGE_PATH}/${id}`;
81
82   const imageFullPath = `${path}/original.jpg`;
83
84   // Blur the original
85   const command = `mogrify -blur 50x50 -pointsize 50 -fill white -stroke black -strokewidth 2 -gravity center -annotate +0+0 "CENSORED" "${imageFullPath}"`;
86
87   // Execute
88   const { stderr } = await exec(command);
89
90   if (stderr) {
91     this.logger.error(stderr);
92     throw new Error("Error while executing mogrify");
93   }
94
95   this.logger.log("Censored!");
96
97   return true;
98 }

```

Рис. 41. Код цензурування непристойних фотографій

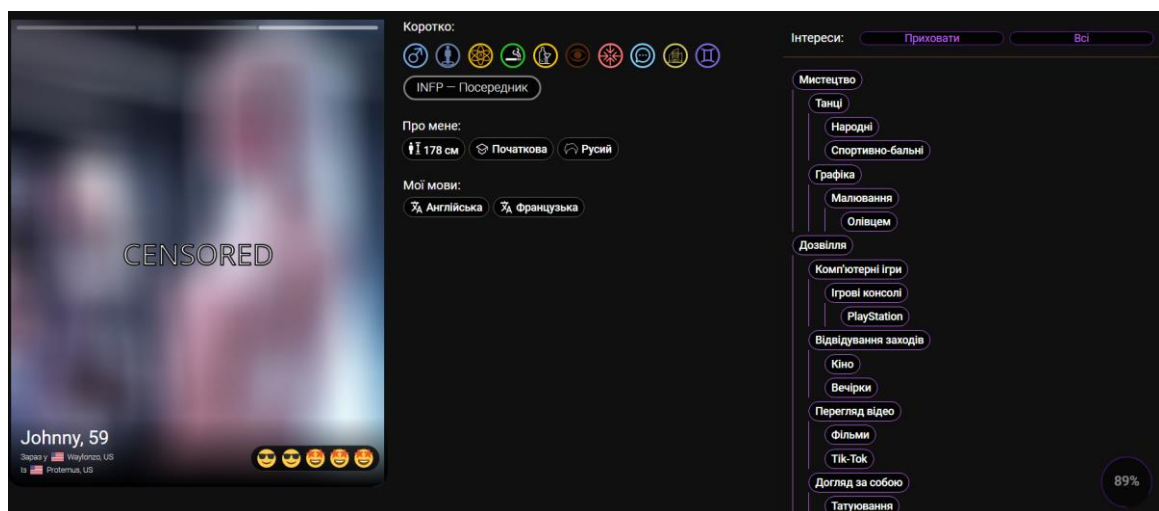


Рис. 42. Результат роботи алгоритму цензурування

2.4.4 Сповіщення за допомогою Telegram боту

При авторизації через Telegram, у бота авторизації з'являється можливість відправляти користувачеві повідомлення. Цим можна скористатися, надаючи можливість користувачеві отримувати важливі повідомлення напряму у Telegram.

Для реалізації функціоналу керування ботами існує безліч бібліотек, проте для вирішення цієї задачі з їх функціоналу необхідна лише одна функція, а саме, відправка повідомлень, знаючи ID чату (ми маємо цю інформацію, адже вона отримується при використанні OIDC). Тому було прийняте рішення реалізувати цю функцію самостійно, використовуючи документацію Telegram Bot API.

Відправка повідомлень у Telegram Bot API відбувається за допомогою GET запиту на URL «https://api.telegram.org/bot{TOKEN}/sendMessage». Це йде врозріз із семантикою REST API запитів, адже GET запити зазвичай використовуються для отримання інформації, а для створення повідомлення доцільніше було б використати метод POST; проте, завдяки GET запиту, його можна виконувати, просто перейшовши за URL, що є зручним у деяких випадках.

Для відправки запиту виконується GET request за допомогою бібліотеки Axios. Даний функціонал реалізовано у рамках TelegramService, що є складовою NotificationModule.

```

15 @Injectable()
16 export class TelegramService {
17     private readonly logger = new Logger(TelegramService.name);
18
19     async sendMessageByChatId(telegramUserId: number, messageMarkdown: string): Promise<boolean> {
20         this.logger.log(`Sending message to ${telegramUserId}`);
21
22         try {
23             await axios.get(
24                 `https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessage?chat_id=${telegramUserId}&text=${encodeURIComponent(
25                     messageMarkdown.replace(/([!])/g, "\\$1"),
26                     )}&parse_mode=MarkdownV2`,
27             );
28             return true;
29         } catch (e) {
30             this.logger.error(`Failed to send message to ${telegramUserId}: ${e}`);
31             return false;
32         }
33     }
34
35     async sendMessage(userToNotify: UniversalModel<UserModel>, messageMarkdown: string) {
36         if (!userToNotify.oAuth?.telegram?.id) {
37             this.logger.warn(`User ${userToNotify.id} has no telegram id`);
38             return false;
39         }
40
41         return this.sendMessageByChatId(userToNotify.oAuth.telegram.id, messageMarkdown);
42     }
43 }

```

Рис. 43. Код відправки сповіщення користувачу через Telegram бот

Для форматування тексту використовується формат Markdown, а саме, його реалізація MarkdownV2 у Telegram. Але, як можна побачити на стрічці 25 на Рис. 43, використання цього формату має деякі особливості, а саме, у тексті повідомлення не повинно бути неекраниваних символів «.» та «!». Тому перед відправкою повідомлення, воно пропускається через регулярний вираз, що виконує екранування цих спец. символів.

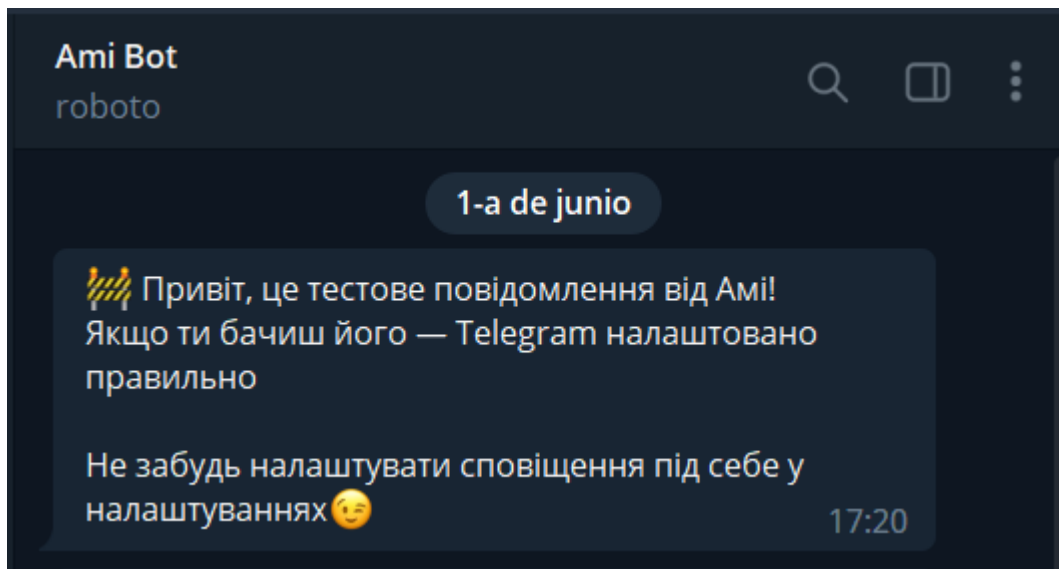


Рис. 44. Приклад сповіщення від Telegram бота

2.5 Створення Swagger документації

Документування API викликів є важливою частиною розробки серверного додатку. Проте, бібліотека Swagger окрім документації, також надає можливості зручного виклику Rest API у вигляді інтерактивної сторінки із підтримкою різних видів авторизації, тому було обрано саме її. Документування проводиться шляхом додавання декораторів із бібліотеки-адаптера `@nestjs/swagger`. Ці декоратори, зазвичай, починаються із `@Api`.

Для прискорення процесу розробки, було прийняте рішення додавати лише необхідні декоратори документації, такі як `@ApiTags` для групування схожих API, `@ApiQuery` для оголошення URL Query параметрів та декораторів помилок для методів, що викликають нестандартні помилки. Приклад використання декоратора `@ApiQuery` можна побачити на Рис. 45.

```

10  @Get("countries")
11  @Header("Cache-Control", "max-age=600")
12  @ApiQuery({ name: "locale", required: false, enum: ["en", "uk"] })
13  async getCountries(@Query("locale") locale = "en") {
14    return this.locationService.getCountries(locale);
15  }

```

Рис. 45. Приклад використання Swagger-декораторів

На Рис. 46 можна побачити приклад згенерованої документації. Схожі API згруповані разом, кожний шлях можна розгорнути та виконати запит, також, запити різних типів зручно виділені різними кольорами.

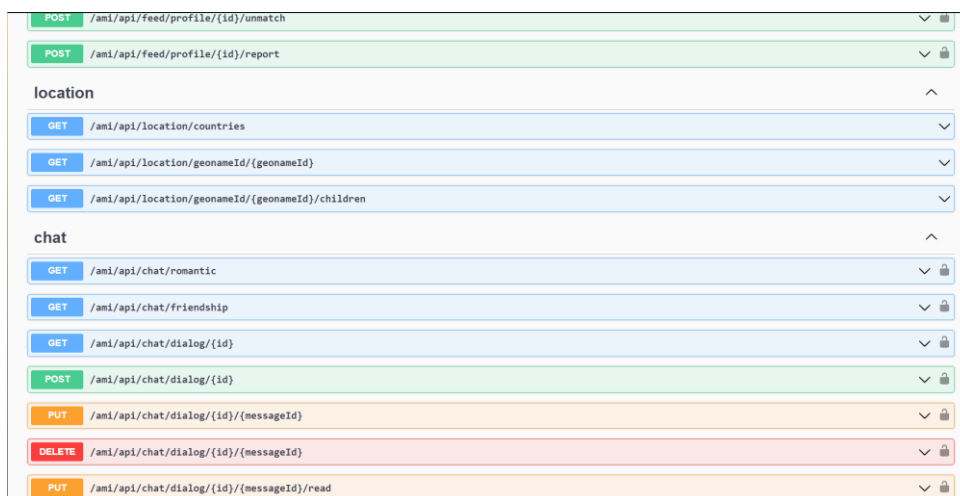


Рис. 46. Сторінка документації, згенерована Swagger

2.6 Адаптація клієнтської частини під роботу з сервером

Головною відмінністю при роботі з сервером є те, що стан знаходиться на стороні сервера та ми не можемо змінювати його напряму засобами Pinia (насправді, можемо, використовуючи прослуховування змін та виконуючи запити на зміну, проте, у такому випадку втрачається контроль та кількість запитів значно зростає через відправку окремого запита на кожну зміну, тому цей спосіб для вирішення задач синхронізації станів не розглядається), тому для зміни зовнішнього стану потрібно виконувати запити до API.

2.6.1 Робота з API

Для виконання таких запитів можна використовувати або стандартні можливості JavaScript (a same, XMLHttpRequest або Fetch API), або сторонні бібліотеки. Провівши аналіз стандартних засобів та бібліотеки Axios [34], було прийняте рішення щодо використання останньої, адже вона, окрім зручних методів для роботи та дуже зручної інтеграції із TypeScript, надає можливість централізованої обробки запитів та відповідей (щось схоже на механізм Middleware у Express.JS). Цей механізм зручно використовувати для відстеження

та обробки глобальних стандартних помилок, таких як помилка авторизації, коли токен стає неактуальним.

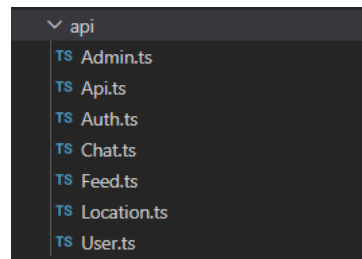


Рис. 47. Вміст папки src/api

Для зручності організації, усі API виклики згруповані у класи та винесені в окрему папку. Головним файлом є Api.ts, що виконує налаштування бібліотеки Axios та надає методи для виконання основних запитів (GET, POST, PUT та DELETE).

```

12  const request = axios.create();
13
14  request.interceptors.request.use((config) => {
15    const appStore = useAppStore();
16
17    const token = appStore.token;
18
19    if (token) {
20      config.headers["Authorization"] = `Bearer ${token}`;
21    }
22
23    config.headers["X-Ami-Mode"] = appStore.mode;
24    config.headers["X-Ami-Language"] = appStore.locale;
25    // You, 3 місяця назад • feat: api requests ...
26    return config;
27  });
28
29  const ERRORS_TO_IGNORE_TOAST = [IS_ERROR_FEED_NO_PROFILES];
30
31  request.interceptors.response.use(
32    (response) => response.data,
33    (error: AxiosError) => {
34      console.log("Axios Error", error);
35
36      const appStore = useAppStore();
37
38      if (error.code === "ERR_NETWORK") {
39        appStore.isOffline = true;
40        return;
41      }
42
43      if (error.response?.data) {
44        const { message } = error.response.data as {
45          message: string;
46          statusCode: number;
47        };
48
49        if (message === IS_ERROR_INVALID_TOKEN) { ...
50      } else if (message === IS_ERROR_BANNED) { ...
51      } else if (ERRORS_TO_IGNORE_TOAST.includes(message)) { ...
52      } else if (Array.isArray(message)) { ...
53      } else { ...
54      }
55    }
56  );
57
58  };

```

Рис. 48. Налаштування бібліотеки Axios

Для налаштування, як зображено на Рис. 48, використовується два interceptor-и: один на відправку запита і один на отримання відповіді. При відправці запиту, цей перехоплювач додає до заголовків токен авторизації користувача, поточний режим роботи додатку та обрану мову. Це дає можливість не прописувати заголовки кожного разу та дозволяє з легкістю проводити налагодження за необхідності. При отриманні відповіді від сервера, інший перехоплювач аналізує код відповіді, якщо це мережева помилка, він автоматично перемикає додаток в офлайн режим, якщо помилка про не валідний токен, стирає токен та переводить додаток на сторінку авторизації, у інших випадках, при помилці, повідомляє про неї користувачу.

```

74  export class Api {
75      static _apiUrl = config.apiUrl;
76
77      static get = <T = unknown>(url: string, options?: AxiosRequestConfig) =>
78          request.get<any, T>(`${config.apiUrl}/${url}`, options);
79
80      static post = <T>(url: string, data?: object, options?: AxiosRequestConfig) =>
81          request.post<any, T>(`${config.apiUrl}/${url}`, data, options);
82
83      static patch = <T = void>(url: string, data?: object) =>
84          request.patch<any, T>(`${config.apiUrl}/${url}`, data);
85
86      static put = <T>(url: string, data?: object) =>
87          request.put<any, T>(`${config.apiUrl}/${url}`, data);
88
89      static delete = <T = void>(url: string, data?: object) =>
90          request.delete<any, T>(`${config.apiUrl}/${url}`, { data });
91  }

```

Рис. 49. Методи для виконання запитів до сервера

Нижче, на Рис. 50, зображено приклад API-модуля, що відповідає за функціонування стрічки пошуку людей та використовує клас з Рис. 49 для виконання запитів.

```

11 export class FeedApi {
12   static getProfile = (id: string) =>
13     Api.get<ISFeedProfileDto>(`feed/profile/${id}`);
14
15   static getNextProfile = () => Api.get<ISFeedResponse>("feed/next");
16
17   // Unseen likes (to display the number under the feed icon)
18   static getLikes = (mode: AppMode) =>
19     Api.get<IWSLikeDto[]>(`feed/likes/${mode}`);
20
21   static likeProfileRaw = (dto: ICFedLike) =>
22     Api.post<ISLikedTodayDto>(`feed/like`, dto);
23
24   static likeProfile = (profileId: string, message?: string) =>
25     FeedApi.likeProfileRaw({ profileId, message, isLiked: true });
26
27   static dislikeProfile = (profileId: string) =>
28     FeedApi.likeProfileRaw({ profileId, isLiked: false });
29
30   static blockProfile = (profileId: string) =>
31     Api.post(`feed/profile/${profileId}/block`);
32
33   static unmatched = (id: string) => Api.post(`feed/profile/${id}/unmatch`);
34 }

```

Рис. 50. API-модуль на прикладі FeedApi

Як можна побачити на коді вище, використання такого підходу дає можливість використовувати TypeScript типізацію значень, що повертаються, а це, у свою чергу, допомагає боротися з помилками.

2.6.2 Спільні типи для клієнта та сервера

Під час спілкування клієнта з сервером варто пам'ятати, що вони повинні спілкуватися однією мовою, а під час внесення змін до API, іноді складно відслідкувати та внести ці зміни в усі місця, що були змінені. Також, на клієнті та сервері потрібно тримати дублікати деяких типів, які теж потрібно завжди тримати у актуальному стані. Для вирішення цих проблем та спрощення менеджменту типів, було прийняте рішення винести спільні типи у окремий репозиторій та підключати його як на клієнті, так і на сервері. Це дасть змогу легко перевіряти сумісність API, адже TypeScript буде показувати на помилки у випадках несумісності.

Реалізувати підключення репозиторію спільних типів можна двома базовими способами: у вигляді NPM пакету та у вигляді Git Submodule. Розглянемо детальніше кожен з цих варіантів.

NPM пакет це зручний спосіб організації коду коли потрібно чітко слідувати за версією, що використовується. У якості хостинга для пакетів можна використовувати як сам NPM, так і GitHub Packages або GitLab Packages. Цей спосіб було використано у виробничій практиці на тему створення системи керування Feature Flags та виявлено деякі недоліки: по-перше, ці типи потрібно компілювати перед публікацією, по-друге, необхідно налаштовувати доступ до приватного репозиторію NPM пакетів, по-третє, після кожного оновлення типів необхідно публікувати нову версію пакету та перевстановлювати його в усіх репозиторіях, в-четвертих, сама публікація пакету повинна виконуватися як етап CI/CD, що витрачає додаткові CI/CD хвилини на виконання. Тому у даній кваліфікаційній роботі було обрано використовувати другий спосіб.

Підключення спільного репозиторію типів у вигляді Git Submodule [35] виявилось дуже зручним, адже для його налаштування не потрібно публікувати додаткові пакети, не потрібно компілювати типи, можна у бідь-який момент перемикнутися на потрібний коміт, підключення типів відбувається у CI/CD конвеєрі під час збирання самих сервісів, тому цей процес не вимагає додаткових запусків CI/CD у репозиторії типів, а також, використання під-модулів дозволяє мати у них окремі гілки, що можуть використовуватися під час розробки нового API та не впливати на основну гілку. Також, не потрібно мати підключений репозиторій в окремій папці, адже він автоматично підключається до проекту клієнта та сервера, та оновлювати його можна з обох цих проектів.

Для реалізації підключення репозиторію у вигляді Git Submodule потрібно виконати наступні кроки:

1. Створити репозиторій у GitLab.
2. Виконати команду «git submodule add <https://...>» в обох репозиторіях.
3. Додати наступний код до файлу .gitlab-ci.yml (Рис. 51).

```

15   variables:
16   |   GIT_SUBMODULE_STRATEGY: recursive

```

Рис. 51. Увімкнення підтримки під-модулів для CI/CD

4. Увімкнути доступ до репозиторію типів за допомогою CI_JOB_TOKEN (у випадку, якщо репозиторій типів приватний) (Рис. 52).

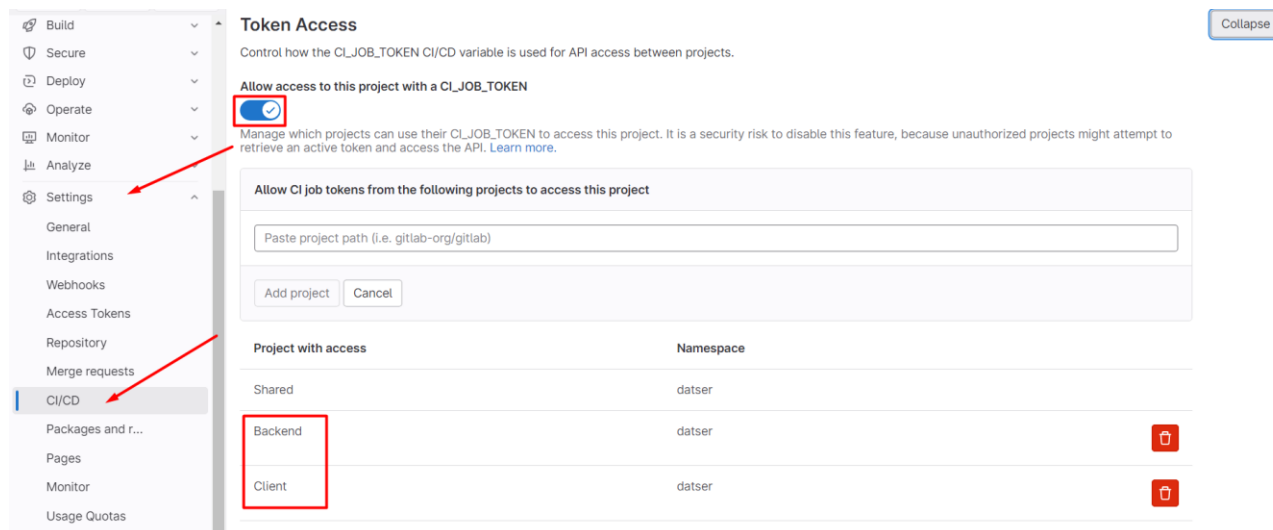


Рис. 52. Увімкнення доступу до репозиторію типів із клієнтського та серверного репозиторіїв

2.7 Налаштування Web Sockets

Для роботи технології Web Sockets потрібно налаштувати клієнт та сервер для правильної взаємодії. Для комунікації використовуємо бібліотеку Socket.io, що має як клієнтський, так і серверний NPM пакети.

2.7.1 Налаштування серверної частини

Для інтеграції Socket.io із Nest.js, нам знадобиться пакет @nestjs/websockets, а саме, декоратор @WebSocketGateway.

Після цього необхідно створити новий gateway (Рис. 53).

```

24 @WebSocketGateway({
25   path: '/api/ws',
26   namespace: '/main',
27 })
28 export class WsGateway implements OnGatewayDisconnect {
29   constructor(
30     @Inject(forwardRef(() => DatabaseService))
31     private readonly databaseService: DatabaseService,
32   ) {}
33
34   private readonly logger = new Logger(WsGateway.name);
35
36   private clients: Record<string, Socket[]> = {};
37   You, 2 місяця назад • feat: chat websockets receive me
38   private getClientsByUserId(userId: string): Socket[] {
39     return this.clients[userId] || [];
40   }
41
42   private registerClient(userId: string, client: Socket) {
43     if (!this.clients[userId]) this.clients[userId] = [];
44     this.clients[userId].push(client);
45   }
46
47

```

Рис. 53. Фрагмент файлу ws.gateway.ts

Та реалізувати механізм реєстрації нових клієнтів (Рис. 54). Після цього відкривається можливість відправляти повідомлення на під'єднані клієнти у реальному часі. Параметр path у декораторі @WebSocketGateway завдає шлях, за яким буде відбуватися HTTP запит рукописання [36]. Параметр namespace [37] дозволяє працювати з конкретним потоком подій. У нашому випадку використовується лише один потік – «/main».

```

48 @SubscribeMessage("identify")
49 protected async identify(@ConnectedSocket() client: Socket, @MessageBody() token: string) {
50   if (!token) return;
51
52   const user = await this.databaseService.authGetUserByToken(token);
53
54   if (!user) {
55     this.logger.warn(`New client with invalid token ${token}`);
56     client.emit("error", IW_ERROR_INVALID_TOKEN);
57     return client.disconnect(true);
58   }
59
60   client["token"] = token;
61   this.registerClient(user.id, client);
62
63   this.logger.debug(`New client connected! ${user.name}#${user.id}, client id: ${client.id}`);
64 }
65
66 handleDisconnect(client: Socket) {
67   this.logger.debug(`Client disconnected ${client.id}`);
68
69   Object.keys(this.clients).forEach((userId) => {
70     this.clients[userId] = this.clients[userId].filter((c) => c.id !== client.id);
71   });
72 }
73

```

Рис. 54. Фрагмент файлу ws.gateway.ts

2.7.2 Налаштування клієнтської частини

На клієнтській частині ми будемо використовувати бібліотеку socket.io напряду. Для зручності, створимо файл WebSockets.ts (Рис. 55), що являтиме собою Vue.js модуль та буде відповідати за взаємодію із сокетом.

Цей файл складатиметься із двох глобальних функцій: connect та disconnect. Перша з них відповідає за підключення до серверу (коли вже пройшов процес аутентифікації та в нас є токен користувача) та реєстрацію обробників подій, а друга – за розірвання з'єднання (при виконанні Logout запиту).

```
30 const socket = ref<Socket>(null as unknown as Socket);
31 You, 2 місяця назад • feat: chat websockets receive message
32 function connect(token: string) {
33   const url = WEBSOCKETS_URL;
34   const path = WEBSOCKETS_PATH;
35
36   const appStore = useAppStore();
37   const chatStore = useChatStore();
38   const feedStore = useFeedStore();
39   const userStore = useUserStore();
40   const router = useRouter();
41
42   socket.value = io(url, { path });
43
44   socket.value.on("connect", () => {
45     // console.log("Websocket connected", socket.value);
46     socket.value.emit("identify", token);
47     state.isConnected = true;
48   });
49
50   socket.value.on("disconnect", () => {
51     console.log("Websocket disconnected");
52     state.isConnected = false;
53   });
54
55 > socket.value.on("chat/message", (event: IWSChatMessageDto) => { ...
71   });
72
73 > socket.value.on("chat/message/edit", (event: IWSEditChatMessageDto) => { ...
```

Рис. 55. Фрагмент файлу WebSockets.ts

2.8 Налаштування CI/CD

Для збірки клієнтської та серверної частини додатку ми будемо використовувати GitLab CI/CD та самописний PHP скрипт для завантаження та розгортання отриманого архіву на сервер.

```

62 | You, на промові медіа + ci: production deployment script
63 | buildanddeployproduction:
64 |   stage: buildanddeployproduction
65 |
66 |   image: node:alpine
67 |
68 |   environment:
69 |     name: production
70 |     url: https://ami.project.li
71 |
72 |   variables:
73 |     GIT_SUBMODULE_STRATEGY: recursive
74 |
75 |   only:
76 |     variables:
77 |       - $CI_COMMIT_BRANCH == "production"
78 |
79 |   script:
80 |     - yarn
81 |     - yarn build
82 |     - apk --no-cache add curl zip
83 |     - cd dist
84 |     - mv ../package.json .
85 |     - zip -r -9 dist.zip . -x '*.d.ts' -x '*.js.map'
86 |     - "curl -F 'userfile=@./dist.zip' -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36' -H \"Authorization: $DEPLOY_TOKEN\" \"  
87 | .php\"

```

Рис. 56. Конфігурація для розгортання серверної частини

На Рис. 56 зображено етап збірки та розгортання серверної частини. Використовується Docker образ node:alpine, адже в ньому відсутні непотрібні пакети, що дає йому змогу розгортатися набагато швидше за Ubuntu збірку Node.js. Після встановлення пакетів та збірки серверної частини додатку, виконується її архівація та відправка до PHP скрипту розгортання. Скрипт перевіряє заголовки та, якщо, переданий токен авторизації співпадає із секретом сервера, виконує розархівування, встановлення пакетів залежностей та перезапуск додатку на сервері.

Ami > Backend > Pipelines

All 78 Finished Branches Tags Clear runner caches CI lint Run pipeline

Filter pipelines Show Pipeline ID ▾

Status	Pipeline	Triggerer	Stages
passed 00:01:32 50 minutes ago	chore: delete matches and chats on profile deletion #852766215 master → 5b3459ab		passed download ▾
passed 00:02:47 3 days ago	fix: bio can be empty #848520066 master → 4517696e		passed download ▾
passed 00:01:24 4 days ago	fix: some emojis have 5 chars #847014543 master → b3d5823a		passed download ▾
passed 00:01:21 5 days ago	feat: logging interceptor #846169812 master → 3811bc7f		passed download ▾
passed 00:01:21 5 days ago	fix: remove guard from account init route #846124759 master → 5b1154dc		passed download ▾

Рис. 57. Приклад виконання CI/CD задач для серверного репозиторію

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	Арк.
						80

На Рис. 57 зображено, як виглядають запуски CI/CD flow на сторінці GitLab репозиторію.

Аналогічним чином налаштована збірка клієнтської частини, проте у ній етапи збірки та розгортання були розділені на окремі етапи.

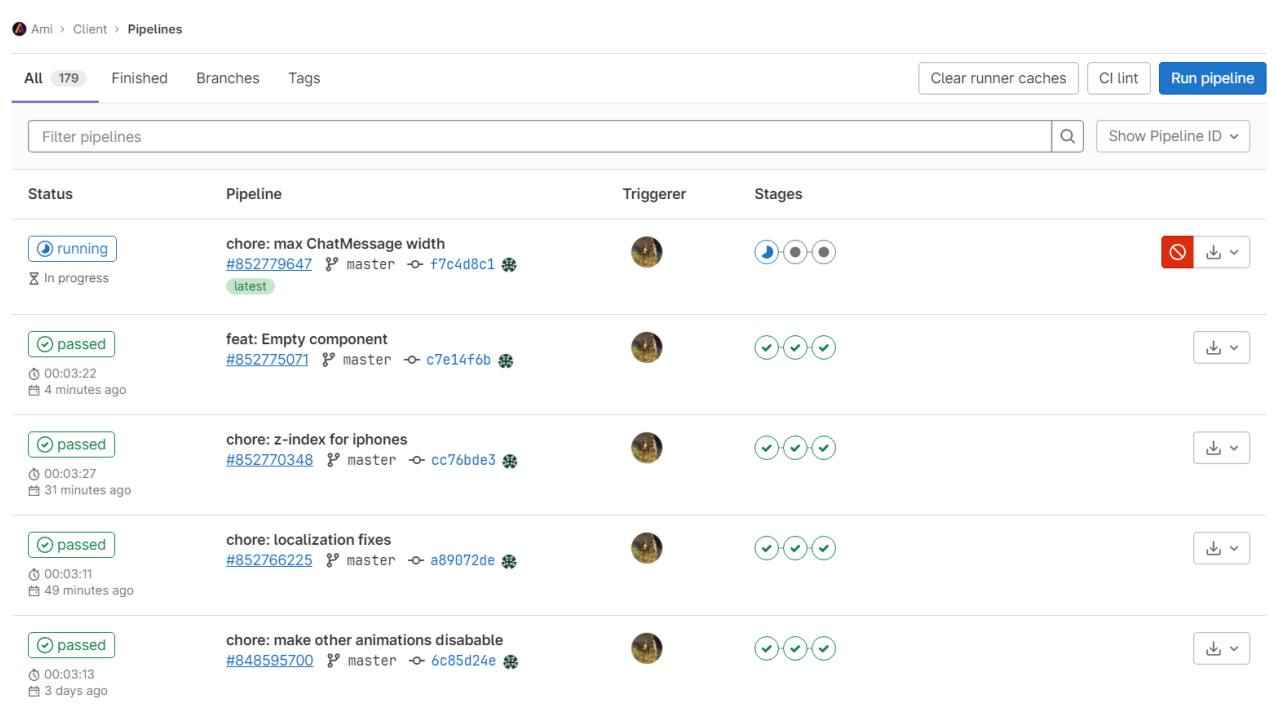


Рис. 58. Приклад виконання CI/CD задач для клієнтського репозиторію

Проте, згодом було вирішено об'єднати етапи збірки та розгортання клієнтської частини, адже виконання трьох окремих етапів займає набагато більше CI/CD хвилин (Рис. 59), надаючи із плюсів тише окрему індикацію для помилок у процесі збірки та завантаження.

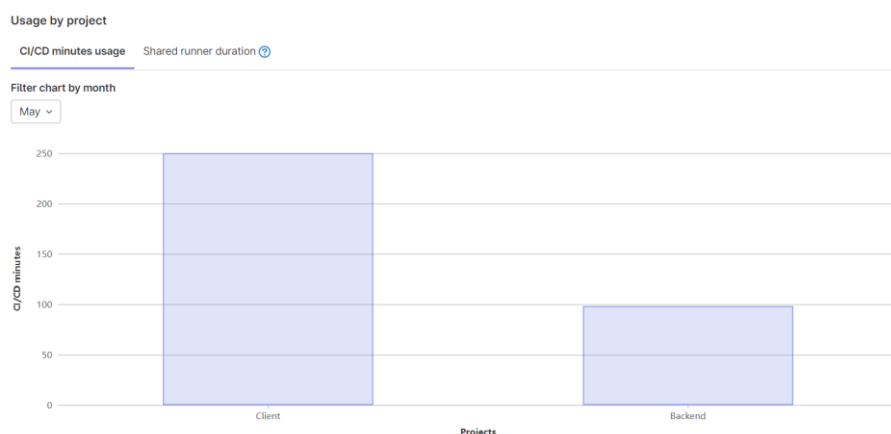


Рис. 59. Нерівномірне використання CI/CD хвилин

All 276	Finished	Branches	Tags	Clear runner caches	CI lint	Run pipeline
Filter pipelines				Q	Show Pipeline ID ▾	
Status	Pipeline	Triggerer	Stages			
passed 00:01:17 13 hours ago	fix: typings #887224623	production - 2281cd52	✓	↓ ▾		
passed 00:01:11 14 hours ago	fix: typings #887181469	master - 2281cd52	✓	↓ ▾		
passed 00:01:11 1 day ago	chore: update package name #885376056	master - e687c2cf	✓	↓ ▾		
passed 00:01:12 1 day ago	chore: update package name #885375920	production - e687c2cf	✓	↓ ▾		
passed 00:01:16 2 days ago	chore: temporary remove emojis sprite preload #884319819	production - 92571e95	✓	↓ ▾		
passed	chore: improve slide animations performance #884067978	production - ac259876	✓	↓ ▾		

Рис. 60. Приклад виконання CI/CD задач для клієнтського репозиторію після внесення змін

2.9 Налаштування серверу та бази даних

База даних знаходиться на тому ж самому фізичному сервері, що і серверна частина додатку, проте, розміщена всередині Docker контейнера.

У якості образу використовується `mongo:latest`, надаючи останню версію бази даних MongoDB.

Команда запуску контейнера виглядає наступним чином (замість Docker використовується його аналог – контейнеризатор Podman):

```
podman run --name ami-production
-e MONGO_INITDB_ROOT_USERNAME=логін
-e MONGO_INITDB_ROOT_PASSWORD=пароль
-v /mnt/hdd/ami-production:/data/db
-p 27017:27017 -d mongo
```

2.10 Налаштування кешування

2.10.1 Кешування статичних файлів

Для кешування статичних файлів використовуються можливості Apache. На Рис. 61 зображено вміст файлу `.htaccess` у корні сайту додатку, що виконує налаштування заголовків часу кешування для різних типів файлів.

```
9 # Кешування файлів
10 ExpiresActive On
11 ExpiresDefault A0
12 ExpiresByType image/gif A604800
13 ExpiresByType image/png A604800
14 ExpiresByType image/jpeg A604800
15 ExpiresByType image/webp A604800
16 ExpiresByType image/svg+xml A86400
```

Рис. 61. Налаштування кешування статичних файлів у `.htaccess`

Числа після «А» вказують термін життя кешу у секундах, використовуючи Apache модуль `mod_expires` [38].

2.10.2 Кешування API запитів

Деякі запити завжди повертають однакові значення, наприклад, це стосується API отримання інформації про доступні локації. Для таких випадків ми використовуємо декоратор `@Header` для встановлення часу дії кешу у NestJS контролері. На Рис. 62 зображено приклад такого використання.

```
5 @Controller("location")
6 @ApiTags("location")
7 export class LocationController {
8   constructor(private readonly locationService: LocationService) {}
9
10  @Get("countries")
11  @Header("Cache-Control", "max-age=600")
12  @ApiQuery({ name: "locale", required: false, enum: ["en", "uk"] })
13  async getCountries(@Query("locale") locale = "en") {
14    return this.locationService.getCountries(locale);
15  }
16
17  @Get("geonameId/:geonameId")
18  @Header("Cache-Control", "max-age=600")
19  @ApiQuery({ name: "locale", required: false, enum: ["en", "uk"] })
20  async getGeonameId(@Param("geonameId") geonameId: number, @Query("locale") locale = "en") {
21    return this.locationService.getByGeonameId(geonameId, locale);
22  }
23
24  @Get("geonameId/:geonameId/children")
25  @Header("Cache-Control", "max-age=600")
26  @ApiQuery({ name: "locale", required: false, enum: ["en", "uk"] })
27  async getChildrenByGeonameId(@Param("geonameId") geonameId: number, @Query("locale") locale = "en") {
28    return this.locationService.getChildrenByGeonameId(geonameId, locale);
29  }
30 }
```

Рис. 62. Приклад налаштування кешування API запитів

2.10.3 Обхід обмежень iOS

Також, кеш можна використовувати для передачі інформації між сайтом та встановленим PWA додатком у операційних системах з ізольованими сховищами, таких як iOS. При використанні прогресивних додатків у iOS можна стикнутися із проблемою ізольованості сховищ, адже localStorage, sessionStorage, cookies та інші види веб-сховищ створюються окремо для сайту та кожного із встановлених PWA додатків, навіть, якщо вони встановлювались із одного маніфесту. Це доволі серйозна проблема, адже нам потрібно реалізувати передачу ID користувача у рамках реферальної системи.

Це обмеження можна обійти, використовуючи кеш, адже закешовані запити діляться між усіма додатками зі спільним доменом. На Рис. 63 зображено обхід обмежень, що був використаний для передачі ID користувача, що запросив людину у PWA додаток.

```
22 @Get("safari-workaround")
23 @Header("Cache-Control", "public, max-age=604800")
24 @ApiHeader({ name: "x-ref", required: true })
25 safariPassRefByCache(@Req() req: Request) {
26     /**
27      * This is a workaround for iOS Safari, which doesn't allow
28      * to share localStorage/cookies/etc between the tab and PWA.
29      * The only thing that is shared is the cache.
30      *
31      * So, we use this endpoint to pass the referral id using the cache.
32      * The referral id is set by the header.
33      */
34
35     const ref = req.headers["x-ref"] as string;
36
37     if (!ref) throw new BadRequestException("No referral id provided");
38
39     return { ref };
40 }
```

Рис. 63. Код для передачі referralId у PWA додаток на iOS

На головній сторінці відбувається завантаження ID користувача у кеш через передачу його у полі заголовку x-ref до серверу (Рис. 64).

```

12  async function safariReferralWorkaround(referralId) {
11  console.log('Using workaround to pass referral id by cache on Safari')
10  const response = fetch("https://ami.project.li/app/api/auth/safari-workaround", {
9    "headers": {
8      "x-ref": referralId
7    },
6    "method": "GET",
5  });
4  }

```

Рис. 64. Збереження referralId за допомогою кешу

А потім на клієнтській частині виконується читання цього ID (Рис. 65).

```

151 const loadReferralUser = async () => {
152   referralUser.value =
153     (localStorage.getItem(LOCALSTORAGE_KEY_REFERRAL) ||
154       new URLSearchParams(window.location.search).get("ref")) ??
155     undefined;
156
157   if (!referralUser.value) {
158     console.log("No referral user found, trying iOS workaround");
159
160     const response = await AuthApi.safariWorkaround();
161
162     if (response.ref !== "null") {
163       referralUser.value = response.ref;
164     } else {
165       console.log("No referral user found");
166     }
167   }
168 };
169
170 onMounted(() => {
171   loadReferralUser();
172 });

```

Рис. 65. Читання referralId на стороні PWA додатку

При читанні поля, у заголовку x-ref передається значення “null”, що дає змогу перевірити, чи було збережено це поле у кеші.

```

10  static safariWorkaround = () =>
11    Api.get<{ ref: string }>("auth/safari-workaround", {
12      headers: {
13        "x-ref": "null",
14      },
15    });
16  }

```

Рис. 66. Метод AuthApi.safariWorkaround

2.11 Налаштування Phusion Passenger

Для запуску серверної частини додатку, балансування навантаження та забезпечення інтеграції з Apache, використовується рішення Phusion Passenger.

Це дає змогу використовувати як Apache, так і Node.js API на одному домені та використовувати SSL сертифікат Apache для Node.js додатку.

Для налаштування інтеграції Phusion Passenger із Apache використовується файл .htaccess.

```
16 RewriteEngine On
15 RewriteCond %{HTTPS} off
14 RewriteRule (.*?) https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
13 Require all granted
12
11 # Passenger
10 PassengerEnabled on
9 PassengerAppType node
8 PassengerAppRoot /home/ami_prod/src
7 PassengerStartupFile main.js
6 PassengerFriendlyErrorPages off
5 PassengerStickySessions on
4 PassengerAppGroupName ami_prod
3 PassengerRestartDir /home/ami_prod/
2 PassengerHighPerformance off
1 PassengerNodejs /usr/bin/node
```

Рис. 67. Налаштування Phusion Passenger для інтеграції із Apache

На початку файлу виконується налаштування автоматичної переадресації із незахищеної http-версії сайту на захищену https.

Далі відбувається увімкнення та налаштування Phusion Passenger. Параметр PassengerFriendlyErrorPages відключає детальну інформацію про помилки, адже вона містить конфіденційну інформацію, таку як налаштування сервера та змінні оточення. Параметр PassengerStickySessions потрібен для коректної роботи протоколу Web Sockets, адже він прив'язує сесію до конкретного процесу Node.js.

2.12 Progressive Web Application

Progressive Web Application дає такі можливості як персональна іконка на робочому столі із можливістю додати лічильник, завантаження в повноекранному режимі без елементів навігації браузера, зручне керування кешуванням та можливість прийому Push Notifications [39].

2.12.1 Web Manifest

Маніфест у контексті PWA це JSON файл, що описує метадані додатку. У ньому міститься така інформація як назва додатку, його іконки, опис, теги, режим відображення, колір фону та інші корисні дані, необхідні для встановлення додатку.

Для спрощення генерації маніфесту та Service Worker, було використано бібліотеку WorkBox та модуль VitePWA [40] для зручної інтеграції.

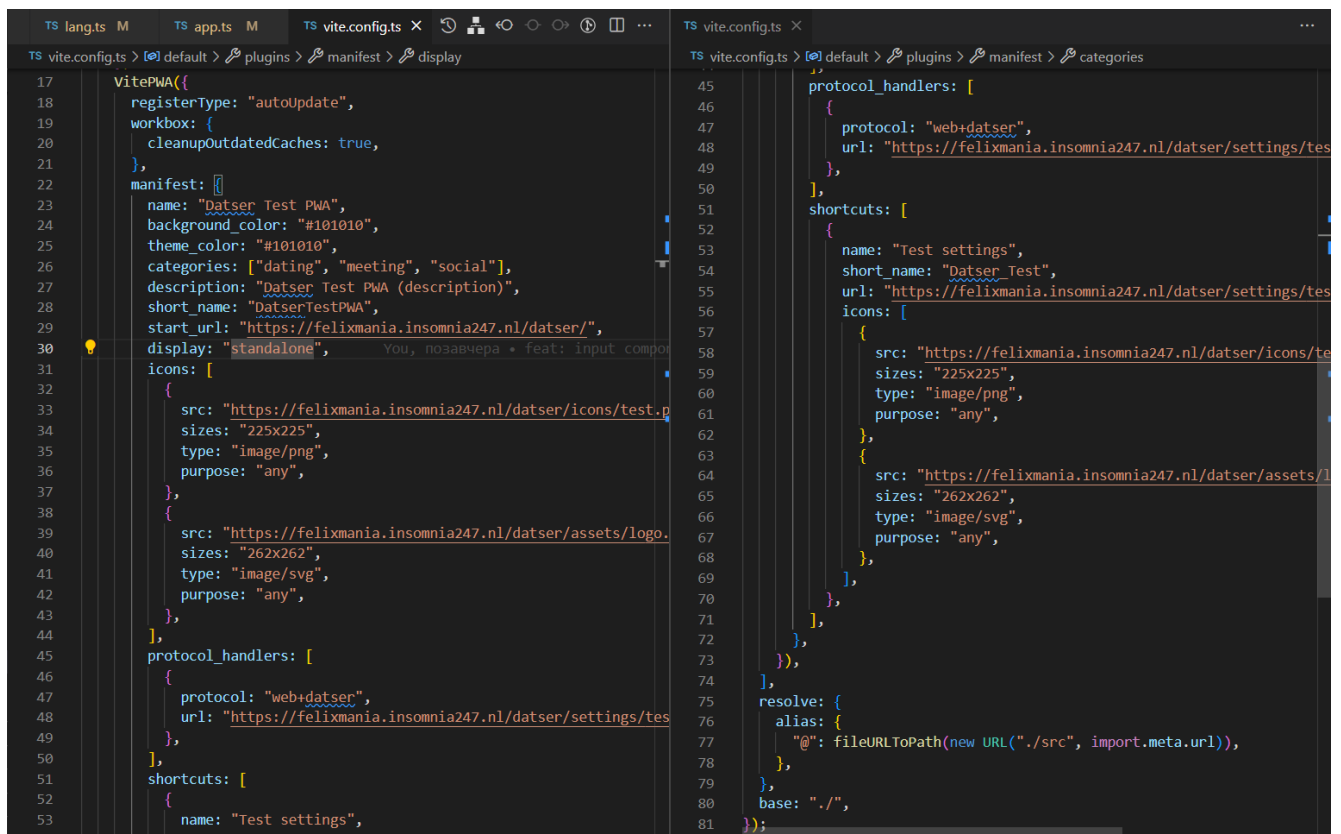


Рис. 68. Конфігурація модуля VitePWA

Модуль VitePWA є надбудовою, що інкапсулює у собі як генерацію маніфесту, так і налаштування генерації Service Worker.

При налаштуванні маніфесту потрібно вказати ім'я ярлика додатку, необхідні іконки, та кольори оформлення для того, щоб телефон міг налаштувати колір своїх елементів, таких як верхня шторка, під кольори додатку для більш кращої візуальної інтеграції.

Налаштування WorkBox необхідне лише тоді, коли планується використовувати власноруч написані Service Worker, наприклад, для отримання Push-повідомлень. Проте, для кешування, в нашому випадку вистачає стандартної генерації Service Worker на основі конфігурації.

2.12.2 Кешування

WorkBox надає дуже широкі можливості для реалізації різних стратегій [41] кешування, починаючи від завчасно заготовлених «рецептів» (наприклад, для автоматичного кешування Google Fonts) до можливостей для написання власної стратегії.

Серед можливих варіантів, для додатку було обрано дві стратегії кешування: Cache-First та Stale-While-Revalidate, адже вони, також, дозволяють обмежено використовувати додаток без підключення до інтернету.

Перша стратегія отримує дані з серверу при першому запиті та заносить їх до кешу. При наступних запитах дані беруться із кешу та фактичних запитів не виконується. Це дуже зручна стратегія для роботи зі статичними ресурсами, вона дозволяє значно пришвидшити очікування виконання запитів додатку та зменшити об'єм мережевого трафіку, адже, статичні дані беруться із кешу миттєво, а полоса пропускання мережі та трафік витрачаються лише на динамічні запити. Ця стратегія використовується для кешування файлів, прямо необхідних для роботи додатку (index.html, *.css, *.js та *.svg файли збірки).

Друга стратегія працює схожим чином, проте після того, як дані взяли із кешу, відправляє у фоні запит на їх оновлення. Цю стратегію зручно використовувати для отримання великих динамічних даних, які змінюються не дуже часто, наприклад дерева інтересів.

Також, було розглянуто можливість використання стратегії Network-First для деяких динамічних запитів, наприклад, отримання інформації про профіль та чати. За цією стратегією, додаток спочатку намагається отримати дані з серверу, проте, якщо мережа недоступна, дані беруться, у разі наявності, із кешу. Це

дозволило б повноцінно використовувати додаток у режимі офлайн. Проте, це також могло породити безліч проблем, пов'язаних із частковим оновленням даних та втрати синхронізації між різними взаємопов'язаними частинами додатку (наприклад, кеш чату може оновитися пізніше та посилатися на ID профіля, якого ще не існує у кеші профілів). Тому на даному етапі було обрано відмовитися від використання цієї стратегії та замість відображення неактуальних даних, просто повідомляти користувача про те, що мережа недоступна.

2.12.3 Власний плагін для Vite

Як згадувалося у розділі 2.12.1, функціоналу, згенерованого Service Worker, в плані кешування, повністю вистачає для задач даної кваліфікаційної роботи, проте, також, необхідно реалізувати функціонал Push-сповіщень. А плагін VitePWA підтримує лише два режими роботи: або Service Worker повністю генерується бібліотекою WorkBox за складеною конфігурацією, або Service Worker цілком пишеться розробником з нуля (використовуючи функції WorkBox).

Проте, реалізація Service Worker з нуля є недоцільною у рамках даної кваліфікаційної роботи, тому було прийняте рішення написати власний плагін для збиральника Vite, що буде викликатися після виконання VitePWA (що генерує Service Worker за конфігурацією) та додавати в кінець цього файлу вміст файлу service-worker.js із кореня проекту.

Ще одною причиною, чому не було обрано шлях написання Service Worker з нуля є те, що отриманий Service Worker не має доступу до змінних оточення під час збирання. А змінні оточення грають величезну роль у даному додатку, адже через них налаштовуються різні гілки у рамках CI/CD. Наприклад, бета-гілка та production-гілка мають зовсім різні домени, використовують різні боти, мають різні ключі та додаткові налаштування, тому маніфест та Service Worker повинні генеруватися, приймаючи до уваги, для якого середовища виконується збирання. Налаштування конфігурації для генерації Service Worker виконується

у файлі vite.config.ts, з якого у процесі збирання можна отримати доступ до змінних оточення, саме це стало ще одною причиною пошуку спроб залишитися на шляху генерації.

Приймаючи до уваги проблему, що впливає з попереднього абзацу було прийняте рішення додати до написаного плагіну можливість передавати змінні оточення до скрипта перед додаванням його у кінець згенерованого Service Worker.

```
32 export function ServiceWorkerPatcherPlugin(  
33   inputFile: string,  
34   outputFile: string  
35 ): Plugin {  
36   // Copyright (c) Illia Kliuiev 2023  
37   return {  
38     name: "vite-plugin-patch-service-worker",  
39     enforce: "post",  
40     apply: "build",  
41     closeBundle: {  
42       sequential: true,  
43       async handler() {  
44         const inputContent = await readFile(  
45           inputFile.replace(/\\/g, "/"),  
46           "utf8"  
47         );  
48  
49         const safeEnv: Record<string, string> = {};  
50  
51         for (const [key, value] of Object.entries(process.env)) {  
52           if (key.startsWith("VITE_") && value) safeEnv[key] = value;  
53         }  
54  
55         const envPatch = `SW_ENV = ${JSON.stringify(safeEnv)};\n`;  
56  
57         await appendFile(  
58           outputFile.replace(/\\/g, "/"),  
59           envPatch + inputContent,  
60           "utf8"  
61         );  
62       },  
63     },  
64     async buildEnd(error) {  
65       if (error) throw error;  
66     },  
67   };  
68 }
```

Рис. 69. Плагін для ін'єкції власного коду до Service Worker

Вище зображено код плагіну, що вміщується у 36 стрічок форматowanego коду. На стрічці 52 можна побачити те, що не всі змінні оточення передаються до Service Worker, адже ці дані можна буде побачити на клієнті, а змінні оточення можуть містити конфіденційну інформацію. Префікс «VITE_» вже

використовується однойменним збиральником для змінних, до яких додаток може мати доступ, тому було обрано його використання і для змінних, що передаються до Service Worker.

2.12.4 Push Notifications

Відправка персоналізованих повідомлень за допомогою технології Web Push є зручним способом показати користувачеві, що у додатку щось змінилося. Раніше такі можливості були лише для додатків, що розроблялися спеціально для мобільних платформ, проте зараз, за допомогою Web Push, можна відправляти повідомлення прямо із веб-додатків відразу на всіх платформах.

Робота з Push-повідомленнями складається з чотирьох складових. Основний потік браузера, Service Worker, серверна частина додатку та сервер Push-повідомлень. Основний потік браузера повинен отримати права на відображення сповіщень за допомогою методу «Notification.requestPermission()», оформити підписку на Push-повідомлення через звернення до Service Worker та відправити отриманий URL та ключі на сервер додатку. Далі Service Worker підписується на подію «push» та реалізує код парсингу отриманих з серверу додатку даних та виводу їх у повідомлення за допомогою методу «self.registration.showNotification(...)». Далі сервер додатку, зберігає отриманий від клієнта URL та ключі доступу у базу даних та прив'язує їх до конкретного користувача, що їх відправив. Коли сервер додатку хоче відправити повідомлення конкретному пристрою конкретного користувача, він бере з бази URL серверу Push-повідомлень та відправляє йому запит, шифруючи повідомлення ключами, що отримав від клієнта. Далі браузер клієнта робить опитування сервера Push-повідомлень та у випадку виявлення нових повідомлень для цього пристрою, надсилає подію «push», яку перехоплює Service Worker та відображає повідомлення користувачу.

```

17  /**
18   *
19   * @param {PushEvent} event
20   * @returns
21   */
22  async function handlePushEvent(event) {
23    if (!(self.Notification && self.Notification.permission === "granted")) {
24      console.warn(
25        "[Service Worker] Notifications aren't supported or permission not granted!"
26      );
27      return;
28    }
29
30    const defaultMessage = {
31      title: "Ami",
32      body: "Hello!",
33      icon: "https://ami.project.li/logo.svg",
34      badge: "https://ami.project.li/logo.svg",
35      vibrate: [200, 0, 200, 0],
36    };
37
38    const message = Object.assign(defaultMessage, JSON.parse(event.data.text()));
39
40    self.registration.showNotification(message.title, message);
41
42    self.navigator.setAppBadge(1);
43  }
44
45  // Register event listener for the 'push' event.
46  self.addEventListener("push", (event) => {
47    // Keep the service worker alive until the notification is created.
48    event.waitUntil(handlePushEvent(event));
49  });
50
51  self.addEventListener("notificationclick", (event) => {
52    event.preventDefault();
53
54    const notification = event.notification;
55
56    self.navigator.clearAppBadge();
57    clients.openWindow(APP_URL);
58    notification.close();
59  });
60
61  console.log("Ami's Service Worker Loaded...");

```

Рис. 70. Налаштування Service Worker для роботи з Push Notifications

Також, на стрічці 42 Рис. 70 можна побачити ще одну можливість Progressive Web Application, а саме, можливість додавати лічильник біля іконки на головному екрані. Це відбувається за допомогою методів «» та «». У даному прикладі відображається число 1, адже для точного відображення кількості нових подій потрібно виконувати окремі запити та синхронізувати стан з іншими сеансами користувача. Цей функціонал оформлено у вигляді задачі на майбутнє та переміщено у Backlog.

На Рис. 71 зображено декілька методів класу PushNotifications, що використовується для підписки на повідомлення та відписки від них. Метод register викликається після отримання прав на показ сповіщень, а метод unregister – коли користувач відмовляється від підписки у налаштуваннях.

```

145 export class PushNotifications {
146   static isSupported() {
147     return Boolean(
148       "serviceWorker" in navigator && "PushManager" in window && navigator.serviceWorker.controller
149     );
150   }
151   // You, 6 дней назад • chore: push messages test playground ...
152   static async isRegistered() {
153     const registration = await navigator.serviceWorker.ready;
154     const subscription = await registration.pushManager.getSubscription();
155
156     return Boolean(subscription);
157   }
158   static async register() {
159     const registration = await navigator.serviceWorker.ready;
160
161     if (await registration.pushManager.getSubscription()) {
162       Modal.toast("Already subscribed to push notifications");
163       return false;
164     }
165
166     const subscription = await registration.pushManager.subscribe({
167       userVisibleOnly: true,
168       applicationServerKey: PUSH_SERVER_PUBLIC_KEY,
169     });
170
171     await UserApi.subscribePush(subscription);
172
173     return true;
174   }
175   static async unregister() {
176     const registration = await navigator.serviceWorker.ready;
177     const subscription = await registration.pushManager.getSubscription();
178     if (subscription) {
179       await UserApi.unsubscribePush(subscription);
180       await subscription.unsubscribe();
181       return true;
182     } else {
183       Modal.toast("Not subscribed to push notifications");
184     }
185     return false;
186   }
187 }
188
189

```

Рис. 71. Клас для підписки та відписки Push Notifications

Проте, відправка повідомлень значно обмежена. По-перше, для того, щоб навіть встановлений додаток мав можливість відправляти сповіщення на пристрій користувача, користувач повинен надати права цьому додатку. Якщо ж, навіть випадково, користувач натисне на кнопку відміни, додаток більше не матиме можливості запитати доступ на відправку сповіщень, єдина можливість, яку пропонує документація, це показати користувачу покрокову інструкцію, як у налаштуваннях браузера дозволити сповіщення для домену, якому належить додаток. Це дуже незручно, адже браузерів існує велика кількість для різних платформ та інструкції для них можуть значно відрізнятися. Тому для попередження випадкового блокування повідомлень було вирішено спочатку запитати у користувача дозвіл за допомогою внутрішнього модального вікна додатку, а лише потім показувати системне вікно підтвердження. Проте, навіть

такий спосіб не гарантує, що користувач може випадково заблокувати повідомлення.

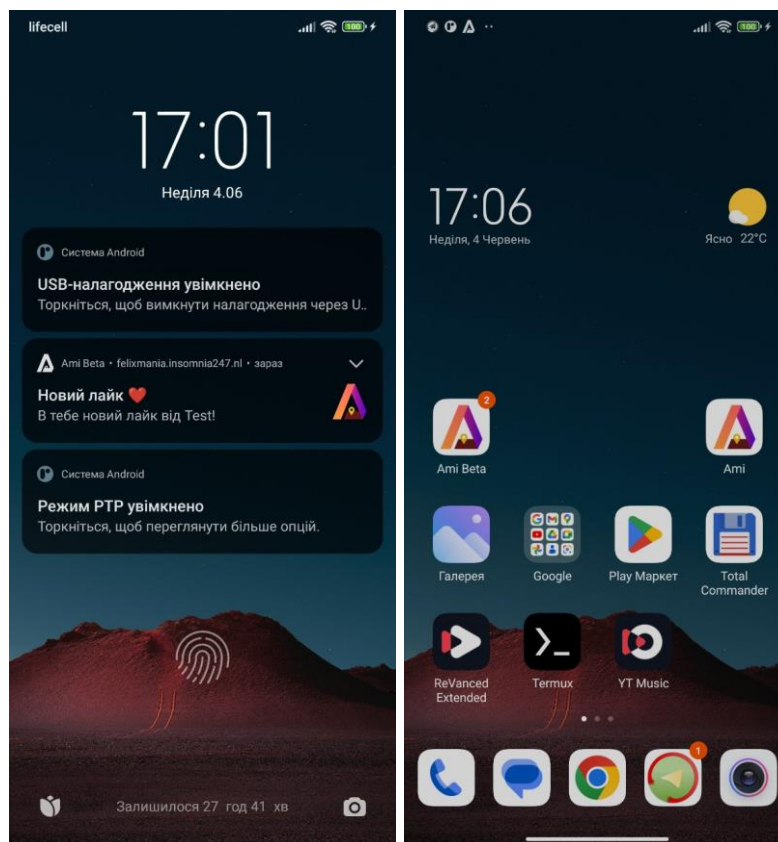


Рис. 72. Приклад Push сповіщення та лічильника біля іконки додатку

На Рис. 72 зображено тестове Push повідомлення. Білий контур у верхньому-лівому робиться через поле badge, а кольорове зображення справа – через поле icon. У майбутньому планується відображати там аватар людини замість логотипу додатку.

Висновок до другого розділу

У цьому розділі було виконано та описано процес розробки додатку, починаючи від проектування графічного інтерфейсу та розробки прототипу, закінчуючи розробкою серверної частини, проектуванням бази даних, налаштуванням кешування, створенням документації, налаштуванням серверу та створенням Web Manifest та Service Worker для Progressive Web Application.

У наступному розділі пояснювальної записки буде представлено отриманий додаток, а також, описано інструменти для його супроводу.

РОЗДІЛ 3. ОГЛЯД ТА СУПРОВІД ГОТОВОГО ПРОДУКТУ

3.1 Огляд функціоналу готового додатку

У цьому розділі ми зробимо огляд функціоналу готового додатку з точки зору кінцевого користувача.

Першим екраном, який бачить користувач при першому запуску додатку є екран вибору мови, із встановленою мовою, в залежності, від налаштувань браузера.

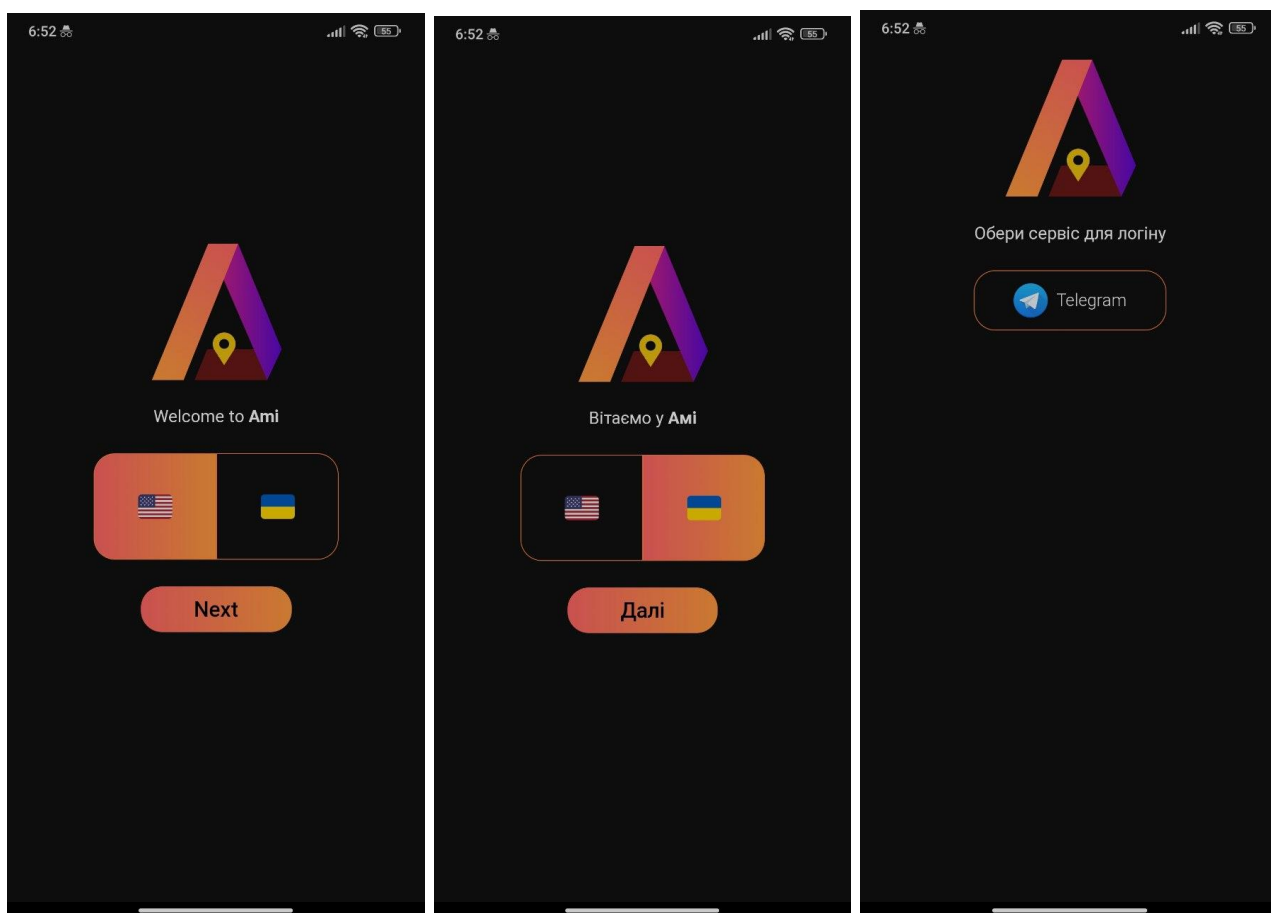


Рис. 73. Сторінка вибору мови (1, 2) та авторизації (3)

Після вибору сервісу авторизації, відкривається спливаюче вікно Telegram, що запитує, чи дійсно користувач хоче авторизуватися у цьому додатку. Після підтвердження відбувається перехід на наступну сторінку, в залежності від того, чи зареєстрований вже користувач у системі.

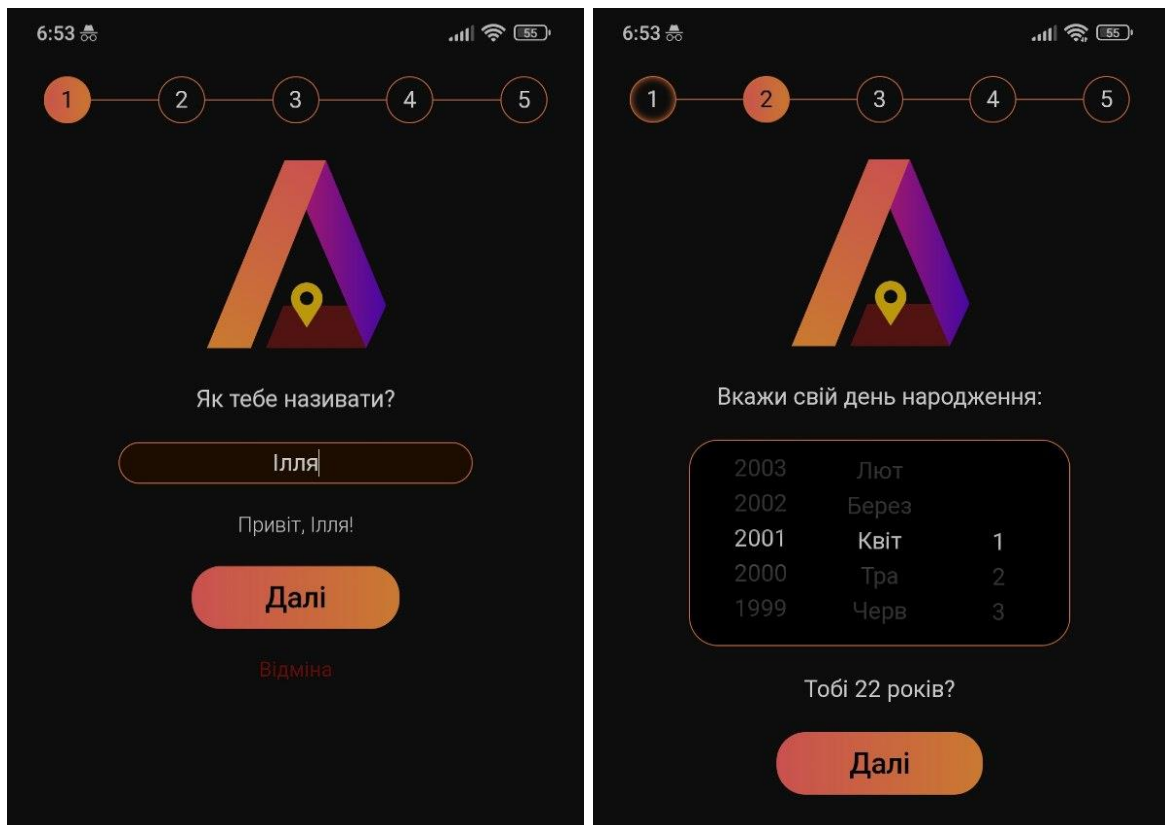


Рис. 74. Вказання імені та дати народження під час реєстрації

Якщо користувач ще не зареєстрований, відкриється процес реєстрації, зображений на Рис. 74. Червона кнопка «Відміна» миттєво видаляє новостворений акаунт та повертає користувача на сторінку авторизації. Це потрібно для випадків, коли користувач випадково використав неправильний акаунт для авторизації.

На першому екрані пропонується ввести ім'я, а на другому – дату народження. Знизу відображається вік людини для перевірки, т, якщо дата народження співпадає з поточною датою, відображається привітання. Якщо вік користувача менше 14 років, подальшої реєстрації не відбувається. Після натискання на кнопку «Далі» відкривається вікно підтвердження дати народження з попередженням, що цю інформацію неможливо буде змінити у майбутньому.

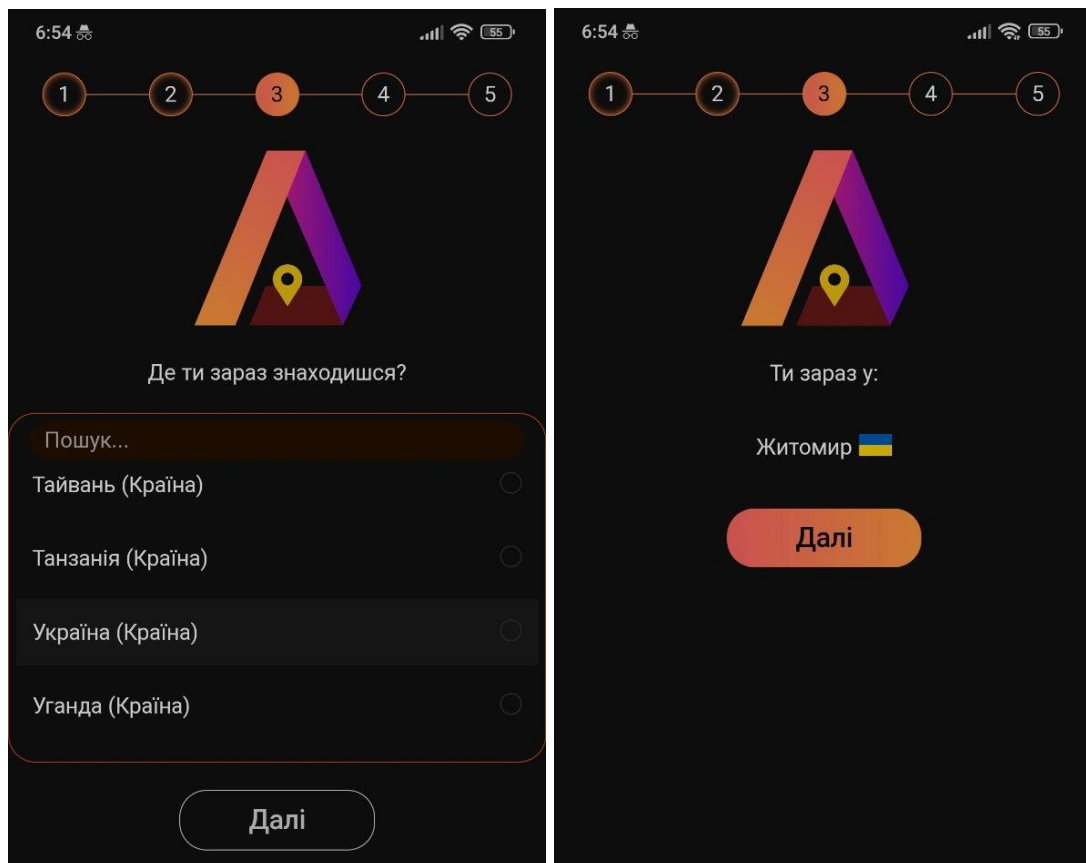


Рис. 75. Вказання поточного місця проживання

На наступному екрані відбувається вказання поточного місця проживання. Інформація береться з ресурсу GeoNames [42] для мови, що була вказана користувачем на першому етапі, проте, для української мови може бути відсутня локалізація деяких міст та селищ.

Вказання місця проживання відбувається у декілька етапів з поступовим уточненням країни, області, району та міста/селища. Російська Федерація відсутня у списку поточного місця проживання через військове вторгнення в Україну, проте, доступна у списку місця народження, адже людина не може обрати, де народитися.

Після вибору кінцевого пункту проживання, відображається його назва поруч із прапором країни. При натисканні на цей текст процес починається спочатку.

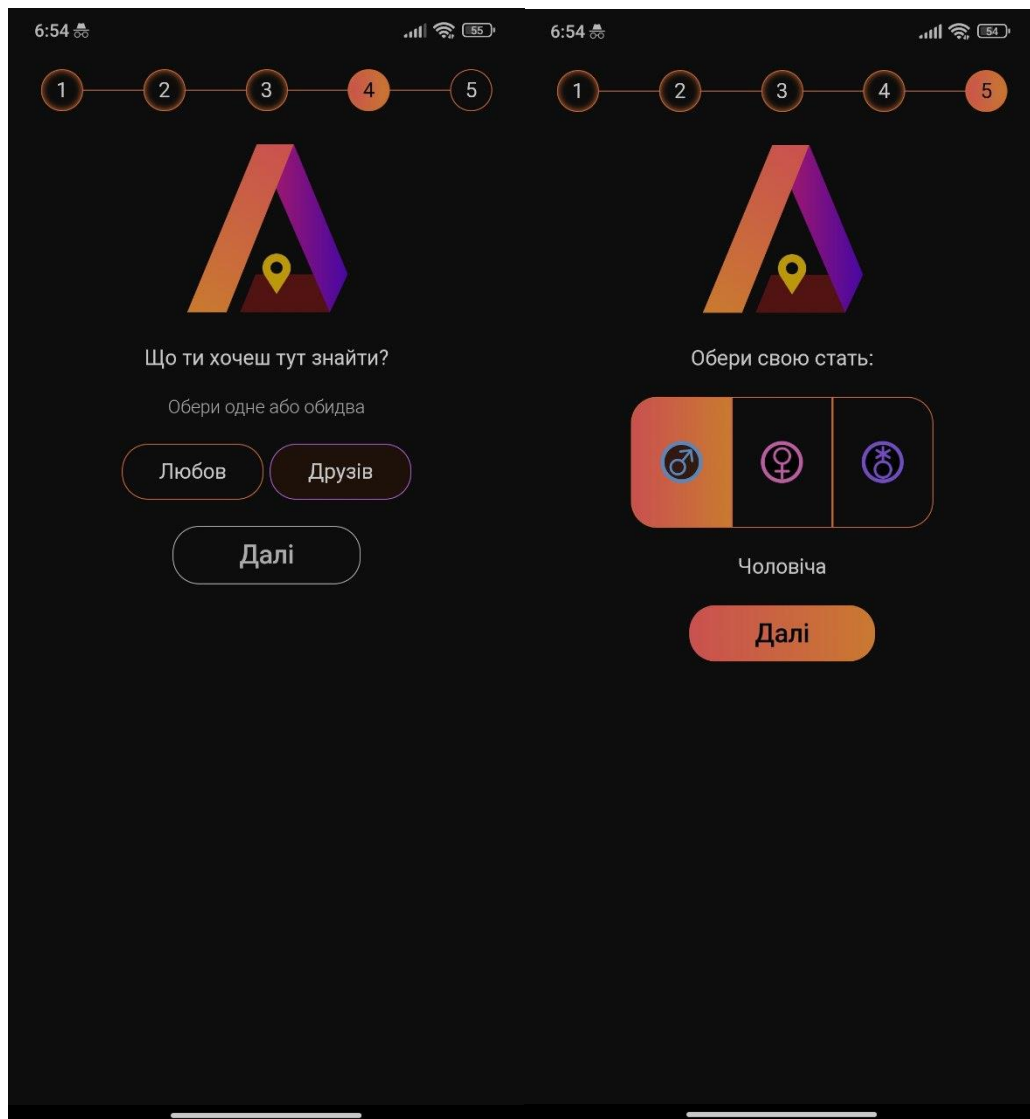


Рис. 76. Вибір бажаних профілів та вказання статі

Наступним кроком пропонується обрати, які профілі користувач хоче створити. У випадку, якщо вік користувача менше 16 років, даний етап пропускається та автоматично обирається створення лише Дружнього профілю, адже Романтичний режим доступний лише з 16 років.

На наступному кроці пропонується обрати стать, після чого йде заповнення обраних профілів у спрощеному вигляді та вказання інтересів. Для економії місця, описання цих кроків буд опущено, проте, звичайне заповнення профілю та інтересів буде показано у наступних пунктах.

3.1.1 Заповнення профілю

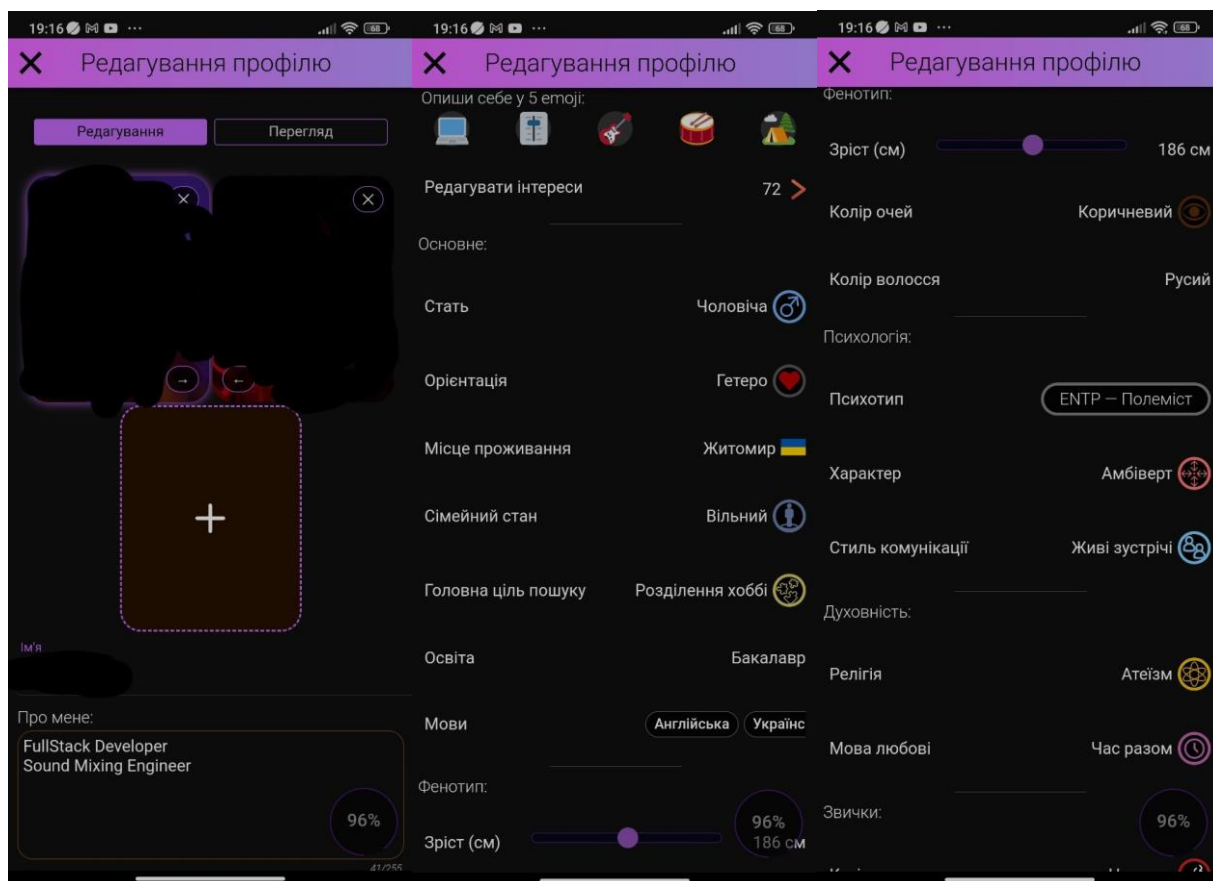


Рис. 77. Заповнення профілю на прикладі дружнього режиму

Заповнення профілю складається з трьох основних секцій: фотографії (потрібно завантажити від 1 до 7 фотографій, під час завантаження використовується компонент кадрування, написаний у розділі 2.2.4), специфічна для режиму інформація (до неї входить опис «Про мене», опис у емої та пункт «Головна ціль пошуку») та загальна інформація, що є спільною для обох профілів.

У правому-нижньому кутку знаходиться плаваючий лічильник наповненості профілю. Під час процесу реєстрації він показує лише на заповнення профілю, проте, у звичайному вигляді, туди, також, входить наповненість інтересів.

У верхній частині знаходиться перемикання між редагуванням та переглядом профілю. Режим перегляду майже аналогічний до того, що зображено у пункті «Пошук людей» цього розділу.

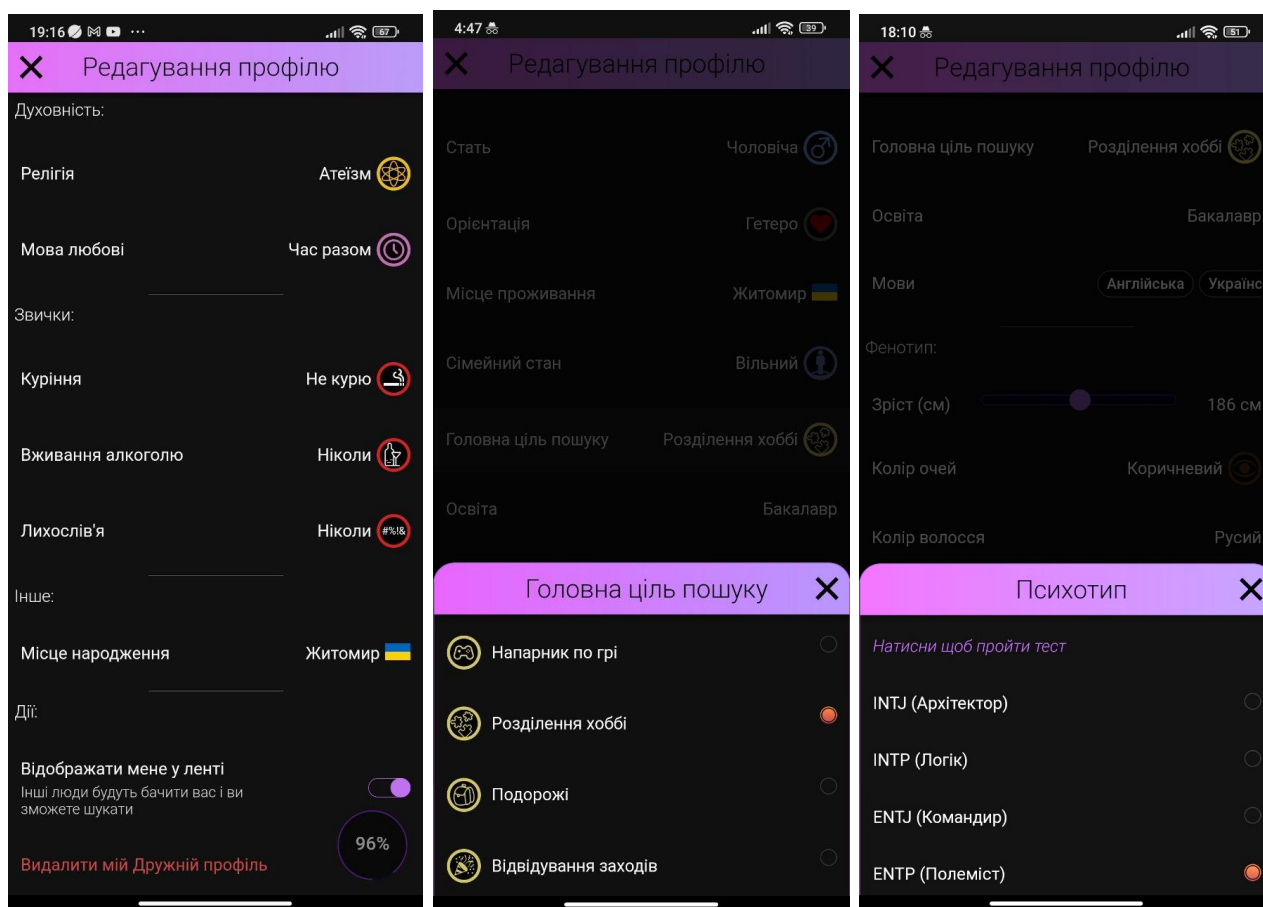


Рис. 78. Приклад заповнення профілю

Більшість полів у профілі заповнюється шляхом вибору значення із списку у окремому вікні, як це виглядає зображено на рисунках вище. Це зроблено задля того, щоб можна було також зручно фільтрувати по цим значенням у майбутньому. Зліва від опцій розташовані іконки, що їх зображають, це допомагає користувачеві познайомитися з іконками та звикнути до них. Також, при заповненні деяких полів користувачу пропонується пройти тест, як це показано на останньому зображенні Рис. 78.

Знизу розташований перемикач видимості анкети для інших людей та кнопка видалення профілю. Відновити (створити новий) видалений профіль можна у налаштуваннях Акаунту.

3.1.2 Дерево інтересів

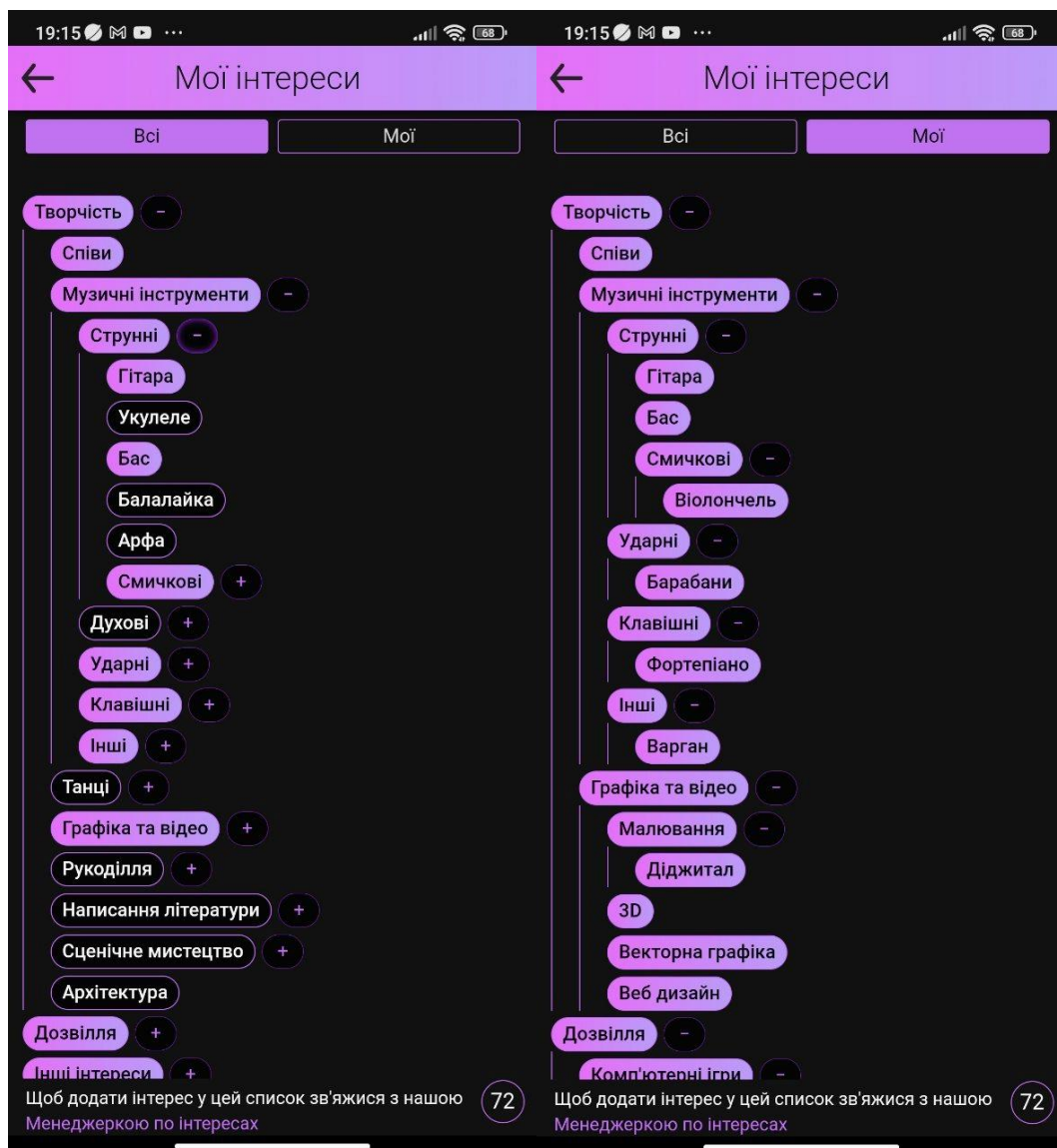


Рис. 79. Заповнення інтересів

Інтереси у додатку було вирішено відображати у вигляді дерева. Це значно спрощує процес їх вказання, адже користувачеві не потрібно скролити величезні списки із існуючих інтересів (як це зроблено у таких додатках як Tinder) або намагатися пригадати, свої захоплення та вписати їх вручну у поле пошуку (як це реалізовано у додатках типу Yubo та Boo). Також, користувач сам обирає степінь деталізації інтересів, заглиблюючись в уточнення за бажанням.

На момент написання записки, додати нові інтереси можливо лише через панель адміністрування, проте у майбутньому планується реалізувати механізм,

при якому користувачі зможуть додавати нові інтереси та вони будуть з'являтися у загальному дереві після підтвердження модераторами та переміщення у потрібне місце дерева.

Також, планується використовувати степінь вкладеності інтересу як його вага у розрахунку сумісності. Чим більш вкладені інтереси співпадають, тим вище пріоритет такого профіля. Це буде як мотивувати детально заповнювати своє дерево, так і допоможе користувачам знаходити людей, що цікавляться саме тими речами, що й вони.

3.1.3 Пошук людей

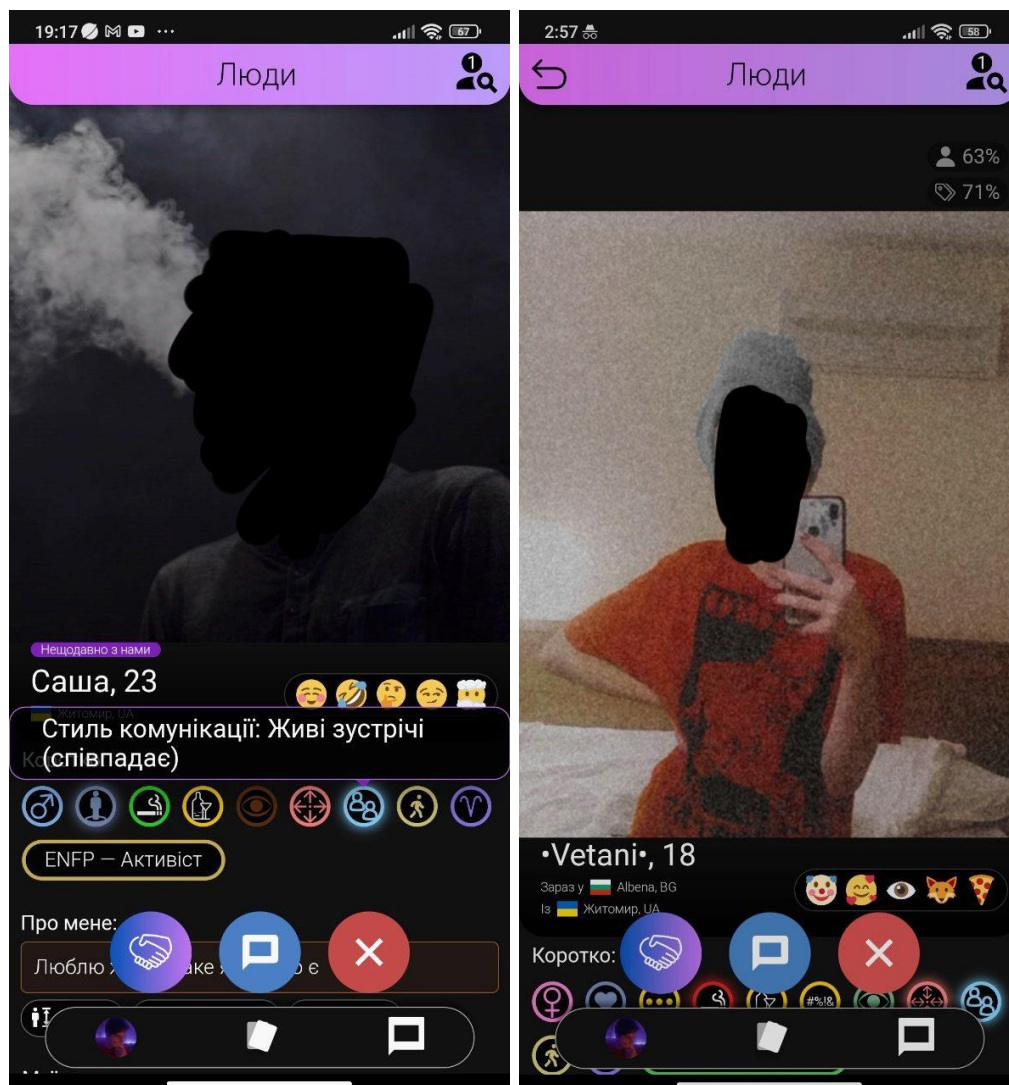


Рис. 80. Перегляд профілів інших користувачів

Перегляд анкет користувачів відбувається схожим чином з іншими додатками для знайомств. Знизу можна побачити три кнопки: «Вподобання» - відправка симпатій, «Повідомлення» - відправка симпатії з прикріпленим повідомленням та «Пропуск» - перехід до наступного профілю. У правому-верхньому кутку фотографії зображено сумісність за особистістю та за інтересами.

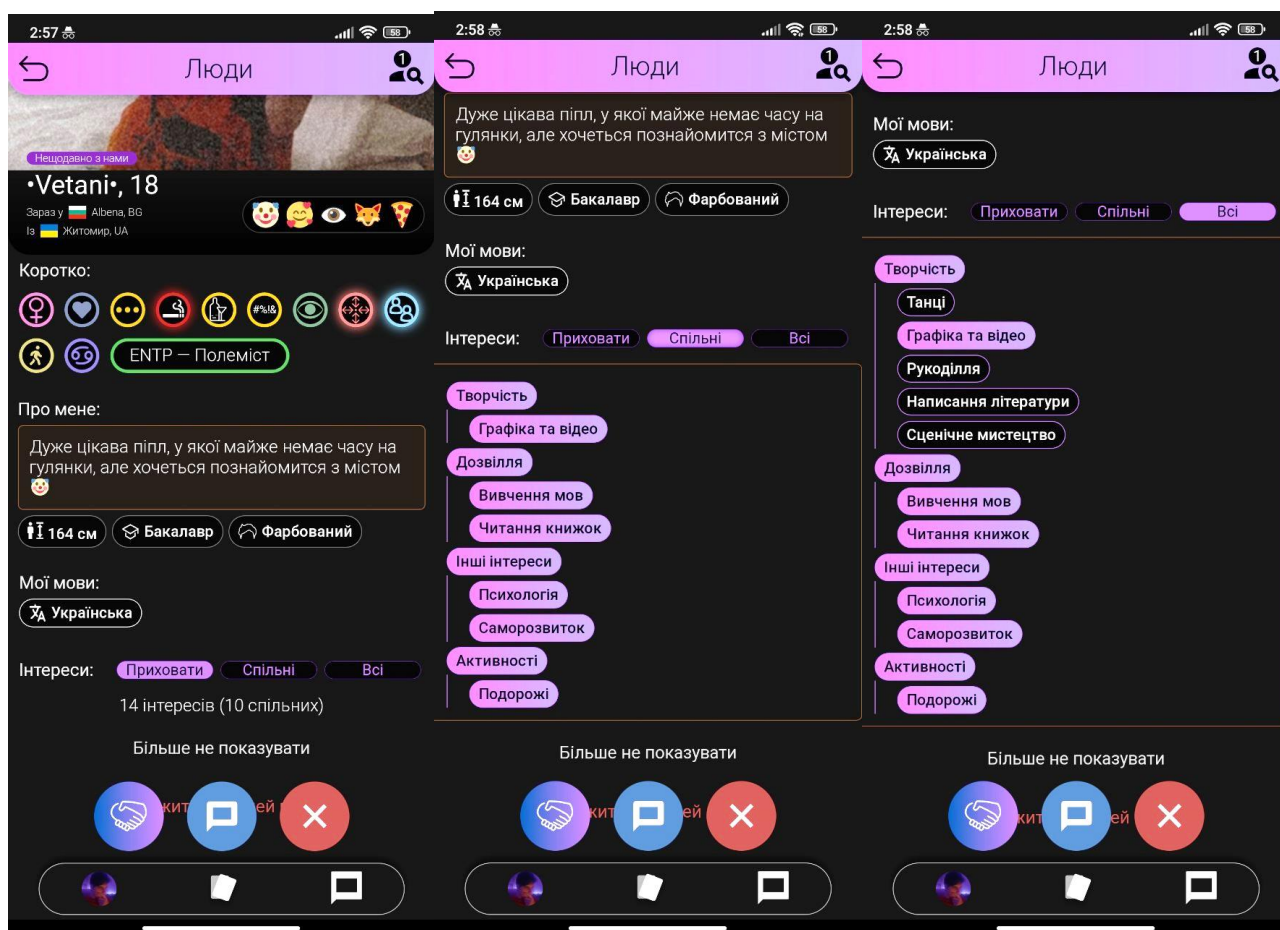


Рис. 81. Інші поля анкети

Нижче зображені цінності користувача у вигляді іконок (іконки, що співпадають із профілем користувача підсвічуються), сумісність за психотипом, текст «Про мене», зріст, освіта, колір волосся та мови у вигляді тегів.

Під ними розташовані інтереси у трьох режимах відображення та дві кнопки: «Більше не показувати», що додає цей профіль у власний чорний список користувача та «Поскаржитися» для сповіщення адміністрації про правопорушення у анкеті.

3.1.4 Налаштування фільтрів

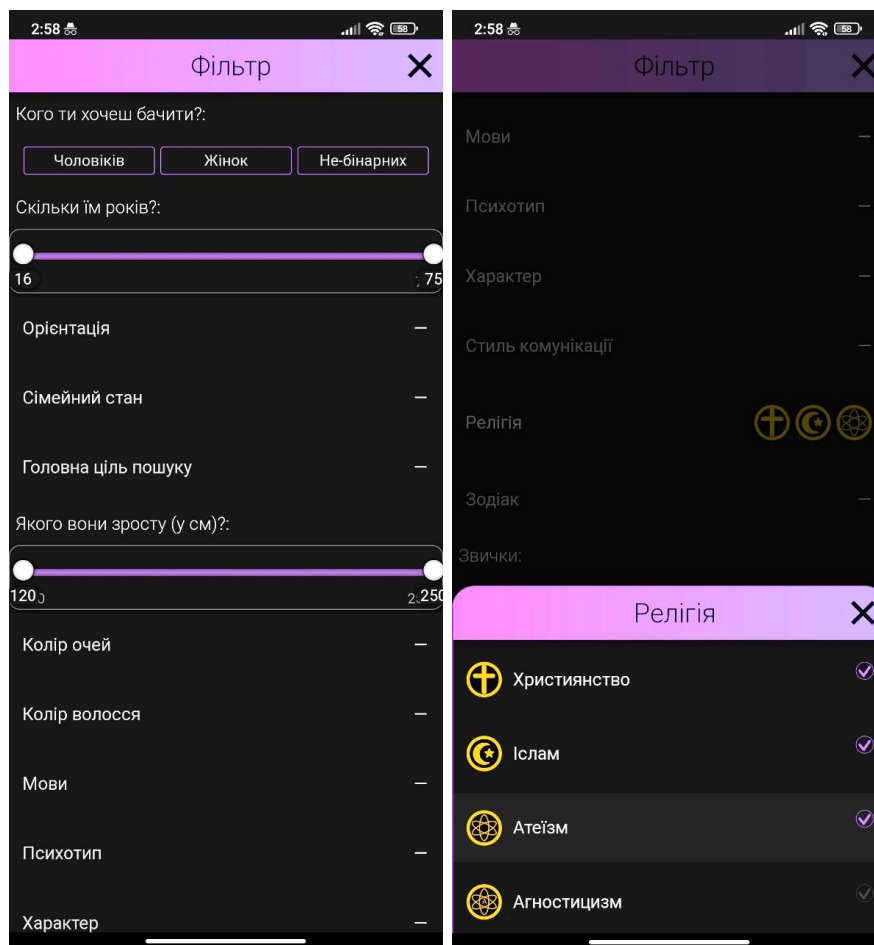


Рис. 82. Екран заповнення фільтрів

Заповнення фільтру схоже на заповнення профілю, проте, замість одиночного вибору опцій, тут можна обрати декілька варіантів.

Важливо зазначити, що відображатимуться лише ті люди, що підпадають під всі критерії фільтру користувача та лише у випадку якщо сам користувач підпадає під всі критерії їх фільтрів.

Така робота алгоритмів дає можливість дуже точного пошуку співпадінь у при умові, що у системі зареєстрована велика кількість людей, а також створює для кожної людини комфортне оточення.

Якщо людей не знайдено, буде відображено прохання змінити фільтри та зробити їх менш жорсткими.

3.1.5 Діалоги

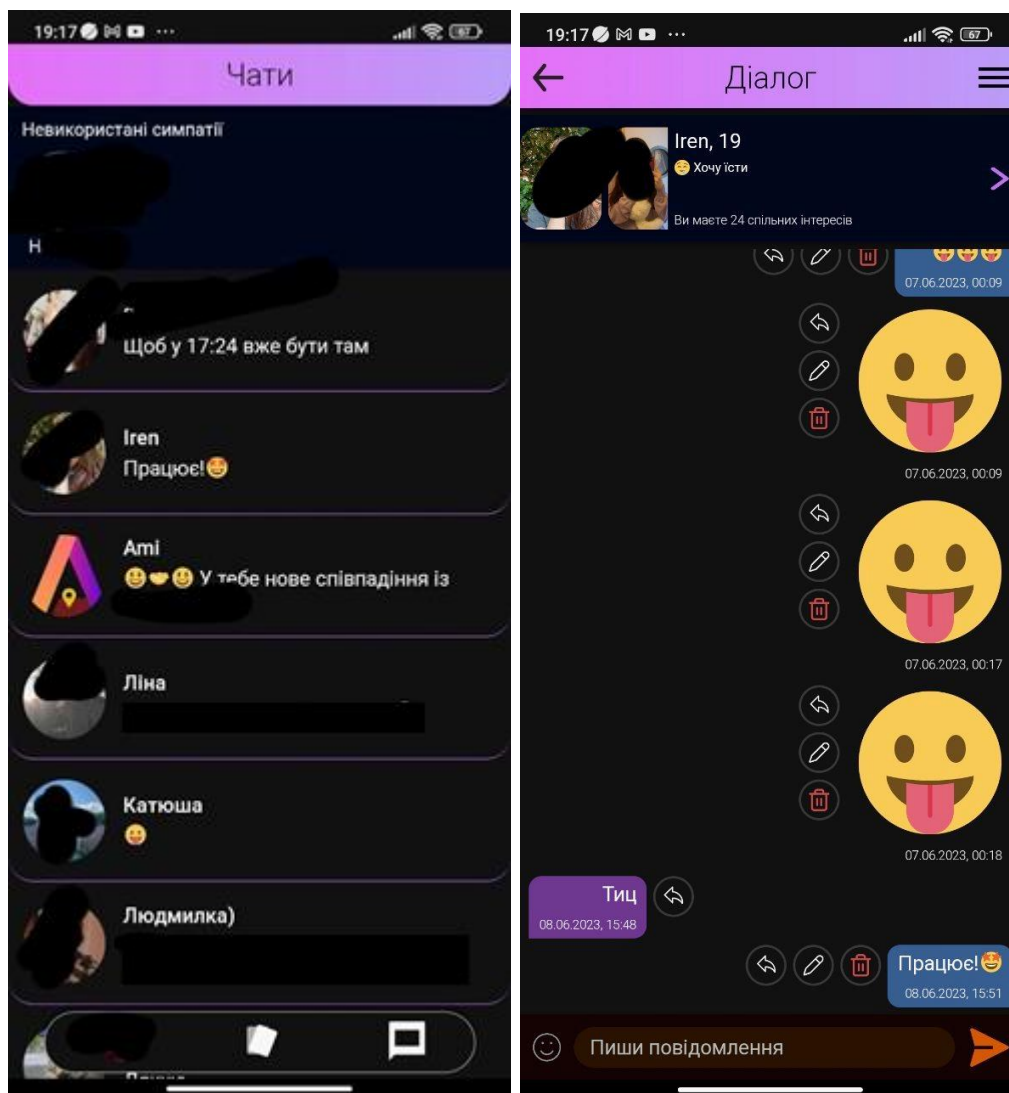


Рис. 83. Сторінка чатів та приклад відкритого діалогу

На сторінці чатів розміщені усі діалог, відсортовані за спаданням дати останнього повідомлення. Діалог з «Амі» є системним та використовується для відправки сповіщень. У верхній частині екрану чатів знаходяться невикористані симпатії, при відкриття яких буде запропоновано відправити перше повідомлення).

Сторінка діалогу схожа на звичайне вікно чату із полем написання тексту повідомлення, кнопкою відправки та самими повідомленнями. У верхній частині зображено фотографії співбесідника, його статус та кількість спільних інтересів. При натисканні на цю область відкриється сторінка перегляду профілю, а у

випадку натискання на конкретну фотографію, на відкритій сторінці автоматично відкриється обрана фотографія.

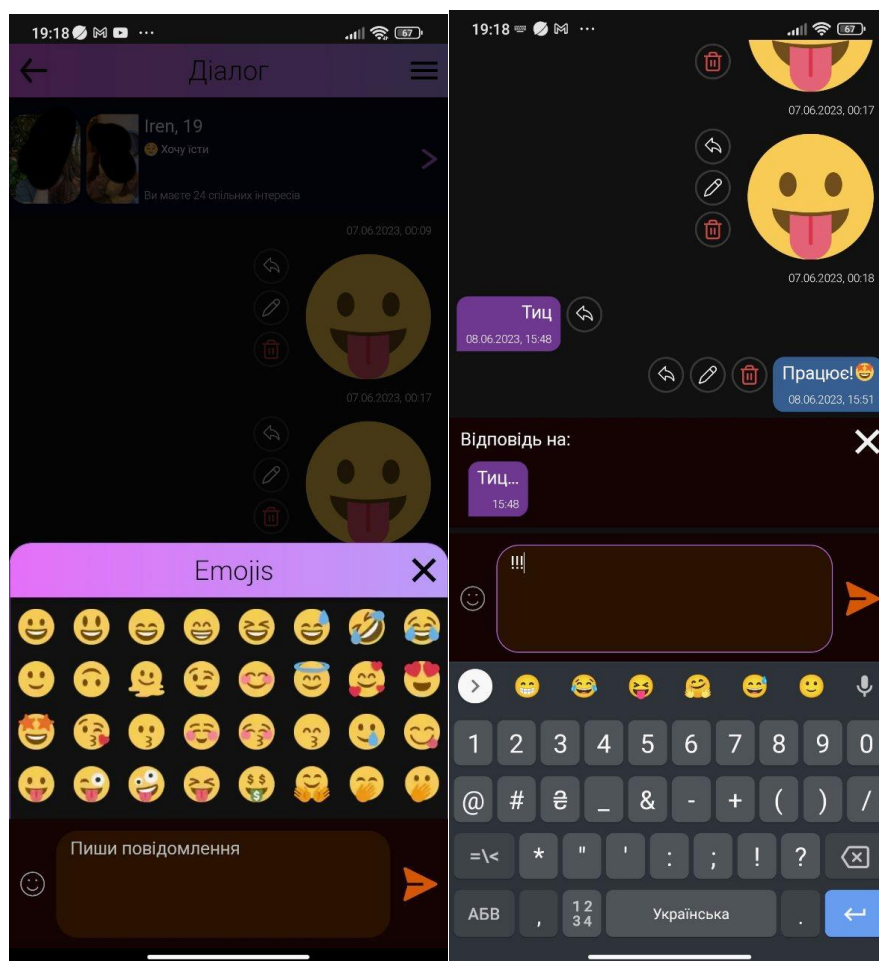


Рис. 84. Відкрите вікно додавання емої та приклад відповіді на повідомлення

На Рис. 84 зображені варіанти сторінки з відкритим вікном емої та приклад відповіді на повідомлення. Також, на цих скріншотах можна побачити стікери, детальніше про реалізацію яких було описано у розділі 2.2.5 цієї записки.

Також, біля кожного повідомлення розміщено кнопки дій: «Відповісти», «Відредагувати» та «Видалити». У майбутньому планується зробити їх більш компактними, приховавши у actions-меню.

Також, у правому-верхньому кутку знаходиться меню чату. На даний момент, там знаходиться лише дві кнопки: «Розірвати контакт», що видаляє чат

та додає користувача до чорного списку, та кнопка «Поскаржитися», що відправляє скаргу на діалог та відкриває доступ до нього адміністрації.

3.1.6 Персональний QR-код

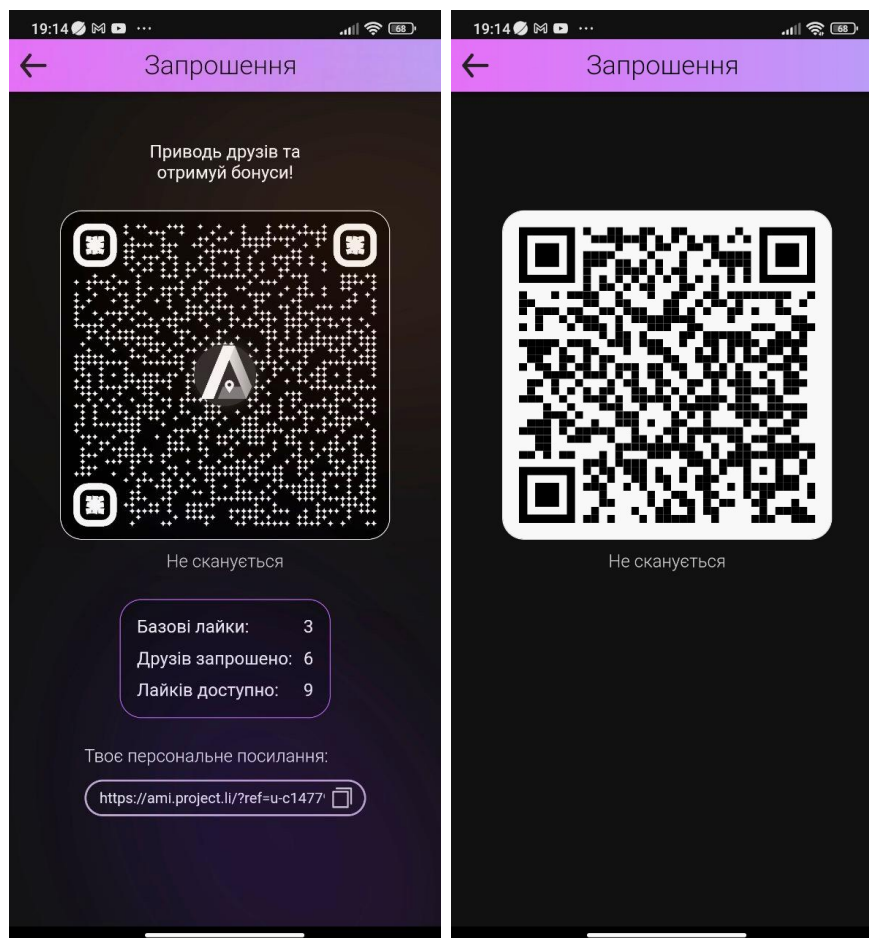


Рис. 85. Сторінка запрошення та два варіанти персонального QR коду

На сторінці запрошення зображена кількість доступних вподобань на день, персональне посилання та персональний QR-код. У випадку, якщо виникли проблеми зі скануванням QR-коду (таке трапляється на телефонах Xiaomi та у деяких програмах для читання, що підтримують лише стандартний вигляд коду з чорними квадратами на білому фоні), є можливість перемикнутися на класичну версію коду.

QR-коди генеруються сервісом QR Code Monkey [43] у SVG форматі та кешуються на стороні серверу, це дозволяє максимально заощадити трафік, а також, досягти максимальної чіткості зображення.

Код виклику генерації QR-коду та кешування результату знаходиться у файлі src/user/qr-code.service.ts.

3.1.7 Налаштування

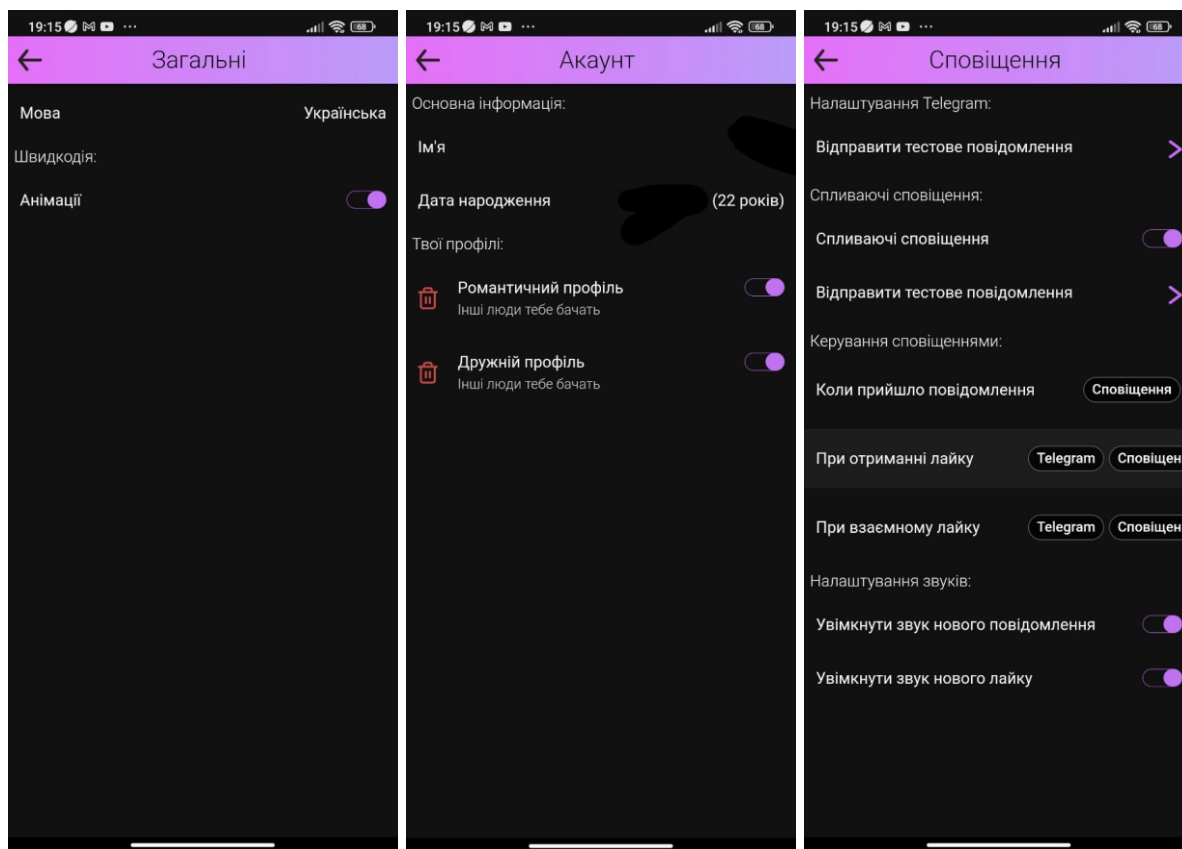


Рис. 86. Налаштування додатку (Загальні, Акаунт та Сповіщення)

На сторінці загальних налаштувань можна змінити мову та відключити анімації.

На сторінці акаунту можна переглянути ім'я при створенні акаунта та дату народження, також, можна створити, відключити або видалити профілі.

На сторінці сповіщень можна налаштувати типи сповіщень для різних дій, а також, перевірити налаштування сповіщень у Telegram шляхом відправки тестового сповіщення через бота (кнопка «Відправити тестове повідомлення» зверху). У випадку невдачі відправки повідомлення, показується інструкція про можливі проблеми та шляхи їх вирішення.

Під налаштуваннями Telegram, можна увімкнути або вимкнути підписку на Push-сповіщення на цьому пристрої, а також, відправити тестове сповіщення. Також, нижче розміщені перемикачі для звукових сигналів всередині додатку.

3.2 Панель адміністрування

Окремою складовою додатку є панель адміністрування, що дає можливість забезпечувати належний контроль над анкетами користувачів та вживати відповідні заходи у разі виявлення порушення правил. Крім того, через панель адміністрування відбувається заповнення глобального списку інтересів.

Панель адміністрування нативно інтегрується у додаток у вигляді пункту «Admin Dashboard» на сторінці профілю користувача. Цей пункт доступний лише коли у налаштуваннях дозволів користувача є доступ до адміністрування. Задля більшої наочності, колір верхньої панелі у підрозділах панелі адміністрування змінений на сірий, це попереджає помилкові необачні дії, адже постійно нагадує адміністратору, що він знаходиться у режимі підвищених привілеїв.

Всередині панелі адміністрування є декілька розділів, в залежності від набору прав конкретного адміністратора, та вікно швидкої статистики у верхній частині, що показує мінімальну необхідну на даний момент часу.

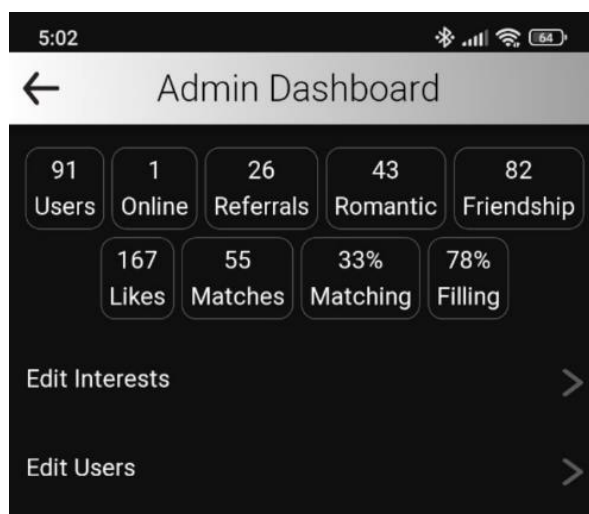


Рис. 87. Зовнішній вигляд головного меню панелі адміністрування

Також, слід зазначити, що заради заощадження об'єму файлу локалізації, панель адміністрування доступна лише англійською мовою, проте, розглядається можливість її локалізації після модульного розділення мовних файлів.

3.2.1 Редагування інтересів

На сторінці редагування інтересів є можливість переглядати дерево інтересів, додавати нові гілки інтересів, редагувати та видаляти існуючі. У випадку видалення гілки, що містить в собі під-інтереси, усі під-інтереси автоматично переміщуються до батьківської гілки.

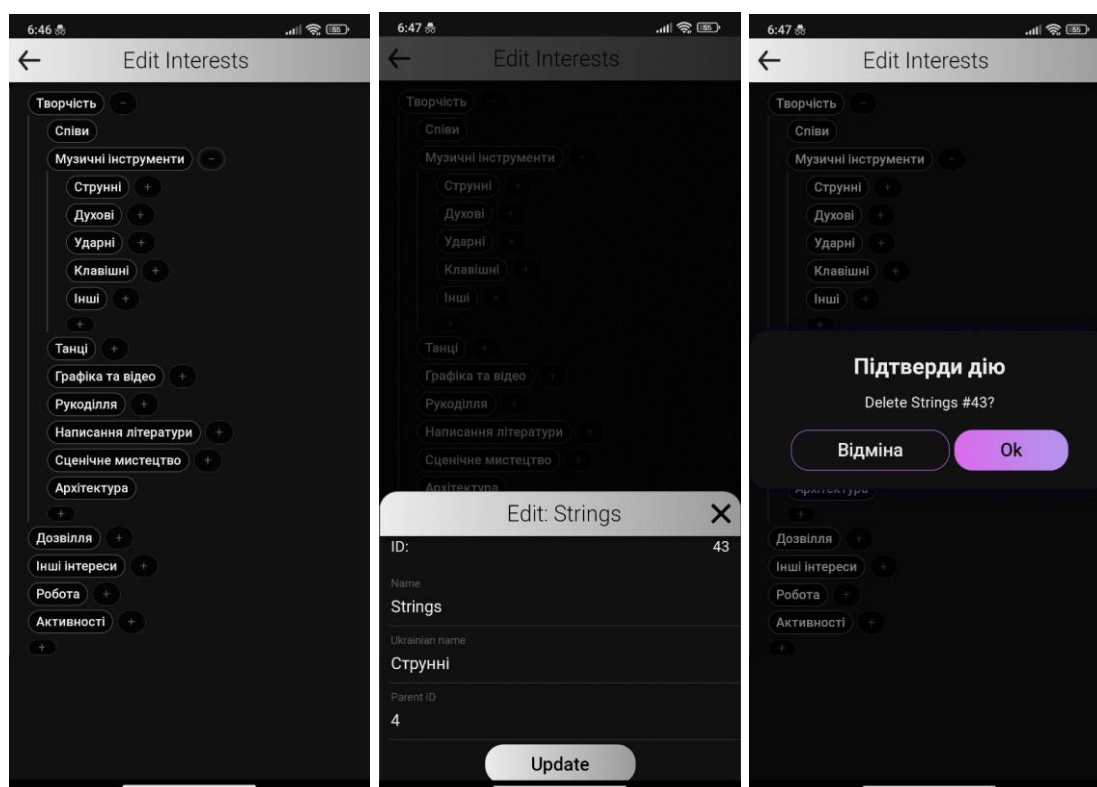


Рис. 88. Сторінка дерева інтересів, приклад редагування та видалення інтересу

Кожен інтерес має додаткову інформацію, а саме, ID, ID батьківського інтересу (або -1 для інтересів першого порядку), назву англійською та українською мовами. У випадку відсутності перекладу на українську мову (пуста стрічка), переклад береться з англійської. Саме дерево інтересів відображається на мові користувача.

Після редагування інтересів відбувається генерація нового дерева інтересів та оновлення кешу на сервері. Час життя локального кешу дерева інтересів складає одну добу.

3.2.2 Перегляд та блокування користувачів

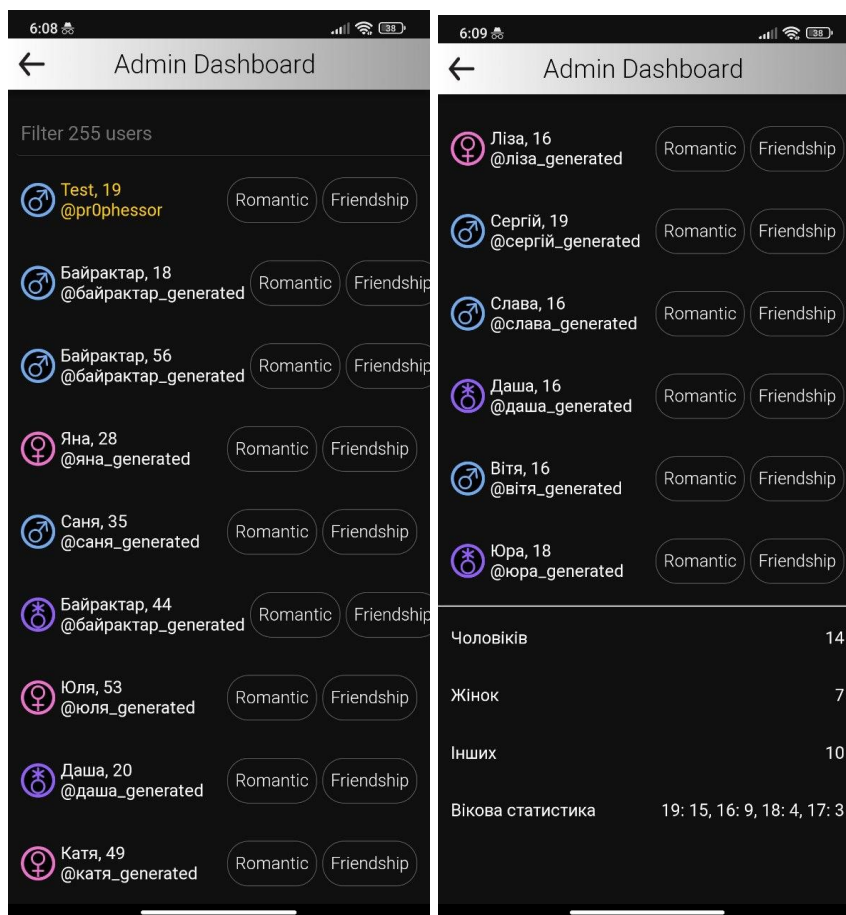


Рис. 89. Перегляд списку користувачів

На сторінці перегляду інформації про користувачів можна побачити їх ім'я, вік, username у Telegram (якщо є) та наявність профілів. На Рис. 89 зображені згенеровані користувачі на тестовій базі даних по причині конфіденційності. Жовтим кольором позначені адміністратори, коричневим – заблоковані користувачі. У верхній частині розташоване поле фільтрації. Фільтрувати можна по імені, віку, Telegram username, та ID користувача або будь-якого з його профілів. Завдяки тому, що ID профілів мають префікси режиму, можна відобразити лише користувачів, у яких є відповідні профілі, увівши «г-» або «f-». У нижній частині зображена кількісна статистика із

розбиттям за статтю та віком. При використанні фільтру, статистика будується за користувачами, що пройшли через фільтр.

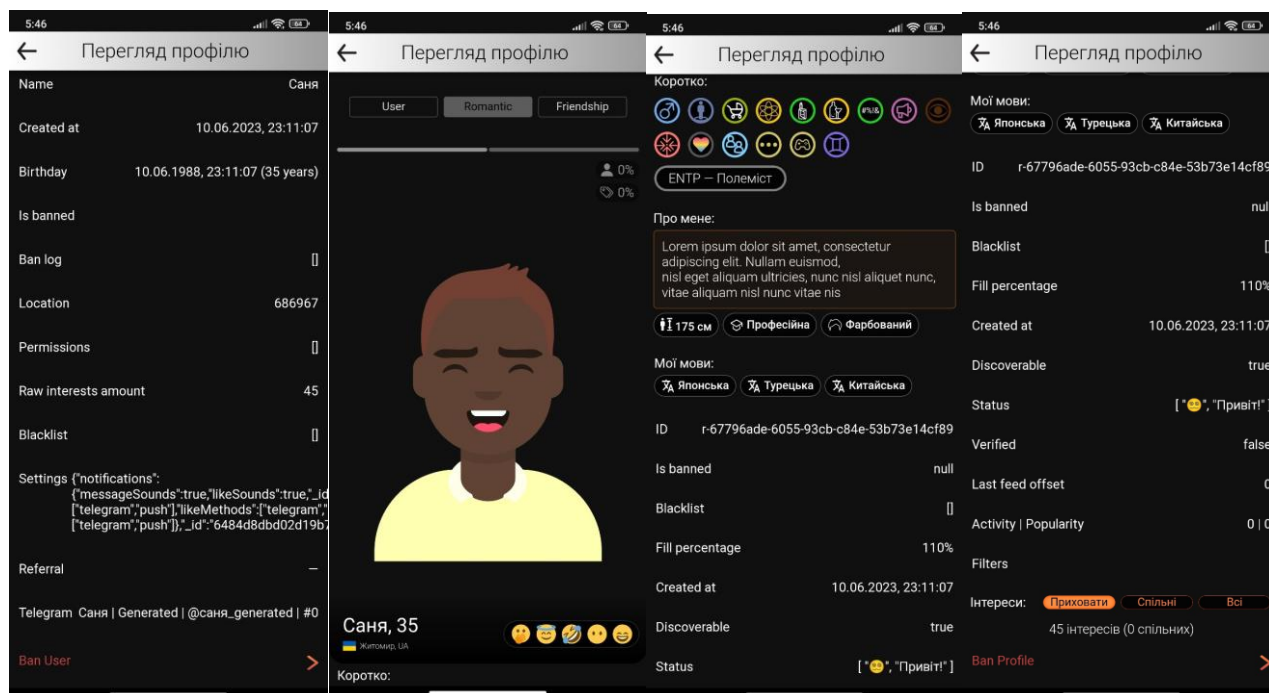


Рис. 90. Перегляд інформації про користувача

Перегляд профілів користувача (Рис. 90) дуже схожий на те, як це виглядає у звичайному пошуку людей, проте відображаються усі іконки незалежно від типу профілю, а також відображені системні поля у сірому вигляді. У самому низу є можливість заблокувати користувача або якийсь з його профілів, або ж відредагувати інформацію.

При редагуванні інформації, відкривається звичайна сторінка редагування профіля, проте, з даними обраного користувача. Перед виходом із сторінки з'являється вікно підтвердження змін.

При блокуванні користувача або його профілів можна вказати причину блокування та дату розблокування (або заблокувати навечно, якщо увімкнути відповідний перемикач).

У заблокованого користувача буде відображатися сторінка як на другому Рис. 91. Якщо заблокований лише один із профілів, користувач, все ще, може перемикнути на інший профіль, натиснувши на верхню панель.

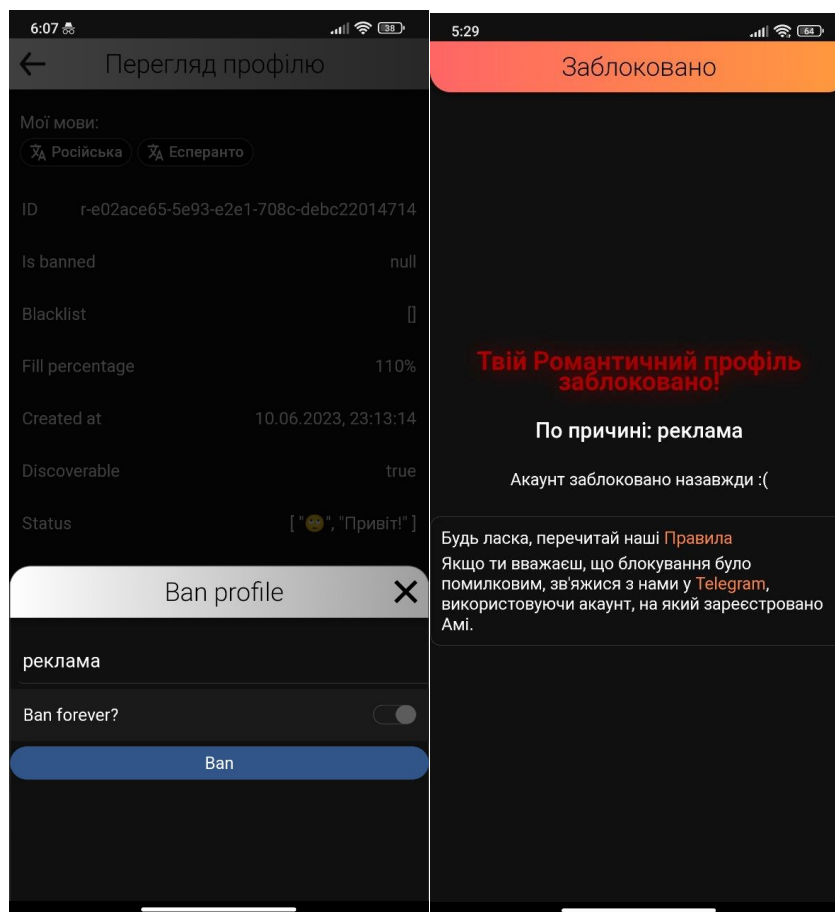


Рис. 91. Блокування профілю користувача та екран блокування

Також, слід зазначити, що для заблокованих профілів усі запити на сервер скасовуються з помилкою 403 та текстом «You are banned». Якщо клієнт отримує цю помилку, він відправляє запит на оновлення стану блокування та відображає відповідну сторінку.

3.3 Аналітика поведінки користувачів за допомогою Amplitude

Розуміння поведінки користувачів є ключовим елементом успіху у розвитку продукту. Аналіз поведінки дає можливість побачити слабкі місця додатку для їх наступного виправлення.

Щоб отримати цінні інсайти та покращити користувацький досвід, ми будемо використовувати Amplitude - потужний інструмент для аналізу внутрішньої поведінки користувачів. Amplitude надає можливість збирати дані про взаємодію користувачів із додатком (Рис. 92), аналізувати їхні дії та здійснювати глибоке дослідження способу використання. Це дозволить виявити

сильні та слабкі сторони продукту, вносити необхідні зміни та створювати персоналізовані рішення для користувачів.

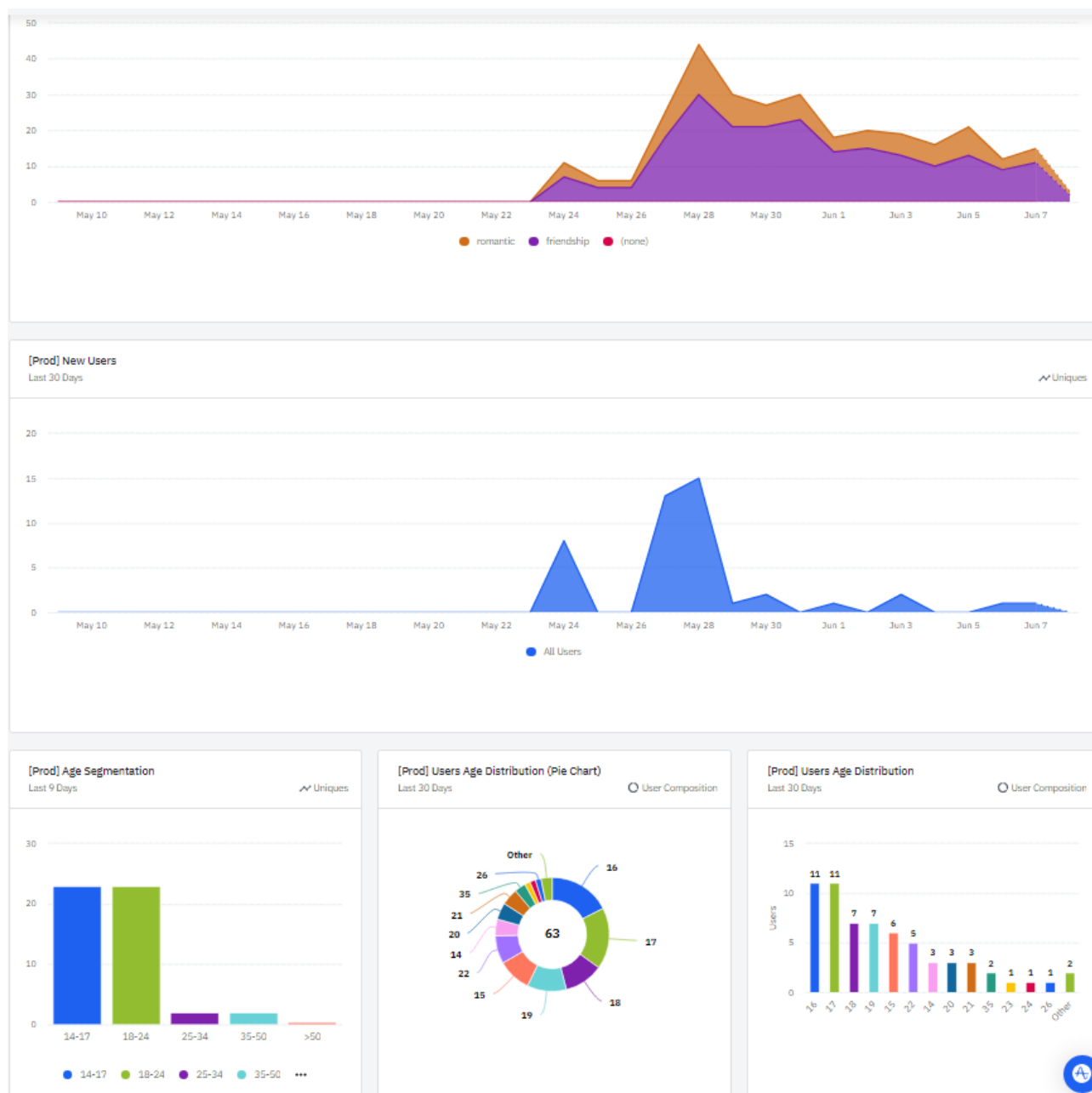


Рис. 92. Вікно загальної статистики

Інформація до Amplitude та Google Analytics 4 надходить через сервіс-маршрутизатор подій Segment, що дозволяє налаштувати відправку схожих даних у різні сервіси, використовуючи один скрипт. Це дуже зручно, адже у клієнтській частині додатку достатньо відправляти події, які виконує користувач у Segment, а сервіс вже розподіляє ці дані за сервісами аналітики (Рис. 93).



Рис. 93. Маршрутизація даних аналітики за допомогою сервісу Segment.io

3.4 Відстеження помилок за допомогою Sentry

Відстеження помилок на клієнтській частині є дуже важливою складовою контролю якості, адже лише невелика кількість користувачів звертається до служби підтримки коли бачить помилку, а деякі помилки можуть бути взагалі непомітними. Для відстеження клієнтських помилок ми будемо використовувати сервіс Sentry. Він інтегрується [44] із Vue.js, перехоплює усі помилки, групує їх, додає метадані та показує цю інформацію у зручному для читання вигляді (Рис. 94).

Issues

All Unresolved20

For Review16

Ignored

My Projects

All Envs

30D

All Categories

Custom Search

is:unresolved

☐	Resolve	Ignore	Mark Reviewed	Merge	...	Last Seen	GRAPH:	24h30d	EVENTS	USERS	ASSIGNEE
☐	AxiosError	Fe(ami/assets/index)	Request failed with status code 400				1	1	1		
	New Issue	JAVASCRIPT-VUE-M	12hr ago	12hr old							
☐	NotAllowedError	play([native code])	The request is not allowed by the user agent or the platform in the current context, possibly b...				1	1	1		
	New Issue	JAVASCRIPT-VUE-K	Unhandled	12hr ago	12hr old						
☐	TypeError	d(ami/assets/index)	Cannot read properties of undefined (reading 'birthLocation')					7	4		
	New Issue	JAVASCRIPT-VUE-A	Unhandled	24hr ago	5d old						
☐	TypeError	o(ami/assets/index)	Cannot read properties of undefined (reading 'birthday')					7	5		
	New Issue	JAVASCRIPT-VUE-B	Unhandled	24hr ago	5d old						
☐	TypeError	?(ami/assets/index)	Cannot read properties of undefined (reading 'discoverable')					3	2		
	New Issue	JAVASCRIPT-VUE-J	2d ago	3d old							
☐	AxiosError	G_(ami/assets/index)	Request failed with status code 400					3	2		
	New Issue	JAVASCRIPT-VUE-D	2d ago	4d old							

Рис. 94. Перегляд необроблених помилок

3.6 Резервне копіювання бази даних

Для забезпечення надійності, на сервері було налаштовано щоденне нічне резервування бази даних.

```
backup-ami-production.sh • x
6 #!/bin/bash
5 /home/.../mongodump \
4 --uri 'mongodb://...?retryWrites=true&connectTimeoutM
3 --archive
2 tee "/home/...backups/mongo-ami-production/${date +%Y-%m-%d-%H-%M-%S'}.dump"
1 > /dev/null
```

Рис. 95. Створення резервної копії бази даних

Резервування виконується за допомогою утиліти `mongodump`, що робить зліпок бази даних та зберігає його у стисненому `.bson` файлі. Далі цей переміщується до архіву резервних копій.

```
@reboot /home/.../run_ami_prod.sh
@reboot /home/.../run_staging.sh
3 20 * * * /home/.../backup-ami-production.sh
```

Рис. 96. Налаштування `crontab -e`

За виклик утиліти резервування відповідає `cron`-задача, налаштована на сервері на час 4:20 UTC+3 (3:20 за CEST), адже, за результатами попереднього дослідження, навантаження системи у цей час мінімальне.

Висновок до третього розділу

У цьому розділі було зроблено огляд готового додатку, панелі адміністрування та описано інструменти для його подальшої підтримки та адміністрування. Також, було описано налаштування щоденного нічного резервування бази даних.

У наступному розділі пояснювальної записки буде підведено підсумки виконання кваліфікаційної роботи, а також, підведено загальну статистику.

ВИСНОВКИ

У рамках даної кваліфікаційної роботи було проаналізовано актуальність онлайн знайомств, розглянуто існуючі додатки та розроблено концепцію і основні вимоги додатку, що буде створюватися, розроблено дизайн додатку у програмі Figma, створено прототип на основі цього дизайну та перевірено його на цільовій аудиторії, зібрано відгуки та внесено зміни перед початком розробки, спроектовано базу даних на основі СКБД MongoDB, реалізовано серверну та клієнтську частини з використанням найсучасніших бібліотек таких як Nest.JS та Vue.JS, та технологій, таких як Progressive Web Application, Push Notifications, OAuth & OIDC, HuggingFace, Service Worker та інших.

Використано API моделі нейронної мережі для виявлення непристойних фотографій, реалізовано реферальну систему з генерацією персональних QR-кодів, дерево інтересів, алгоритми підбору людей на основі спільних інтересів та індексу сумісності, що включає в розрахунок такі параметри як деякі цінності та психотип користувача.

Також, було розроблено панель адміністрування для зручного супроводу додатку під час експлуатації.

При загальному підрахунку, у трьох репозиторіях виявилось близько 30 тисяч стрічок лістингу (у це число включено файли конфігурації, проте, не включено сторонні бібліотеки).

Якщо оцінити отриманий додаток за критеріями, описаними у розділі 1.1.3 та порівняти його з існуючими додатками, отримаємо наступну таблицю (таблиця 2). Варто зазначити, що через брак вільного місця на листі, бот Дайвінчик було видалено зі таблиці для можливості додати новостворений додаток.

Порівняльна таблиця існуючих додатків з новоствореним

Сервіс	Tinder	Badoo	Bumble	Boo	Ami
Знайомства	Романтичні	Романтичні	Романтичні + Дружні	Романтичні + Дружні	Романтичні + Дружні
Формат	Додаток + PWA	Додаток + Сайт	Додаток	Додаток	PWA
Інтереси [0-5]	1	1	1	3	5
Психотип [0-5]	1	0	0	4	5
Обмеження	2	3	1	4	5
Фільтр	1	3	3 (1)	4	5
Сумісність	0	1	0	3	5
Популярність в Україні	5	5	3	2	0

Отриманий додаток успішно введено в експлуатацію та вже за перший тиждень вдалося набрати більше 90 активних користувачів, що, в загальному, створили більше 110 профілів та відправили більше 150 вподобань. З них 26 користувачів зареєструвалися у додатку по реферальній програмі. Активною цільовою аудиторією виявилися молоді люди від 16 до 25 років (Рис. 97), проте є і більш старші користувачі, що виражали задоволеність додатком.

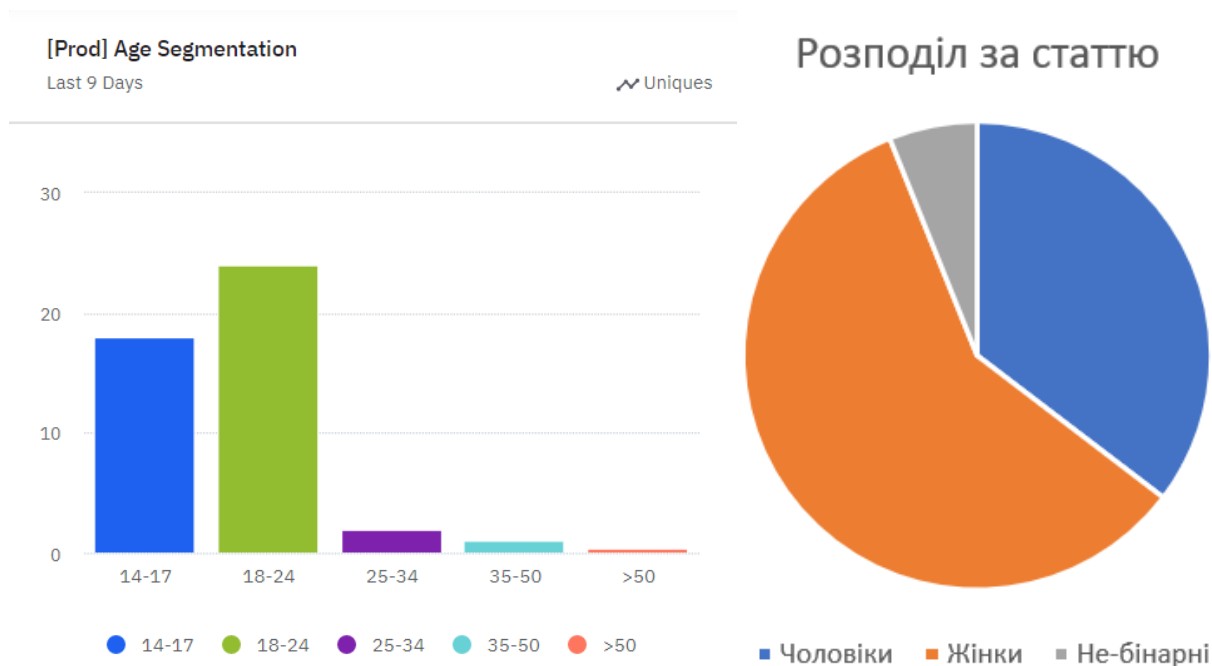


Рис. 97. Аудиторія додатку за віком та статтю, станом на 10.06.2023

За час роботи над проектом було створено більше 500 карток у системі керування проектами Jira. Починаючи з лютого 2023 року, сумарно зафіксовано більше 800 комітів (Рис. 98) у системі контролю версій Git.



Рис. 98. Діаграма комітів, починаючи з лютого 2023 (сервер-клієнт-типи)

За весь час було використано 404 хвилини CI/CD (Рис. 99) для безперервного збирання та розгортання системи (включаючи staging та production оточення).

Usage by month

Compute usage Shared runner duration ?

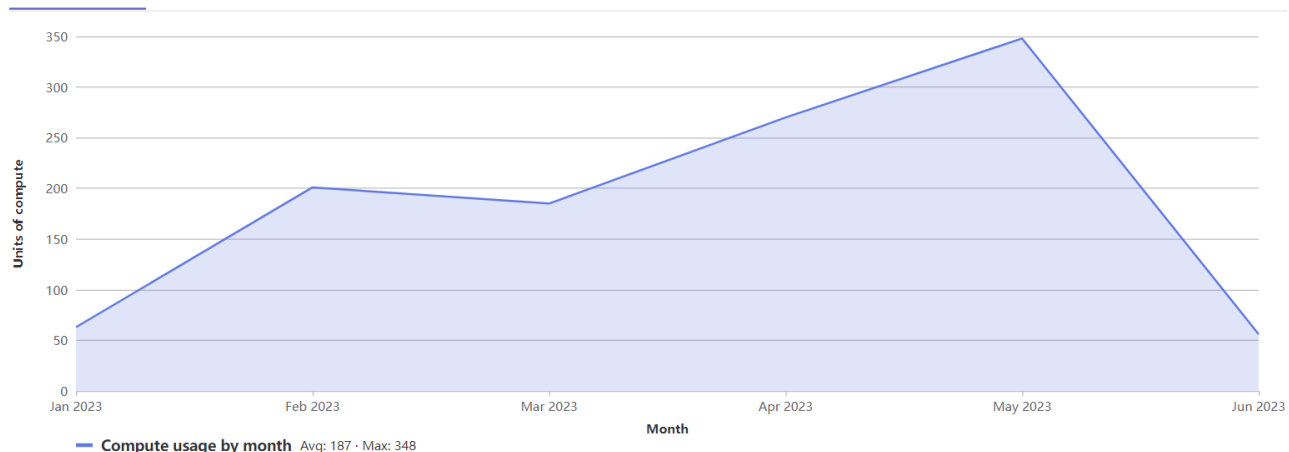


Рис. 99. Використання CI/CD minutes у GitLab

База даних у production-середовищі містить 8 колекцій, 23 індекси, сумарний об'єм бази даних складає 1.5 Мб, кожен день створюється запланована стиснена резервна копія, об'ємом 500 Кб.

Мета роботи виконана у повному обсязі та у кінцевому результаті отримано працюючий додаток для знайомств та пошуку друзів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. COMPATIBILITY AND MYERS & BRIGGS' PERSONALITY TYPES [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://www.truity.com/myers-briggs/compatibility-myers-briggs-personality-type>.
2. Progressive Web Apps - web.dev [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://web.dev/progressive-web-apps/>.
3. Документація Vue 3 [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://v3.vuejs.org/guide/introduction.html>.
4. Документація VueRouter [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://router.vuejs.org/guide>.
5. Документація Pinia [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://pinia.vuejs.org/introduction.html>.
6. Vue 3 Composition API vs Options API - Jennifer Bland [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://www.jenniferbland.com/vue-3-composition-api-vs-options-api>.
7. Документація Vite [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://vitejs.dev/guide>.
8. sass-lang.com Documentation to SASS pre-processor [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://sass-lang.com/documentation>.
9. Documentation | NestJS - A progressive Node.js framework [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://docs.nestjs.com/>.
10. GraphQL Vs. REST APIs - Hygraph [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://hygraph.com/blog/graphql-vs-rest-apis>.

11. What is PostgreSQL [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/>.
12. Getting Started — MongoDB Manual [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://dev.to/jctaveras/sequelize-typescript-what-you-need-to-know-41mj>.
13. Jsonpath – a query language for JSON in Postgres [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <http://www.sai.msu.su/~megera/postgres/talks/jsonpath-pgday.it-2019.pdf>.
14. TypeORM is poorly maintained [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://news.ycombinator.com/item?id=26888369>.
15. Sequelize + TypeScript - What you need to know [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://dev.to/jctaveras/sequelize-typescript-what-you-need-to-know-41mj>.
16. Quick Start Guide | typegoose [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://typegoose.github.io/typegoose/docs/guides/quick-start-guide>.
17. @nestjs/mongoose - npm [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://www.npmjs.com/package/@nestjs/mongoose>.
18. Kindagoose docs [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://grapeoffjs.github.io/kindagoose/#/>.
19. OpenAPI (Swagger) | NestJS - A progressive Node.js framework [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://docs.nestjs.com/openapi/introduction>.

20. Get Started | Socker.IO [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://socket.io/get-started/chat>.
21. GitHub Copilot · Your AI pair programmer [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://github.com/features/copilot>.
22. CSS Variables - The var() Function [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: https://www.w3schools.com/css/css3_variables.asp.
23. Vue CLI [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://cli.vuejs.org/>.
24. vuejs-clipper [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://timtnleeproject.github.io/vuejs-clipper/#/>.
25. Twemoji [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://twemoji.twitter.com/>.
26. Gender Emoji ZWJ Sequences [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://www.unicode.org/L2/L2016/16181-gender-zwj-sequences.pdf>.
27. Intl - JavaScript | MDN [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Intl.
28. How To Use Indexes in MongoDB - DigitalOcean [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: <https://www.digitalocean.com/community/tutorials/how-to-use-indexes-in-mongodb>.
29. Lazy loading [Электронный ресурс] // 1. – 2023 – Режим доступа до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading.

30. OpenID Connect Protocol [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://auth0.com/docs/authenticate/protocols/openid-connect-protocol>.
31. OAuth 2.0 – OAuth [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://oauth.net/2/>.
32. Документація з Telegram Login Widget [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://core.telegram.org/widgets/login>.
33. ImageMagick – Command-line Options [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://imagemagick.org/script/command-line-options.php>.
34. Axios API | Axios Docs [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: https://axios-http.com/uk/docs/api_intro.
35. Git - Submodules [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://git-scm.com/book/en/v2/Git-Tools-Submodules>.
36. RFC 6455 - The WebSocket Protocol [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://datatracker.ietf.org/doc/html/rfc6455#section-1.3>.
37. Gateways – Nest.js [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://docs.nestjs.com/websockets/gateways#overview>.
38. mod_expires - Apache HTTP Server Version 2.5 [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: https://httpd.apache.org/docs/trunk/mod/mod_expires.html.
39. Web Push and Notifications [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://web.dev/notifications>.
40. Документація Vite PWA [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://vite-pwa-org.netlify.app>.

41. Strategies for service worker caching [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://developer.chrome.com/docs/workbox/caching-strategies-overview/>.
42. GeoNames Web Service Documentation [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <http://www.geonames.org/export/web-services.html>.
43. QR Code API with Logo and Design - QRCode Monkey [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://www.qrcode-monkey.com/qrcode-api-with-logo/>.
44. Vue | Sentry Documentation [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://docs.sentry.io/platforms/javascript/guides/vue/>.
45. Документація пакету vue-svg-loader [Електронний ресурс] // 1. – 2023 – Режим доступу до ресурсу: <https://www.npmjs.com/package/vite-svg-loader>.

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	Арк.
						126

Діаграма варіантів використання

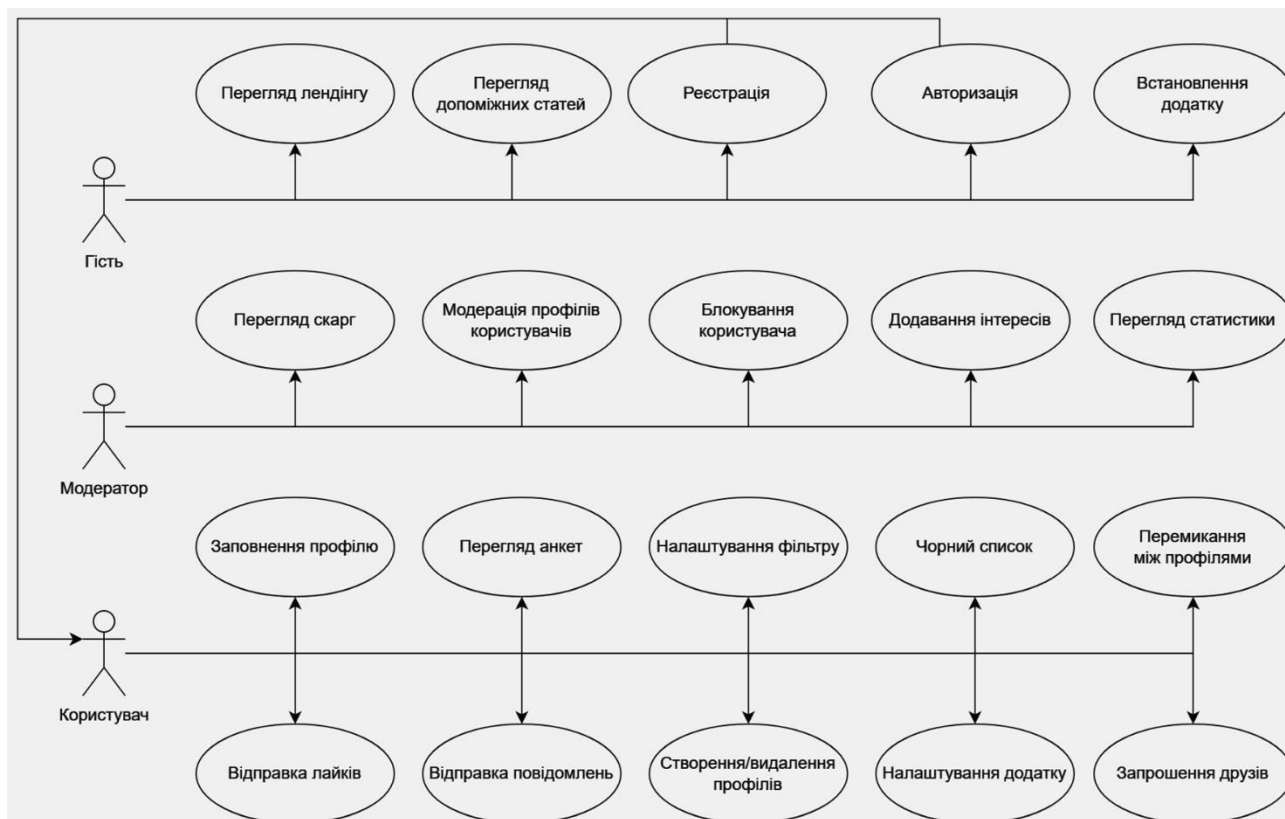


Рис. А.1 Діаграма варіантів використання
