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:

«Вебплатформа управління рекламними кампаніями в Instagram»

Виконав студент групи ІПЗ-22-4
КОТВІЦЬКИЙ Ілля Віталійович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2026

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

«ЗАТВЕРДЖУЮ»
Зав. каф. інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КОТВІЦЬКИЙ Ілля Віталійович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Вебплатформа управління рекламними кампаніями в Instagram»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: **розробка веборієнтованої системи управління рекламними кампаніями в соціальній мережі Instagram з використанням подієво-орієнтованої архітектури на платформі .NET з інтеграцією Instagram Graph API.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р. С.	04.03.2026р.	20.03.2026р.
2	Савіцький Р. С.	10.04.2026р.	17.04.2026р.
3	Савіцький Р. С.	03.05.2026р.	15.05.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного продукту у вигляді вебплатформи для управління рекламними кампаніями в Instagram та пояснювальної записки. Пояснювальна записка до дипломної роботи містить 99 сторінок, 58 ілюстрацій, 16 таблиць та 3 додатки. Метою роботи є розробка вебплатформи для централізованого управління рекламними кампаніями в Instagram.

Метою роботи є створення системи для управління рекламними кампаніями в Instagram.

У першому розділі охарактеризовано предметну область інфлюенсер-маркетингу, виконано огляд та порівняльний аналіз існуючих програмних рішень, сформульовано задачу дослідження та обґрунтовано вибір технологій реалізації.

У другому розділі визначено вимоги до системи, обрано архітектурний підхід та інструментальні засоби розробки. Спроєктовано структуру вебплатформи, модель даних і механізми взаємодії між компонентами.

У третьому розділі описано реалізацію серверної та клієнтської частин платформи, конфігурацію середовища розгортання та заходи із забезпечення безпеки. Наведено приклади інтерфейсу та результати тестування системи.

В результаті виконання роботи розроблено вебплатформу, що забезпечує повний цикл управління рекламними кампаніями в Instagram від пошуку інфлюенсерів та узгодження умов співпраці до контролю виконання й аналізу результатів.

КЛЮЧОВІ СЛОВА: ІНФЛЮЕНСЕР-МАРКЕТИНГ, INSTAGRAM, МОДУЛЬНИЙ МОНОЛІТ, DOMAIN-DRIVEN DESIGN, ASP.NET CORE, POSTGRESQL, MONGODB, RABBITMQ, ELASTICSEARCH, KEYCLOAK.

ABSTRACT

The bachelor's qualification thesis consists of a software product in the form of a web platform for managing advertising campaigns on Instagram and an explanatory note. The explanatory note contains 99 pages, 58 illustrations, 16 tables, and 3 appendices. The purpose of the work is to develop a web platform for the centralised management of advertising campaigns on Instagram.

The first chapter characterises the subject area of influencer marketing, provides a review and comparative analysis of existing software solutions, formulates the research objectives, and justifies the choice of implementation technologies.

The second chapter defines the system requirements, selects the architectural approach and development tools, and presents the designed platform structure, data model, and component interaction mechanisms.

The third chapter describes the implementation of the server-side and client-side parts of the platform, the deployment environment configuration, and security measures. Examples of the user interface and system testing results are provided.

As a result of the work, a web platform has been developed that covers the complete cycle of managing advertising campaigns on Instagram – from searching for influencers and negotiating collaboration terms to monitoring execution and analysing outcomes.

KEYWORDS: INFLUENCER MARKETING, INSTAGRAM, MODULAR MONOLITH, DOMAIN-DRIVEN DESIGN, ASP.NET CORE, POSTGRESQL, MONGODB, RABBITMQ, ELASTICSEARCH, KEYCLOAK.

ЗМІСТ

РЕФЕРАТ 3
ABSTRACT 4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ 9
1.1 Постановка задачі 9
1.2 Аналіз аналогів програмного продукту 11
1.3 Вибір архітектури системи 15
1.4 Обґрунтування вибору інструментальних засобів 18
Висновки до першого розділу 21
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ УПРАВЛІННЯ РЕКЛАМНИМИ КАМПАНІЯМИ В INSTAGRAM 22
2.1 Визначення вимог до системи 22
2.2 Розробка бази даних системи 25
2.3 Проєктування та реалізація алгоритмів роботи системи 38
2.4 Реалізація функціоналу системи управління рекламниим кампаніями 46
Висновки до другого розділу 51
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ УПРАВЛІННЯ РЕКЛАМНИМИ КАМПАНІЯМИ В INSTAGRAM 53
3.1 Порядок встановлення та налаштування параметрів платформи 53
3.2 Структура інтерфейсу системи управління рекламними кампаніями в Instagram 57
3.3 Тестування платформи 84
Висновки до третього розділу 87
ВИСНОВКИ 88
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 89
ДОДАТКИ 93
Додаток А 94

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CDC (Change Data Capture) – захоплення змін даних.

CQRS (Command Query Responsibility Segregation) – розподіл відповідальності команд і запитів.

DDD (Domain-Driven Design) – предметно-орієнтоване проєктування.

OAuth 2.0 (Open Authorization 2.0) – протокол делегованого доступу.

OIDC (OpenID Connect) – протокол автентифікації.

ORM (Object-Relational Mapping) – об'єктно-реляційне відображення.

SDK (Software Development Kit) – набір засобів розробки.

SPA (Single Page Application) – односторінковий вебзастосунок.

ВСТУП

Актуальність теми полягає в розробці централізованої вебплатформи для управління рекламними кампаніями в Instagram, яка забезпечує повний цикл співпраці між блогерами та рекламодавцями на основі достовірних показників аудиторії. Інтеграція з соціальною мережею Instagram дозволяє автоматично отримувати аудитовані метрики акаунтів і використовувати їх як основу для відбору партнерів і оцінки ефективності кампаній. Модульна монолітна архітектура забезпечує гнучкість підтримки і розвитку платформи. Асинхронна обробка подій через RabbitMQ та саги гарантує надійність бізнес-процесів навіть за збоїв інфраструктури.

У 2024 році глобальний обсяг ринку інфлюенсер-маркетингу досяг 24 млрд доларів США, а Instagram залишається основною платформою для рекламних кампаній [13, 30]. Попри це, процеси пошуку інфлюенсерів, узгодження умов і аналізу результатів досі ведуться у розрізнених інструментах, що знижує ефективність управління. Наявні програмні рішення орієнтовані переважно на корпоративний сегмент і не охоплюють повного циклу кампанії в єдиному середовищі, що зумовлює актуальність розробки спеціалізованої вебплатформи.

Метою кваліфікаційної роботи є розробка вебплатформи управління рекламними кампаніями в соціальній мережі Instagram.

Для досягнення поставленої мети необхідно вирішити такі завдання: проаналізувати предметну область цифрового маркетингу та процеси організації рекламних кампаній в Instagram; виконати огляд існуючих програмних рішень для управління інфлюенсер-маркетингом і визначити їх обмеження; обґрунтувати вибір архітектури та інструментальних засобів реалізації; спроєктувати структуру програмної платформи та модель даних; реалізувати серверну та клієнтську частини вебплатформи; провести тестування розробленого програмного продукту.

Об’єктом дослідження є процеси управління рекламними кампаніями у соціальній мережі Instagram, що охоплюють весь життєвий цикл співпраці між рекламодавцями та інфлюенсерами. Розгляд цих процесів передбачає аналіз

способів отримання та обробки достовірних аналітичних даних з Instagram, механізмів комунікації між учасниками й автоматизації переходів між станами рекламної кампанії.

Предметом дослідження є методи та програмні засоби підтримки управління рекламними кампаніями в Instagram у межах цифрового маркетингу.

Практичне значення одержаних результатів полягає у створенні працездатної вебплатформи, яка забезпечує централізоване управління рекламними кампаніями в Instagram на основі достовірних показників аудиторії, отриманих через офіційне Meta Graph API. Розроблена платформа може бути використана блогерами та брендами для організації співпраці у сфері інфлюенсер-маркетингу, а описані архітектурні та технологічні рішення можуть слугувати основою для розробки аналогічних систем у суміжних галузях цифрового маркетингу.

За темою випускної кваліфікаційної роботи було опубліковано тези [1, 2].

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

1.1 Постановка задачі

У межах кваліфікаційної роботи необхідно дослідити предметну область управління рекламними кампаніями у соціальній мережі Instagram та сформувати формалізоване уявлення про задачу, що підлягає подальшому програмному розв’язанню [6]. Постановка задачі передбачає визначення цілей створення програмної платформи, основних функціональних можливостей та обмежень, а також кола користувачів і їхніх потреб.

Потенційними користувачами платформи є як індивідуальні блогери, так і бренди, які зацікавлені у проведенні рекламних кампаній у соціальній мережі Instagram. Вони можуть як замовляти рекламу у інших учасників, так і самі надавати рекламні послуги, тому від системи очікується підтримка обох сценаріїв взаємодії.

Основною проблемою організації такої взаємодії є відсутність достовірної інформації про потенційного партнера. Під час пошуку виконавця замовник змушений покладатися на самостійно наведені блогером дані про аудиторію та охоплення, достовірність яких неможливо перевірити. Це підвищує ризики неефективної співпраці та витрачання рекламного бюджету на партнерів із накрученими або неякісними показниками.

Крім того, процеси пошуку партнера, узгодження умов співпраці, контролю виконання та аналізу результатів зазвичай ведуться у розрізнених інструментах: месенджерах, електронних таблицях та окремих аналітичних сервісах. Через це втрачається історія домовленостей, ускладнюється контроль статусу кампанії та знижується загальна ефективність рекламної діяльності.

З огляду на це, задачею кваліфікаційної роботи є розробка централізованої вебплатформи, яка дає змогу знайти блогера або бренд за реальними, підтвердженими показниками, розпочати з ним спілкування та провести рекламну кампанію від пошуку партнера до її завершення. Платформа має забезпечувати підключення облікового запису Instagram із отриманням

достовірних показників аудиторії, пошук та фільтрацію партнерів за цими показниками, опис умов співпраці, внутрішнє листування, а також фіксацію та контроль виконання рекламної кампанії у єдиному середовищі.

При цьому платформа виступає саме як середовище для пошуку партнерів та організації співпраці і не підмінює собою офіційний рекламний кабінет Instagram та не публікує рекламний контент замість користувача. Розмежування цих меж дозволяє зосередитись на ключовій цінності, якою є достовірний пошук партнера та прозоре ведення кампанії.

На основі аналізу виявлених проблем сформульовано перелік функціональних вимог до платформи: підключення облікового запису Instagram через Meta Graph API для отримання достовірних метрик аудиторії і показників залученності; періодична синхронізація аналітичних даних для підтримання актуальності показників аудиторії блогерів; пошук блогерів з фільтрацією за аудиторними показниками, тематикою контенту та іншими параметрами; управління пропозиціями (оферами) та умовами співпраці (пактом), на основі яких рекламодавць ініціює кампанію; внутрішнє листування між учасниками кампанії для узгодження умов і обговорення деталей співпраці; супровід рекламної кампанії від створення до завершення з контролем статусу, підтвердженням виконань і збором відгуків;формування сповіщень про зміну статусу кампанії, отримання нових заявок і повідомлень про події в системі.

Крім функціональних вимог, до платформи висуваються нефункціональні вимоги: доступність платформи через веббраузер без встановлення додаткового програмного забезпечення, адаптивність інтерфейсу до різних розмірів екрана, швидке завантаження сторінок і пошукових запитів. Система має забезпечувати безпечну взаємодію із зовнішніми API-сервісами, надійне зберігання даних та стійкість до збоїв за рахунок асинхронної обробки подій.

Таким чином, основними функціями платформи є підключення та верифікація облікового запису Instagram, пошук і відбір партнерів за підтвердженими показниками, ведення перемовин, оформлення умов співпраці та пропозицій, а також супровід рекламної кампанії від початку до завершення

із фіксацією виконаних дій. Визначені функції та потреби користувачів становлять основу для формування вимог до системи та її подальшого проєктування і реалізації.

1.2 Аналіз аналогів програмного продукту

У ході дослідження було розглянуто найбільш поширені програмні рішення, які використовуються для пошуку інфлюенсерів, аналізу їх аудиторії, планування рекламних кампаній та оцінювання їх ефективності. Серед таких платформ було обрано HypeAuditor, Upfluence та AspireIQ.

HypeAuditor є програмною платформою, орієнтованою на аналітику акаунтів інфлюенсерів у соціальних мережах. Основним призначенням даного рішення є оцінювання якості аудиторії, рівня залученості підписників та виявлення фейкової активності. Платформа активно використовується маркетологами для перевірки надійності інфлюенсерів перед початком рекламної співпраці.

Інтерфейс платформи надає доступ до детальних статистичних показників, включаючи демографічні характеристики аудиторії, географічне розташування підписників, динаміку росту акаунта та показники взаємодії з контентом (рис. 1.1).

До переваг платформи HypeAuditor належать розвинені аналітичні інструменти для оцінювання якості аудиторії, можливість виявлення штучних підписників та неорганічної активності, зручна візуалізація статистичних даних та підтримка аналізу акаунтів у кількох соціальних мережах.

Серед недоліків HypeAuditor варто виділити відсутність інструментів повноцінного управління рекламними кампаніями, неможливість організації взаємодії між рекламодавцями та інфлюенсерами в межах платформи та орієнтацію переважно на аналітичні задачі, а не на повний цикл рекламної кампанії.

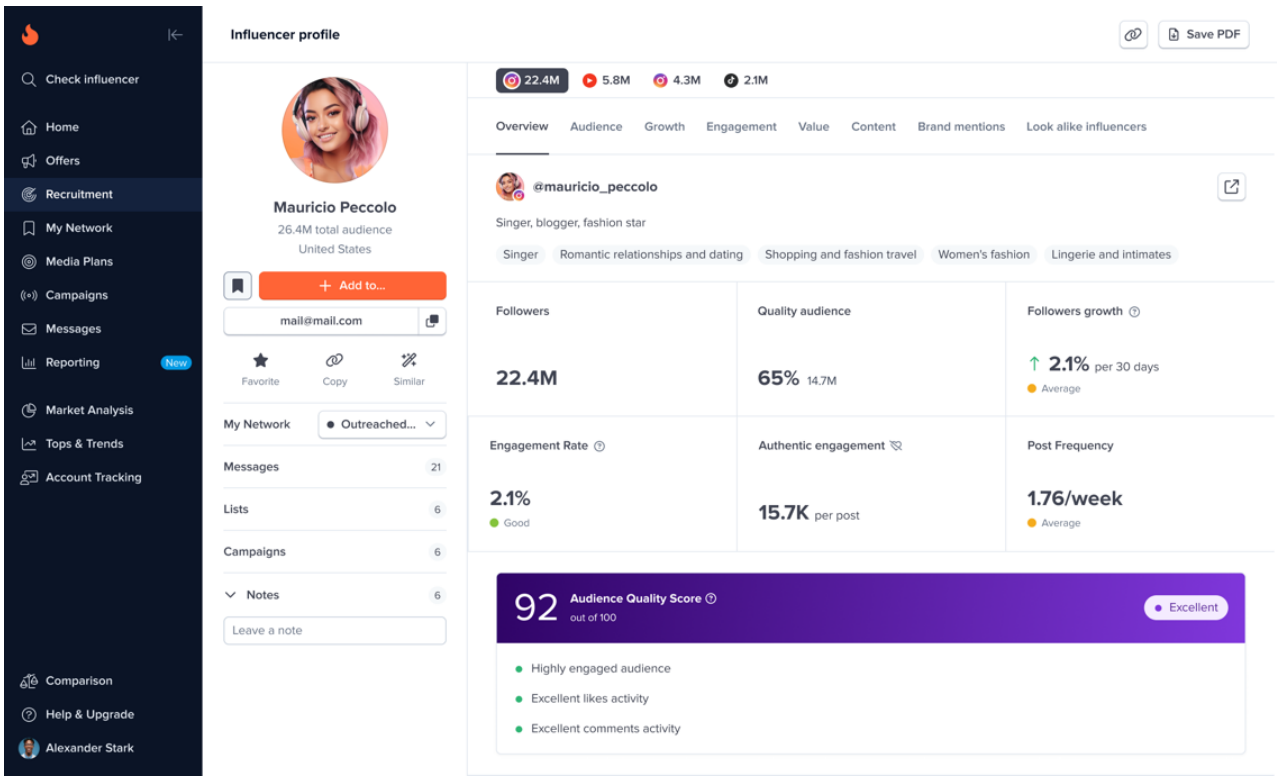


Рис. 1.1. Інтерфейс аналітичних даних інфлюенсера платформи HypeAuditor

Upfluence є комплексною програмною платформою для управління інфлюенсер-маркетингом, яка поєднує інструменти пошуку інфлюенсерів, управління співпрацею та аналізу ефективності рекламних кампаній. Платформа використовується брендами та маркетинговими агенціями для централізованого керування рекламною діяльністю в соціальних мережах. Upfluence надає можливість фільтрації інфлюенсерів за різними параметрами, зберігання інформації про співпрацю, а також інтеграцію з іншими маркетинговими системами.

Серед переваг платформи Upfluence можна виділити широкий функціонал для пошуку та відбору інфлюенсерів, підтримку управління рекламними кампаніями на різних етапах, можливість аналізу результатів рекламних інтеграцій та розвинену інтеграцію із зовнішніми сервісами, зокрема з платформами електронної комерції.

До недоліків Upfluence належать складний інтерфейс користувача, що потребує тривалого освоєння, висока вартість використання, орієнтація переважно на великі компанії та рекламні агенції, а також надмірний функціонал

для невеликих рекламних кампаній. Зовнішній вигляд ін терфейсу пошуку блогерів наведено на рис. 1.2.

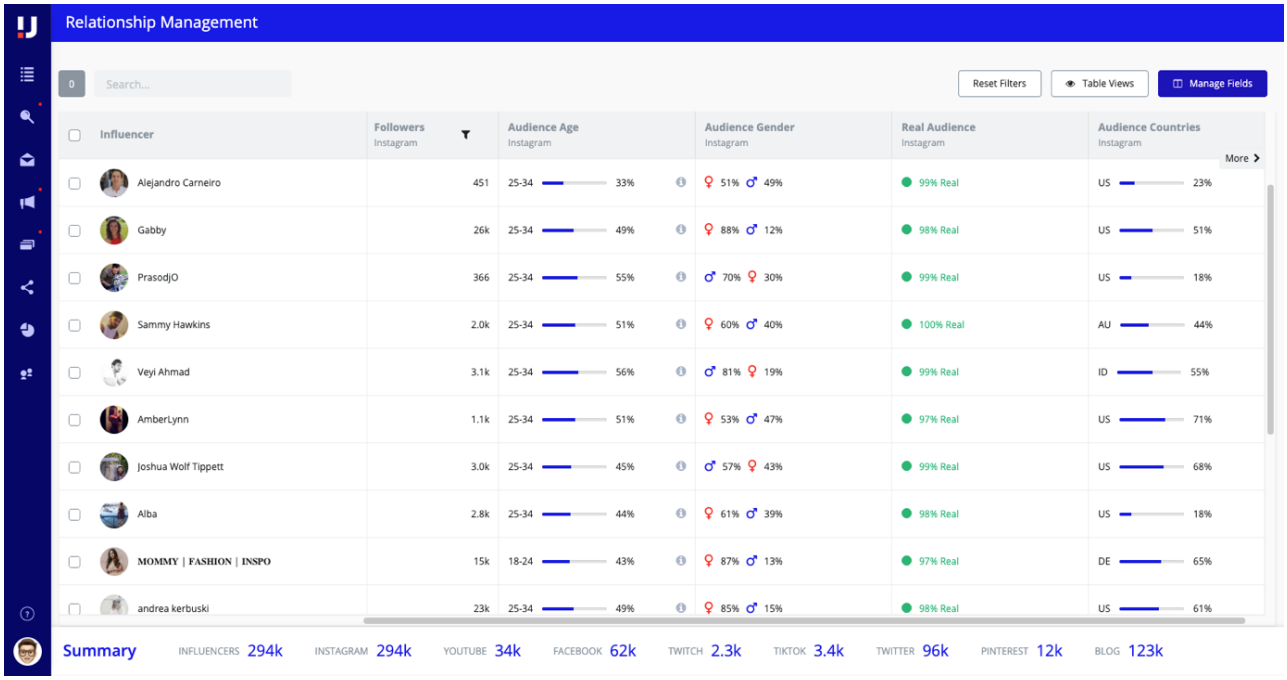


Рис. 1.2. Інтерфейс пошуку блогерів платформи Upfluence

AspireIQ є програмною платформою, основна мета якої полягає у побудові довгострокових партнерських відносин між брендами та інфлюенсерами. Платформа надає інструменти для управління спільнотами творців контенту, планування рекламних кампаній та координації процесу створення контенту.

AspireIQ дозволяє централізовано керувати взаємодією з інфлюенсерами, зберігати історію співпраці та контролювати виконання умов рекламних кампаній.

Перевагами платформи AspireIQ є орієнтація на довгострокову співпрацю з інфлюенсерами, інструменти управління контентом та комунікацією, підтримка аналітики результатів кампаній та зручна організація роботи з великими командами.

Серед недоліків AspireIQ виділяють високий поріг входу для нових користувачів, значні фінансові витрати на використання, складність налаштування для невеликих проєктів та орієнтацію переважно на корпоративних клієнтів. Загальний вигляд інтерфейсу платформи наведено на рис. 1.3.

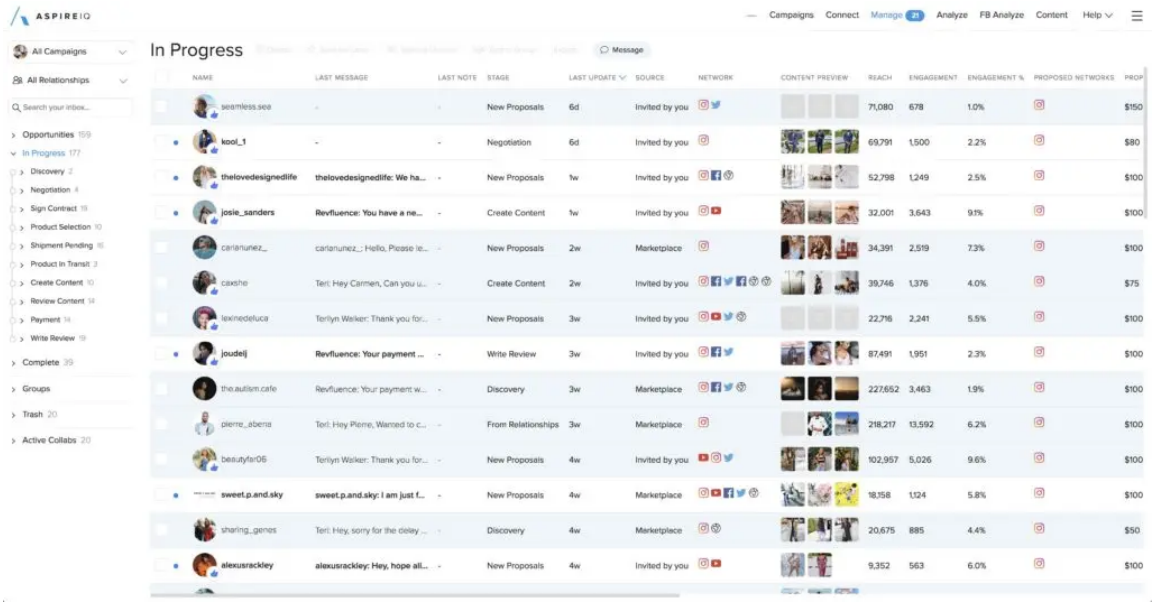


Рис. 1.3. Інтерфейс платформи AspireIQ

На основі проведеного аналізу було сформовано порівняльну характеристику розглянутих програмних платформ за основними критеріями (табл. 1.1).

Таблиця 1.1

Порівняльна характеристика розглянутих програмних платформ

Характеристика	**HypeAuditor**	**Upfluence**	**AspireIQ**	**Lanka**
Пошук інфлюенсерів	+	+	+	+
Аналітика акаунтів	+	+	+	+
Виявлення фейкової аудиторії	+	-	-	-
Управління рекламними кампаніями	-	+	+	+
Централізована комунікація	-	+	+	+
Планування рекламних інтеграцій	-	+	+	+

продовж. табл. 1.1

Контроль виконання умов кампанії	-	+	+	+
Орієнтація на малий бізнес	-	-	-	+
Простота використання	+	-	-	+
Інтегрована система обміну повідомленнями	-	+	+	+
Підтримка повного циклу від пошуку до оплати	-	-	-	+

Порівняння показує, що жодна з розглянутих платформ не покриває всіх ключових критеріїв одночасно. HypeAuditor сильна в аналітиці, але не підтримує управління кампаніями. Upfluence та AspireIQ закривають управлінську частину, проте складні у використанні та орієнтовані на великий бізнес. Lanka поєднує аналітику й простоту інтерфейсу HypeAuditor з повним циклом управління та централізованою комунікацією Upfluence і AspireIQ, додатково забезпечуючи підтримку всіх етапів співпраці від пошуку партнера до проведення оплати та орієнтуючись на малий бізнес.

1.3 Вибір архітектури системи

Одним із ключових етапів проєктування програмної платформи управління рекламними кампаніями є вибір архітектури системи. Архітектурні рішення визначають структуру програмного забезпечення, принципи взаємодії між його компонентами, а також можливості масштабування та подальшого розвитку системи.

З урахуванням специфіки предметної області та результатів аналізу аналогів було розглянуто два основні підходи до побудови архітектури програмних систем: монолітну архітектуру та мікросервісну архітектуру.

Монолітна архітектура передбачає побудову системи у вигляді єдиного програмного застосунку, в межах якого зосереджені всі компоненти бізнес-логіки, доступу до даних та взаємодії з користувачем [8]. Перевагами такого підходу є простота розробки, розгортання та тестування системи, а також відсутність складної міжсервісної взаємодії. Водночас зі зростанням функціональності монолітна архітектура ускладнює супровід системи, знижує гнучкість масштабування та ускладнює внесення змін у окремі частини програмного забезпечення. Схематичне зображення монолітної архітектури наведено на рис. 1.4.

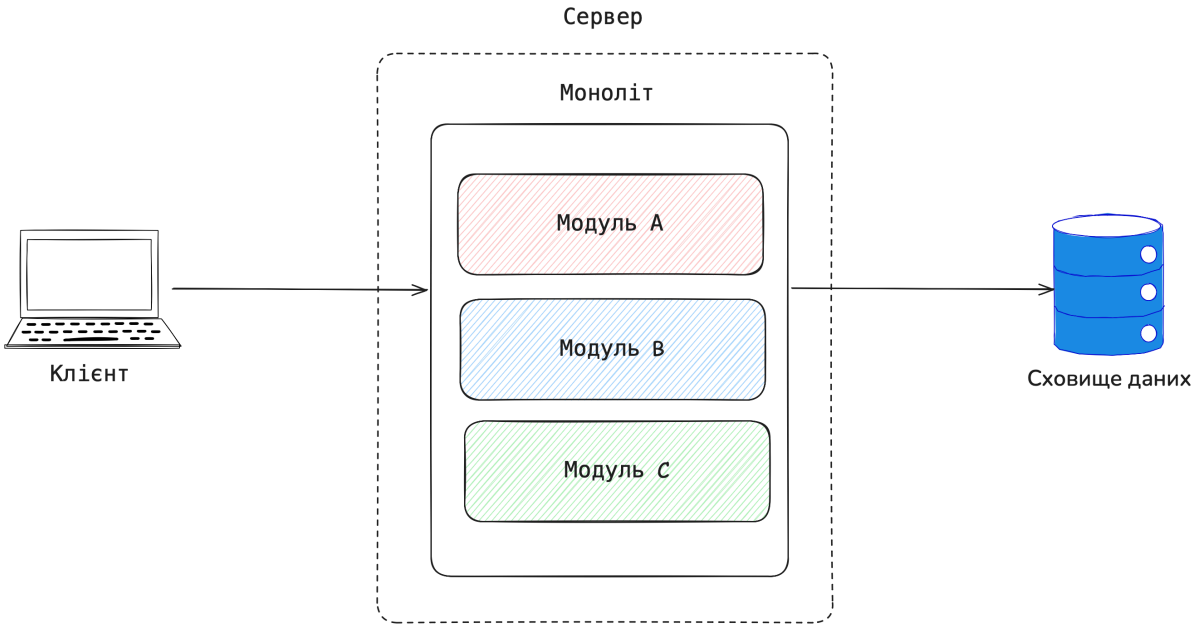


Рис. 1.4. Схематичне зображення монолітної архітектури

Мікросервісна архітектура базується на поділі системи на набір незалежних сервісів, кожен з яких реалізує окрему бізнес-функцію та взаємодіє з іншими компонентами через чітко визначені інтерфейси [25]. Такий підхід забезпечує високу масштабованість, гнучкість та незалежність розвитку окремих компонентів, проте супроводжується підвищеною складністю проєктування, розгортання та адміністрування, що не завжди є доцільним на початкових етапах розробки або для проєктів середнього рівня складності. Схематичне зображення мікросервісної архітектури наведено на рис. 1.5.

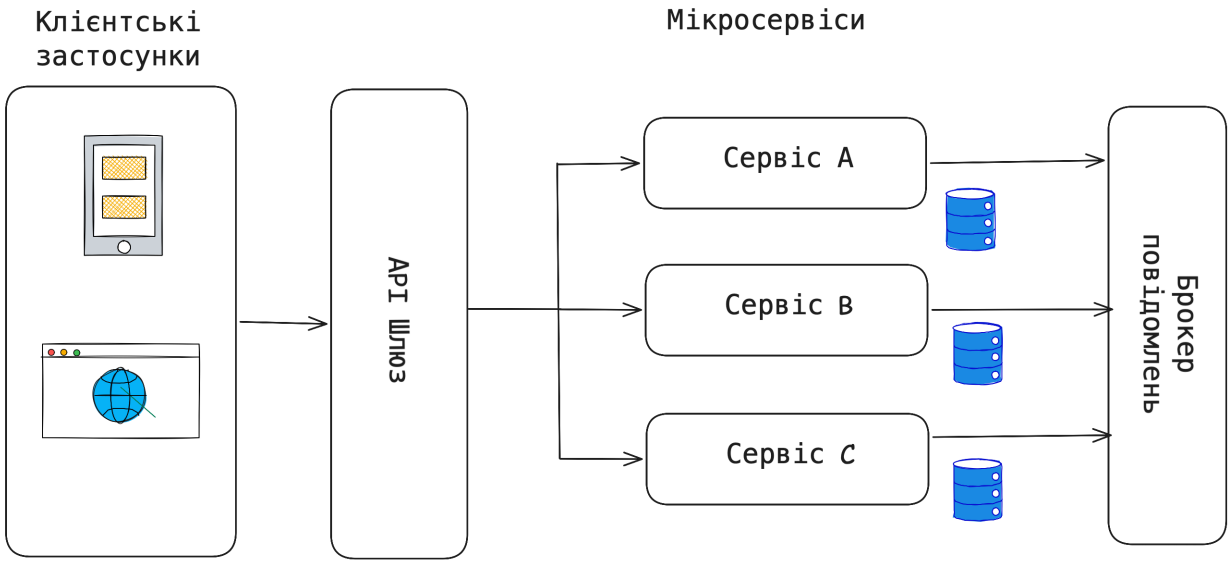


Рис. 1.5. Схематичне зображення мікросервісної архітектури

З урахуванням зазначених особливостей для розробки програмної платформи управління рекламними кампаніями обрано компромісний підхід, а саме модульну монолітну архітектуру [7, 11]. Даний підхід передбачає побудову монолітного застосунку з чітким розмежуванням функціональних модулів, кожен з яких відповідає за окрему підсистему. Така структура дозволяє зберегти переваги монолітної архітектури на етапі розробки та водночас створює передумови для подальшого виділення окремих модулів у незалежні сервіси у разі зростання навантаження або функціональних вимог. Схематичне зображення архітектури модульного моноліту наведено на рис. 1.6.

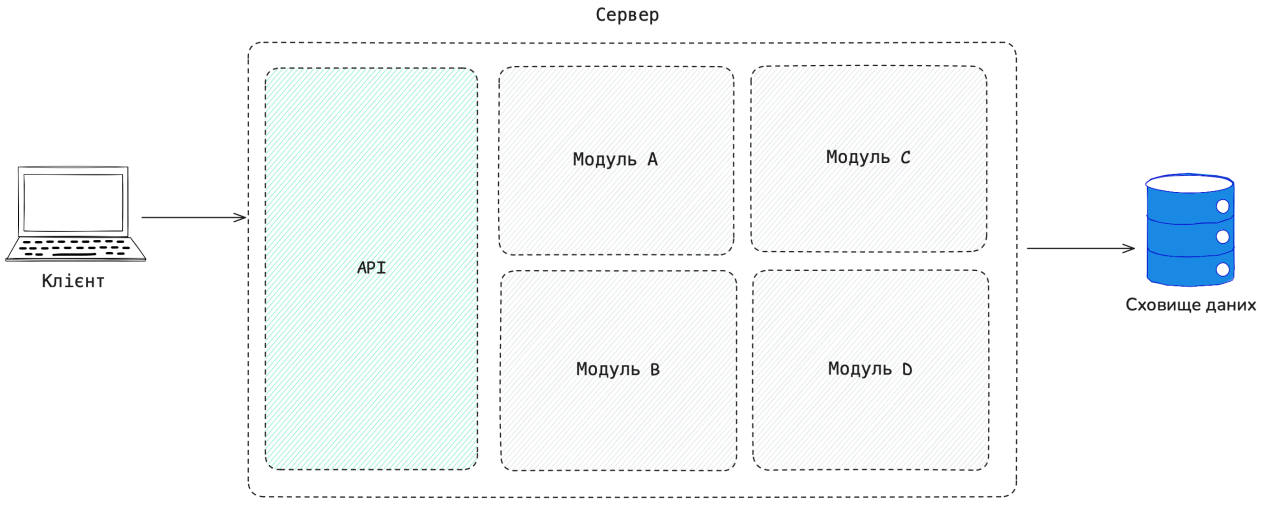


Рис. 1.6. Схематичне зображення архітектури модульного моноліту

Архітектура системи реалізується відповідно до клієнт-серверного підходу, при якому клієнтська частина функціонує у веббраузері користувачів та

забезпечує взаємодію з користувачем, а серверна частина розміщується на хостингу та відповідає за обробку запитів, реалізацію бізнес-логіки та зберігання даних. Такий підхід є класичним для сучасних веборієнтованих інформаційних систем і забезпечує зручність використання, централізоване керування даними та можливість масштабування.

Обраний архітектурний підхід забезпечує баланс між простотою реалізації, гнучкістю системи та можливістю її подальшого розвитку. Він дозволяє адаптувати програмну платформу до зростання кількості користувачів та функціональних вимог без необхідності повного перегляду архітектурних рішень.

1.4 Обґрунтування вибору інструментальних засобів

Технологічний стек для платформи управління рекламними кампаніями обирався з урахуванням специфіки предметної області, функціональних вимог та обраної модульної монолітної архітектури. Критеріями вибору були продуктивність, наявність документації, сумісність компонентів стеку між собою та відкрита ліцензія, що дозволяє адаптувати компоненти під потреби системи.

Серверну частину реалізовано мовою C# на платформі .NET. C# належить до об'єктно-орієнтованих мов зі статичною типізацією, що дає можливість виявляти значну частину помилок на етапі компіляції. C# підтримує дженерики, асинхронні операції через ключові слова async/await та LINQ для роботи з даними.

Веб-серверну частину побудовано на основі ASP.NET Core, кросплатформеного фреймворку з відкритим кодом, який за результатами незалежних тестів TechEmpower демонструє одні з найвищих показників продуктивності серед поширених рішень [20, 31]. ASP.NET Core має вбудовану підтримку інверсії залежностейта забезпечує побудову REST API через контролери або Minimal API.

Для координації .NET-сервісів та інтеграції з супутніми залежностями застосовано .NET Aspire, стек для побудови cloud-native застосунків. Aspire бере

на себе налаштування service discovery, керування конфігурацією, телеметрію на базі OpenTelemetry, а також автоматичне підняття інфраструктурних контейнерів (бази даних, кеш, брокер повідомлень) у середовищі розробки, що суттєво скорочує обсяг шаблонного коду для початкового запуску системи.

Для зберігання даних застосовано дві системи керування базами даних: реляційну та документну. Такий поліглот-підхід, за якого тип сховища обирається відповідно до характеру даних, є усталеною практикою для систем зі змішаним навантаженням [15].

Основою виступає PostgreSQL, у якій зберігаються структуровані дані: інформація про користувачів, рекламні кампанії, пакти, офери, відгуки та фінансові операції. До переваг PostgreSQL належать підтримка ACID-транзакцій, наявність кількох типів індексів (B-tree, GIN, GiST), підтримка типу JSON та відкрита ліцензія [32].

На стовпцях зовнішніх ключів та полях, що використовуються у фільтрах (status, created_at), створено індекси для прискорення типових вибірок. Підтримка типу JSONB дозволяє зберігати у PostgreSQL метадані Instagram-акаунтів без введення окремих таблиць для кожного варіанту структури профілю.

Поряд із PostgreSQL використовується MongoDB для збереження аналітичних метрик та слабоструктурованих даних, оскільки документна модель не вимагає жорсткої схеми й відповідає природі JSON-подібних структур [24]. У MongoDB зберігаються історичні зрізи показників Instagram-акаунтів та агрегована статистика аудиторії.

У ролі ORM застосовано Entity Framework Core [22]. Інструмент уможливлює опис моделі даних класами C#, а вбудований механізм міграцій спрощує версіонування схеми та її розгортання у різних оточеннях.

Доступ до даних організовано на основі патернів Repository та Unit of Work [9]. Repository інкапсулює логіку доступу до агрегатів, а Unit of Work забезпечує атомарність змін у межах однієї бізнес-операції.

Відповідно до підходу Domain-Driven Design агрегати взаємодіють з інтерфейсами репозиторіїв, а не з класами Entity Framework Core безпосередньо. Інтерфейс репозиторію описує операції з точки зору предметної області, наприклад отримання кампанії за ідентифікатором або збереження нового оферу, тоді як реалізація визначає, яким чином ці операції транслюються в запити до бази даних. Така організація дозволяє підмінити реалізацію сховища без змін у доменній моделі, що використовується при модульному тестуванні, коли реальна база даних замінюється тестовим двійником.

Кешування реалізовано засобами Redis, у якому зберігаються результати ресурсоємних запитів, дані користувацьких сесій та проміжні обчислення. Redis є сховищем типу «ключ-значення», що зберігає дані в оперативній пам'яті. Окрім простих значень Redis підтримує списки, множини, відсортовані множини та хеш-таблиці.

Асинхронний обмін повідомленнями між модулями організовано через брокер RabbitMQ [27]. Такий підхід знижує зв'язаність компонентів та спрощує їх незалежне масштабування. RabbitMQ реалізує протокол AMQP, надає гарантії доставки повідомлень, маршрутизацію повідомлень за визначеними правилами та повторну доставку у разі помилок.

У межах платформи черги RabbitMQ застосовуються для передачі подій між модулями: модуль сповіщень підписується на події зміни статусу кампанії, а модуль кампаній отримує підтвердження про проведення оплати від платіжного модуля. У разі тимчасової недоступності одного з модулів повідомлення зберігаються у черзі та доставляються після відновлення роботи.

Клієнтську частину написано мовою TypeScript. Статична типізація дає змогу виявляти розбіжності між структурами даних клієнта та серверним API на етапі збірки. Для всього проєкту увімкнено суворий режим компіляції (strict mode), що включає перевірку null-безпеки та неявних перетворень типів.

Інтерфейс побудовано на Angular, фреймворку для односторінкових застосунків, який від початку розроблявся з підтримкою TypeScript і містить вбудований DI-контейнер та модульну структуру [10, 23]. Angular включає

засоби маршрутизації, побудови форм, валідації введених даних та реактивного програмування через RxJS. Модульна структура дозволяє ділити застосунок на функціональні блоки із завантаженням за потребою.

Усі компоненти системи запаковано у Docker-контейнери. Завдяки цьому поведінка застосунку лишається однаковою на машині розробника, у CI-середовищі та на промисловому сервері, а відмінності між оточеннями практично нівелюються. Локальна оркестрація реалізована через Docker Compose.

Автентифікацію та авторизацію реалізовано на основі Keycloak, відкритого рішення для управління ідентифікацією з підтримкою протоколів OAuth 2.0 та OpenID Connect [33]. Keycloak виступає окремим сервісом, що зберігає облікові дані користувачів і видає JWT-токени. Серверна частина перевіряє токени на рівні middleware і не звертається до власної бази даних при обробці кожного запиту.

Висновки до першого розділу

У першому розділі поставлено задачу на розробку програмної платформи для управління рекламними кампаніями в Instagram та визначено основні функціональні модулі, необхідні для підтримки повного циклу взаємодії брендів та інфлюенсерів.

Виконано порівняльний аналіз існуючих програмних аналогів HypeAuditor, Upfluence та AspireIQ, визначено їх переваги, недоліки та обмеження, що обґрунтовує доцільність розробки власної платформи.

На основі аналізу предметної області та вимог до системи обґрунтовано вибір інструментальних засобів реалізації. Для серверної частини обрано платформу Node.js з фреймворком NestJS, для клієнтської частини обрано Angular, як систему управління базами даних обрано PostgreSQL, а для асинхронного обміну повідомленнями між модулями системи обрано RabbitMQ.

Було проведено порівняння монолітної та мікросервісної архітектури та обрано модульну монолітну архітектуру як компромісний підхід, що поєднує простоту розробки з можливістю подальшого масштабування.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ УПРАВЛІННЯ РЕКЛАМНИМИ КАМПАНІЯМИ В INSTAGRAM

2.1 Визначення вимог до системи

На етапі проєктування системи управління рекламними кампаніями в соціальній мережі Instagram було визначено основні варіанти використання програмної платформи, які описують взаємодію користувачів із системою. Формування варіантів використання здійснювалося на основі аналізу предметної області та визначених функціональних вимог до платформи.

У межах системи передбачено одну основну роль, а саме користувача платформи, який залежно від контексту може створювати рекламні кампанії з метою пошуку інфлюенсерів або брати участь у кампаніях інших користувачів як виконавець рекламних інтеграцій. Такий підхід забезпечує гнучкість системи та дозволяє одному й тому ж користувачу виконувати різні ролі залежно від поточних потреб. Перелік основних варіантів використання системи зображено на діаграмі прецедентів (рис. 2.1).

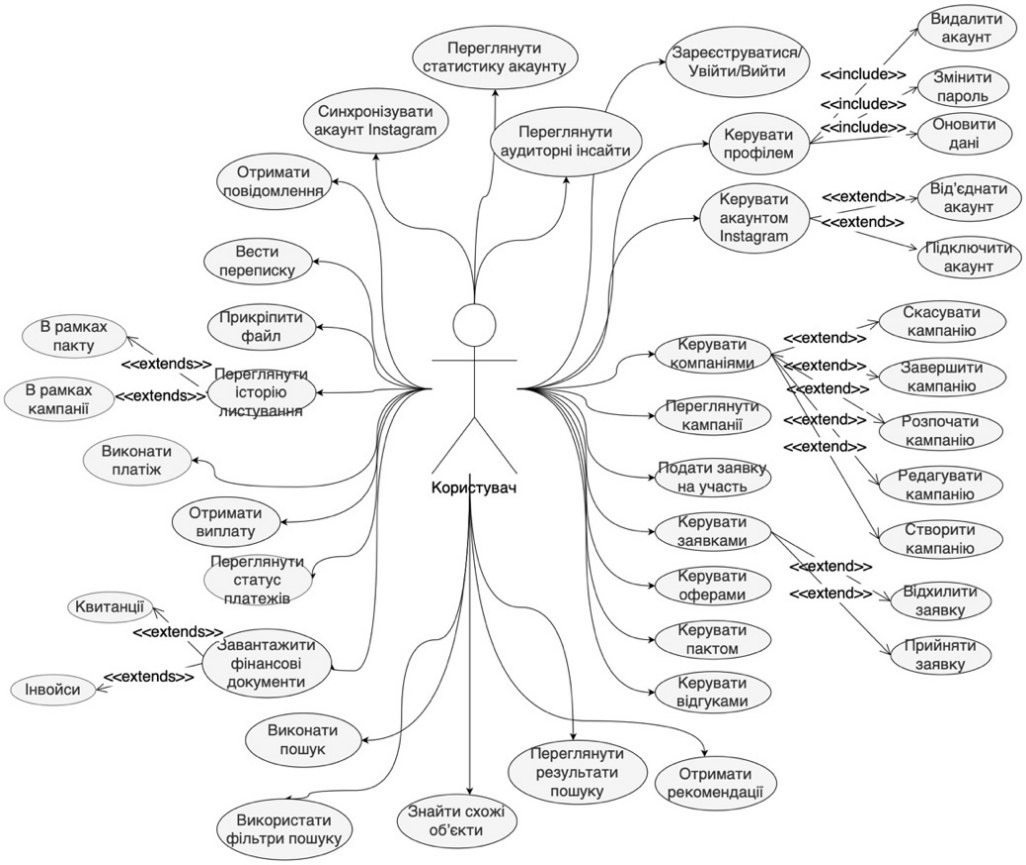


Рис. 2.1. Варіанти використання системи

Початкова взаємодія користувача з системою забезпечується варіантом використання реєстрації та автентифікації, який передбачає створення облікового запису, вхід до системи та вихід з нього. Після проходження автентифікації користувач отримує доступ до керування власним профілем, що дозволяє переглядати персональні дані, оновлювати їх або видаляти акаунт. Для забезпечення основної функціональності платформи передбачено керування акаунтом Instagram, яке охоплює підключення та від'єднання бізнес-акаунта, ініціювання його синхронізації, а також перегляд статистики та аналітичних метрик профілю.

Ключову частину взаємодії в системі становить робота з рекламними кампаніями. Користувач формує загальні умови співпраці у вигляді пакту та визначає на його основі конкретні рекламні пропозиції, на яких базуються майбутні кампанії. У ролі рекламодавця користувач опрацьовує заявки, що надходять від інших учасників на основі його оферів: переглядає умови та приймає або відхиляє їх. Сам процес виконання кампанії здійснюється спільно обома сторонами після узгодження умов, які розпочинають, призупиняють, завершують або скасовують кампанію та контролюють хід виконання рекламних інтеграцій.

Допоміжні варіанти використання забезпечують підтримку основних процесів платформи. Після завершення кампанії учасники залишають зворотний зв'язок, який зберігається у профілі контрагента та формує його репутацію. Листування між учасниками дозволяє вести переписку в межах оферу або конкретної кампанії, отримувати повідомлення та переглядати історію спілкування. Окремий блок функціональності становлять пошук і рекомендації, які забезпечують виконання пошуку за ключовими параметрами, застосування фільтрів, перегляд результатів, знаходження схожих об'єктів та отримання персональних рекомендацій.

Окреслений набір варіантів використання визначає перелік предметних сутностей, якими має оперувати система. Узагальнена структура предметної

області подана на діаграмі класів (рис. 2.2), що відображає вимоги до доменної моделі без прив’язки до конкретної реалізації.

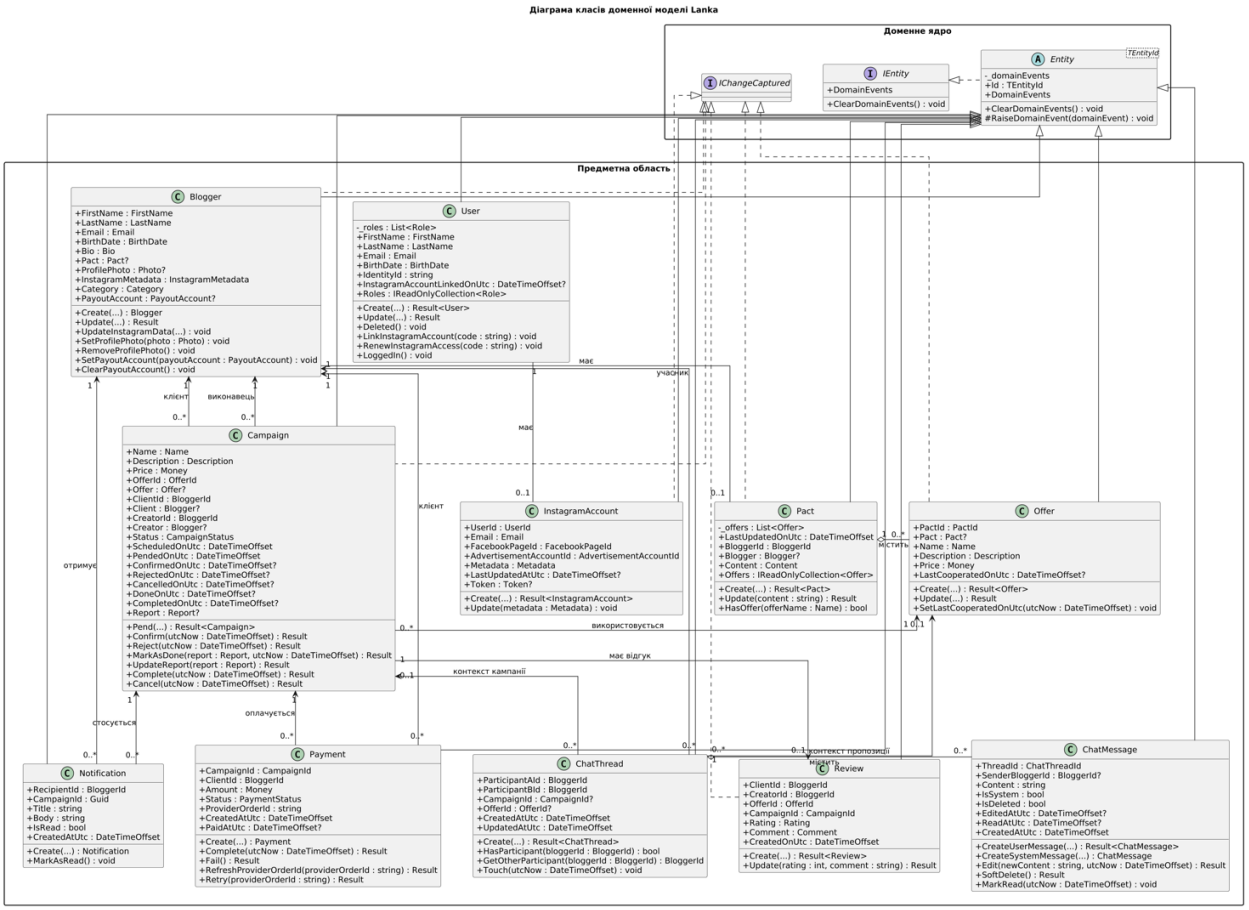


Рис. 2.2. Концептуальна модель предметної області

Центральною сутністю предметної області є користувач платформи, який має змогу виступати як замовником рекламної інтеграції, так і її виконавцем. Для роботи з аналітикою рекламних розміщень система повинна підтримувати поняття підключеного бізнес-акаунта Instagram, що зберігає метадані профілю та показники аудиторії.

Виконавча частина платформи описується сутністю блогера, для якого визначаються загальні умови співпраці (пакт) та сформований на їх основі набір конкретних рекламних пропозицій (оферів). Така структура відповідає вимозі розділення довгострокових принципів роботи блогера та поточних комерційних пропозицій.

Безпосередня взаємодія між учасниками формалізується через сутність рекламної кампанії, що відображає життєвий цикл співпраці від подання заявки до завершення або скасування. За результатами завершених кампаній система

повинна підтримувати зворотний зв’язок учасників у вигляді відгуків, що формують репутаційну складову профілю.

Для забезпечення комунікації між учасниками передбачено сутності гілки спілкування та повідомлення, які можуть бути пов’язані з конкретною кампанією або рекламною пропозицією. Окрім переписки, система має підтримувати інформування користувачів про події, що стосуються їхньої діяльності на платформі.

Фінансовий аспект взаємодії описується сутністю платежу, який фіксує оплату рекламної кампанії та її стан в зовнішній платіжній системі. Виокремлення платежу як самостійної сутності зумовлене необхідністю розмежувати виконання кампанії та її фінансове підтвердження.

Зв’язки між сутностями відображають логіку предметної області: користувач асоційований з власним Instagram-акаунтом та блогерським профілем, блогер має пакт співпраці з набором пропозицій, кампанія базується на конкретній пропозиції та об’єднує двох учасників, а відгук і платіж належать до завершеної кампанії.

2.2 Розробка бази даних системи

Для забезпечення збереження та обробки даних у системі управління рекламними кампаніями в Instagram було спроєктовано реляційну базу даних, структура якої відображає основні бізнес-процеси платформи. База даних реалізує зберігання інформації про користувачів, блогерів, рекламні пропозиції, кампанії, відгуки, платежі, повідомлення, сповіщення, а також дані, пов’язані з автентифікацією, ролями доступу та інтеграцією з Instagram. Проєктування схеми виконувалося з урахуванням принципів нормалізації, що дозволяє зменшити надлишковість даних та забезпечити цілісність інформації під час її оновлення. Кожна предметна сутність подається окремою таблицею, а зв’язки між ними реалізовано через зовнішні ключі для підтримання референційної цілісності на рівні системи керування базою даних [32]. Загальну структуру спроєктованої бази даних та зв’язки між таблицями подано на рис. 2.3.

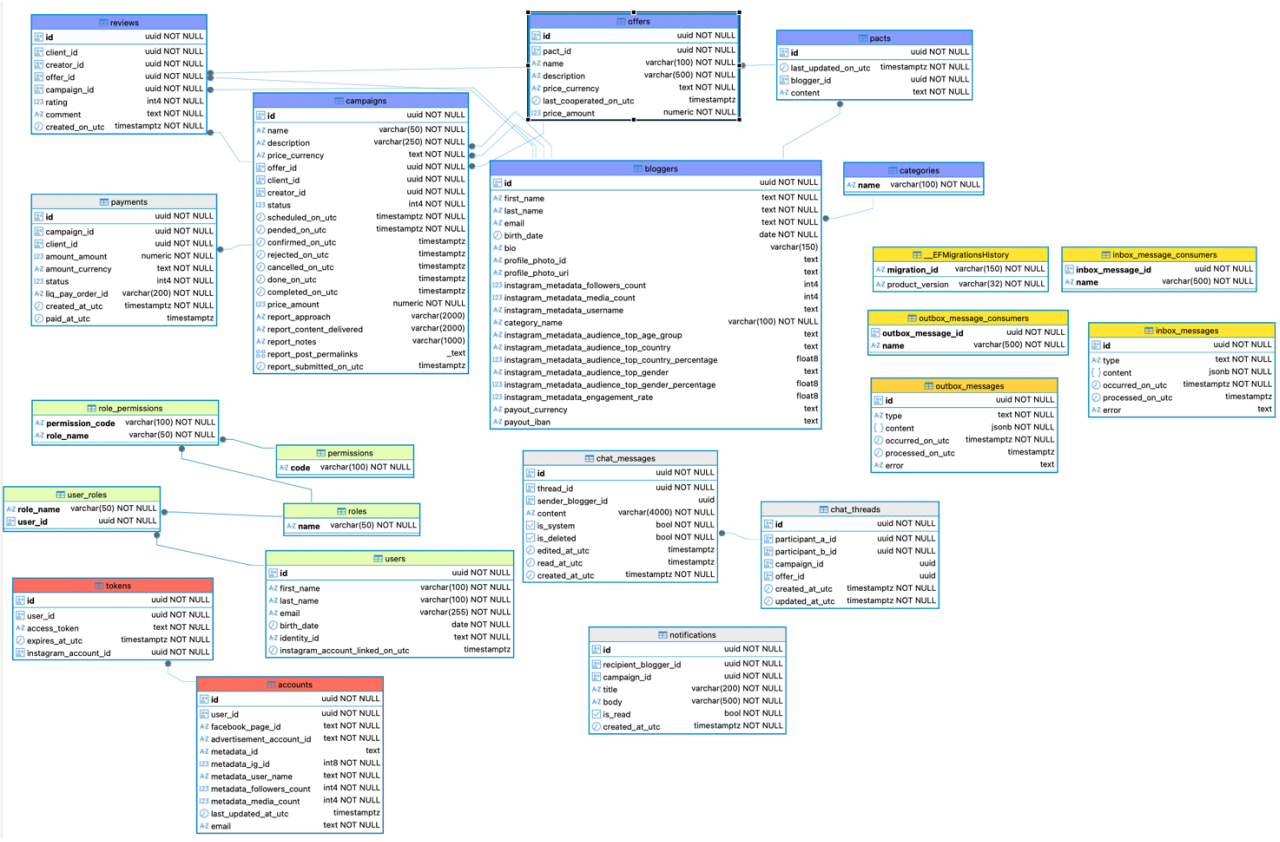


Рис. 2.3. Структура таблиць бази даних системи

Центральною таблицею бази даних є users, яка містить основну інформацію про зареєстрованих користувачів платформи: ім'я, прізвище, електронну адресу, дату народження, ідентифікатор користувача у зовнішній системі автентифікації та дату підключення Instagram-акаунта (табл. 2.1). Для реалізації рольової моделі доступу використовуються таблиці roles, permissions, role_permissions та user_roles, які забезпечують призначення користувачам відповідних ролей і прав доступу в системі (табл. 2.2). Така структура дозволяє гнучко керувати правами доступу без зміни схеми бази даних. Рольова модель може розширюватись шляхом додавання нових ролей або дозволів без модифікації існуючої логіки.

Таблиця 2.1

Опис полів таблиці "users"

Назва	Тип даних	PK	FK	Опис
id	uuid	+	-	Унікальний ідентифікатор користувача

продовж. табл. 2.1

first_name	varchar(100)	-	-	Ім’я користувача
last_name	varchar(100)	-	-	Прізвище користувача
email	varchar(255)	-	-	Електронна адреса користувача
birth_date	date	-	-	Дата народження користувача
identity_id	text	-	-	Ідентифікатор у системі автентифікації
instagram_account_linked_on_utc	timestampz	-	-	Дата та час підключення Instagram-акаунта

Таблиця 2.2

Опис полів таблиці “roles”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
id	uuid	+	-	Ідентифікатор ролі
name	varchar(50)	+	-	Назва ролі

Функціонал блогерської діяльності реалізується за допомогою таблиці bloggers (табл. 2.3), що містить інформацію про користувачів як виконавців рекламних інтеграцій, включаючи публічні дані профілю, біографію, фото профілю, платіжний акаунт та метадані Instagram-акаунта. У таблиці також зберігаються аналітичні показники, зокрема кількість підписників, кількість публікацій, рівень залученості аудиторії, основна вікова група, стать і країна аудиторії. Збереження саме показників залученості та демографії, а не лише кількості підписників, зумовлене тим, що ефективність рекламної співпраці визначається переважно якістю взаємодії інфлюенсера з аудиторією, а не розміром його бази підписників [4]. Таблиця categories використовується для класифікації блогерів за тематичними категоріями.

Таблиця 2.3

Опис полів таблиці “bloggers”

Назва	Тип даних	PK	FK	Опис
id	uuid	+	-	Унікальний ідентифікатор блогера
first_name	varchar(100)	-	-	Ім’я блогера
last_name	varchar(100)	-	-	Прізвище блогера
email	varchar(255)	-	-	Електронна адреса блогера
birth_date	date	-	-	Дата народження блогера
bio	varchar(150)	-	-	Опис профілю блогера
profile_photo_id	text	-	-	Ідентифікатор фото профілю
profile_photo_uri	text	-	-	URI фото профілю
instagram_metadata followers_count	int4	-	-	Кількість підписників Instagram
instagram_metadata media_count	int4	-	-	Кількість публікацій Instagram
instagram_metadata username	text	-	-	Ім’я користувача Instagram
category_name	varchar(100)	-	+	Назва категорії

Умови співпраці блогера з рекламодавцями зберігаються в таблиці pacts (табл. 2.4), яка містить текстовий опис правил та принципів взаємодії. На основі пактів формуються конкретні рекламні пропозиції, що зберігаються в таблиці offers (табл. 2.5) із зазначенням назви, опису, вартості, валюти та дати останньої співпраці. Таким чином, пакт визначає загальні умови взаємодії, а пропозиції конкретизують доступні рекламні послуги блогера.

Таблиця 2.4

Опис полів таблиці “pacts”

Назва	Тип даних	PK	FK	Опис
id	uuid	+	-	Унікальний ідентифікатор пакту
last_updated_on_utc	timestampz	-	-	Дата та час останнього оновлення
blogger_id	uuid	-	+	Ідентифікатор блогера (зовнішній ключ до bloggers)
content	text	-	-	Зміст пакту

Таблиця 2.5

Опис полів таблиці “offers”

Назва	Тип даних	PK	FK	Опис
id	uuid	+	-	Унікальний ідентифікатор пропозиції
pact_id	uuid	-	+	Ідентифікатор пакту (зовнішній ключ до pacts)
name	varchar(100)	-	-	Назва рекламної пропозиції
description	varchar(500)	-	-	Опис послуги
price_currency	text	-	-	Валюта ціни
price_amount	numeric	-	-	Сума вартості
last_cooperated_on_utc	timestampz	-	-	Дата та час останньої співпраці

Процес реалізації рекламних інтеграцій описується таблицею campaigns (табл. 2.6), яка зберігає інформацію про рекламні кампанії, їхній поточний статус, узгоджену вартість, валюту та часові мітки переходів між станами. Кампанія пов’язана з конкретною рекламною пропозицією, клієнтом і виконавцем. Окремі поля таблиці фіксують дату створення заявки, підтвердження, відхилення, скасування, виконання та завершення кампанії, а також містять інформацію про звіт, кількість нотаток, постійні посилання на звіт і дату подання звіту.

Таблиця 2.6

Опис полів таблиці “campaigns”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
Id	uuid	+	-	Унікальний ідентифікатор кампанії
name	varchar(50)	-	-	Назва кампанії
description	varchar(250)	-	-	Опис кампанії
price_currency	text	-	-	Валюта ціни
price_amount	numeric	-	-	Сума вартості
offer_id	uuid	-	+	Ідентифікатор пропозиції (зовнішній ключ до offers)
client_id	uuid	-	+	Ідентифікатор клієнта (зовнішній ключ до bloggers)
creator_id	uuid	-	+	Ідентифікатор виконавця (зовнішній ключ до bloggers)
status	int4	-	-	Статус кампанії
scheduled_on_utc	timestampz	-	-	Дата та час запланування
pended_on_utc	timestampz	-	-	Дата та час очікування
confirmed_on_utc	timestampz	-	-	Дата та час підтвердження

продовж. табл. 2.6

rejected_on_utc	timestampz	-	-	Дата та час відхилення
cancelled_on_utc	timestampz	-	-	Дата та час скасування
done_on_utc	timestampz	-	-	Дата та час завершення
completed_on_utc	timestampz	-	-	Дата та час виконання

Для збереження результатів взаємодії між користувачами використовується таблиця reviews (табл. 2.7), яка містить оцінки та коментарі, залишені після виконання рекламної кампанії. Таблиця пов’язана з відповідною кампанією, клієнтом, виконавцем і рекламною пропозицією, що дозволяє відстежувати якість співпраці в межах конкретної рекламної інтеграції.

Таблиця 2.7

Опис полів таблиці “reviews”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
id	uuid	+	-	Унікальний ідентифікатор відгуку
client_id	uuid	-	+	Ідентифікатор автора відгуку
creator_id	uuid	-	+	Ідентифікатор виконавця
offer_id	uuid	-	+	Ідентифікатор пропозиції
campaign_id	uuid	-	+	Ідентифікатор кампанії
rating	int4	-	-	Оцінка
comment	text	-	-	Текстовий коментар
created_on_utc	timestampz	-	-	Дата та час створення відгуку

Для реалізації інтеграції із зовнішніми сервісами, зокрема Instagram, у базі даних передбачено таблиці accounts (табл. 2.8) та tokens (табл. 2.9). Таблиця accounts зберігає ідентифікатори користувача, Facebook-сторінки, рекламного акаунта, бізнес-акаунта Instagram, а також метадані профілю, зокрема ім’я користувача, кількість підписників, кількість публікацій, дату останнього оновлення та електронну адресу. Таблиця tokens використовується для зберігання токенів доступу, часу їх завершення та зв’язку з відповідним Instagram-акаунтом.

Таблиця 2.8

Опис полів таблиці “accounts”

Назва	Тип даних	PK	FK	Опис
id	uuid	+	-	Унікальний ідентифікатор акаунта
user_id	uuid	-	+	Ідентифікатор користувача (зовнішній ключ до users)
facebook_page_id	text	-	-	Ідентифікатор сторінки Facebook
advertisement account_id	text	-	-	Ідентифікатор рекламного акаунта
metadata_id	text	-	-	Ідентифікатор метаданих
metadata_ig_id	int8	-	-	Ідентифікатор Instagram в метаданих
metadata_user_name	text	-	-	Ім'я користувача в метаданих
metadata followers_count	int8	-	-	Кількість підписників в метаданих
metadata media_count	int4	-	-	Кількість медіа в метаданих
last_updated_at_utc	timestampz	-	-	Дата та час останнього оновлення

Таблиця 2.9

Опис полів таблиці “tokens”

Назва	Тип даних	PK	FK	Опис
id	uuid	+	-	Унікальний ідентифікатор токена
user_id	uuid	-	+	Ідентифікатор користувача
access_token	text	-	-	Токен доступу до API

продовж. табл. 2.9

expires_at_utc	timestampz	-	-	Дата та час закінчення дії токена
instagram account_id	uuid	-	+	Ідентифікатор Instagram-акаунта

Фінансова частина взаємодії між користувачами реалізується за допомогою таблиці payments (табл. 2.10). Вона зберігає інформацію про оплату рекламної кампанії: ідентифікатор кампанії, клієнта, суму платежу, валюту, статус платежу, ідентифікатор замовлення у зовнішнього платіжного провайдера, дату створення та дату фактичної оплати. Це дозволяє відокремити процес виконання кампанії від процесу фінансового підтвердження співпраці.

Таблиця 2.10

Опис полів таблиці “payments”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
id	uuid	+	-	Унікальний ідентифікатор оплати
campaign_id	uuid	-	+	Ідентифікатор кампанії, у межах якої здійснено оплату
client_id	uuid	-	+	Ідентифікатор клієнта (рекламодавця), який ініціював оплату
amount_amount	numeric	-	-	Сума оплати
amount_currency	text	-	-	Валюта оплати
status	int4	-	-	Статус оплати
pay_order_id	uuid	-	+	Ідентифікатор платіжного доручення в платіжній системі
created_at_utc	timestampz	-	-	Дата й час створення запису про оплату
paid_at_utc	Timestampz	-	-	Дата й час фактичного зарахування оплати

Для реалізації комунікації між учасниками платформи передбачено таблиці chat_threads (табл. 2.11) та chat_messages (табл. 2.12). Таблиця chat_threads описує окрему гілку спілкування між двома учасниками, яка може бути пов’язана з конкретною кампанією або рекламною пропозицією. Таблиця chat_messages містить повідомлення в межах відповідної гілки, включаючи відправника, текст повідомлення, ознаку системного повідомлення, стан видалення, дату створення, редагування та прочитання.

Таблиця 2.11

Опис полів таблиці “chat_threads”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
id	uuid	+	-	Унікальний ідентифікатор гілки чату
participant_a_id	uuid	-	+	Ідентифікатор першого учасника гілки чату
participant_b_id	uuid	-	+	Ідентифікатор другого учасника гілки чату
campaign_id	uuid	-	+	Ідентифікатор кампанії, у межах якої ведеться переписка
offer_id	uuid	-	+	Ідентифікатор оферу, у межах якого ведеться переписка
created_at_utc	timestampz	-	-	Дата й час створення гілки чату
updated_at_utc	timestampz	-	-	Дата й час останнього оновлення гілки чату

Таблиця 2.12

Опис полів таблиці “chat_messages”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
id	uuid	+	-	Унікальний ідентифікатор повідомлення
thread_id	uuid	-	+	Ідентифікатор гілки чату, до якої належить повідомлення

продовж. табл. 2.12

sender_blogger_id	uuid	-	+	Ідентифікатор користувача-відправника
content	varchar(4000)	-	-	Текст повідомлення
is_system	bool	-	-	Ознака системного повідомлення
is_deleted	Bool	-	-	Ознака видаленого повідомлення
edited_at_utc	timestampz	-	-	Дата й час останнього редагування повідомлення
read_at_utc	timestampz	-	-	Дата й час прочитання повідомлення отримувачем
created_at_utc	timestampz	-	-	Дата й час надсилання повідомлення

Для інформування користувачів про важливі події системи використовується таблиця notifications (табл. 2.13). Вона містить отримувача сповіщення, пов’язану кампанію, заголовок, текст повідомлення, ознаку прочитання та дату створення. Такі сповіщення можуть використовуватися для повідомлення про зміни стану кампанії, нові повідомлення або інші дії, що потребують уваги користувача.

Таблиця 2.13

Опис полів таблиці “notifications”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
id	uuid	+	-	Унікальний ідентифікатор сповіщення
recipient_blogger_id	uuid	-	+	Ідентифікатор користувача-отримувача сповіщення
campaign_id	uuid	-	+	Ідентифікатор кампанії, з якою пов'язане сповіщення
title	varchar(200)	-	-	Заголовок сповіщення
body	varchar(500)	-	-	Текст сповіщення

продовж. табл. 2.13

is_read	bool	-	-	Ознака того, що сповіщення прочитане отримувачем
created_at_utc	timestampz	-	-	Дата й час створення сповіщення

Таблиця inbox_messages (табл. 2.14) є центральним сховищем вхідних повідомлень, що надходять до системи від зовнішніх або внутрішніх джерел. Кожен запис містить унікальний ідентифікатор, тип повідомлення, серіалізовані дані корисного навантаження та поля для відстеження стану обробки. Поля processed_on_utc та error фіксують відповідно момент успішного завершення обробки і причину збою, що дозволяє реалізувати механізм повторних спроб без втрати даних. Завдяки цьому будь-яке вхідне повідомлення гарантовано буде опрацьоване принаймні один раз, навіть у разі тимчасової недоступності обробника.

Таблиця 2.14

Опис полів таблиці “inbox_messages”

Назва	**Тип даних**	**PK**	**FK**	**Опис**
id	uuid	+	-	Унікальний ідентифікатор вхідного повідомлення
type	text	-	-	Тип повідомлення
content	jsonb	-	-	Вміст повідомлення у форматі JSON
occurred_on_utc	Timestampz	-	-	Дата та час виникнення події
processed_on_utc	timestampz	-	-	Дата та час обробки повідомлення
error	text	-	-	Опис помилки (якщо виникла)

Таблиця outbox_messages (табл. 2.15) зберігає вихідні події, що мають бути опубліковані після завершення поточної транзакції, гарантуючи надійну доставку повідомлень між модулями системи. Кожен запис містить тип події, серіалізовані дані корисного навантаження у форматі JSON та поточний статус обробки — pending, processed або failed. Поле occurred_on_utc фіксує момент виникнення події в межах бізнес-транзакції, а processed_on_utc – момент її успішної публікації брокеру. Завдяки цьому брокер повідомлень може повторно

намагатися опублікувати запис у разі тимчасового збою, не ризикуючи втратити подію або обробити її двічі.

Таблиця 2.15

Опис полів таблиці “outbox_messages”

Назва	Тип даних	PK	FK	Опис
id	uuid	+	-	Унікальний ідентифікатор вихідного повідомлення
type	text	-	-	Тип повідомлення
content	jsonb	-	-	Вміст повідомлення у форматі JSON
occurred_on_utc	timestampz	-	-	Дата та час виникнення події
processed_on_utc	timestampz	-	-	Дата та час обробки повідомлення
error	text	-	-	Опис помилки (якщо виникла)

Описана структура бази даних покриває потреби всіх основних модулів системи. Для більшості бізнес-сутностей використано окремі таблиці з прямими зв’язками між ними, що дозволяє зберігати цілісність даних під час виконання операцій. Таблиці inbox/outbox-повідомлень відокремлено від основної доменної моделі, оскільки вони обслуговують технічний механізм обміну подіями. Така організація даних підтримує поточний функціонал платформи і не потребує суттєвих змін при додаванні нових типів кампаній чи інтеграцій.

Зв’язки між таблицями відображають логіку предметної області системи: користувач має підключений Instagram-акаунт, блогер належить до певної категорії та має пакт співпраці, пакт містить набір рекламних пропозицій, кампанія створюється на основі конкретної пропозиції та пов’язує клієнта з виконавцем, а після її завершення може бути створено відгук і зафіксовано оплату. Додаткові таблиці чатів, повідомлень і сповіщень забезпечують комунікацію та інформування учасників системи, а таблиці для inbox/outbox-повідомлень підтримують узгоджену обробку подій між окремими частинами застосунку.

2.3 Проєктування та реалізація алгоритмів роботи системи

Для опису алгоритмів роботи та станів системи використовуються діаграми активності, станів та послідовності. Нижче описано основні алгоритми платформи.

Створення рекламної кампанії є одним з ключових сценаріїв платформи, який ініціює співпрацю між рекламодавцем і блогером. Спочатку система перевіряє автентифікацію користувача: якщо сесія відсутня, відбувається перенаправлення на сторінку входу. Алгоритм цього сценарію зображено на діаграмі активності (рис. 2.4).

Після успішної автентифікації користувач заповнює дані кампанії: назву, опис рекламної інтеграції, а також обирає пропозицію блогера, на основі якої будується кампанія. Система перевіряє коректність формату полів, наявність обов'язкових значень та узгодженість дат. Якщо дані не проходять валідацію, користувач отримує повідомлення з описом помилок для виправлення.

При успішній валідації система створює об'єкт Campaign з початковим статусом Pending, зберігає його в базі даних та генерує доменну подію «Campaign Created Event». Подія публікується через шину повідомлень, що дозволяє іншим модулям системи реагувати на створення кампанії, наприклад, аналітичний модуль ініціює опрацювання даних кампанії. Користувач отримує підтвердження про успішне створення і може перейти до процесу узгодження умов зі другою стороною.

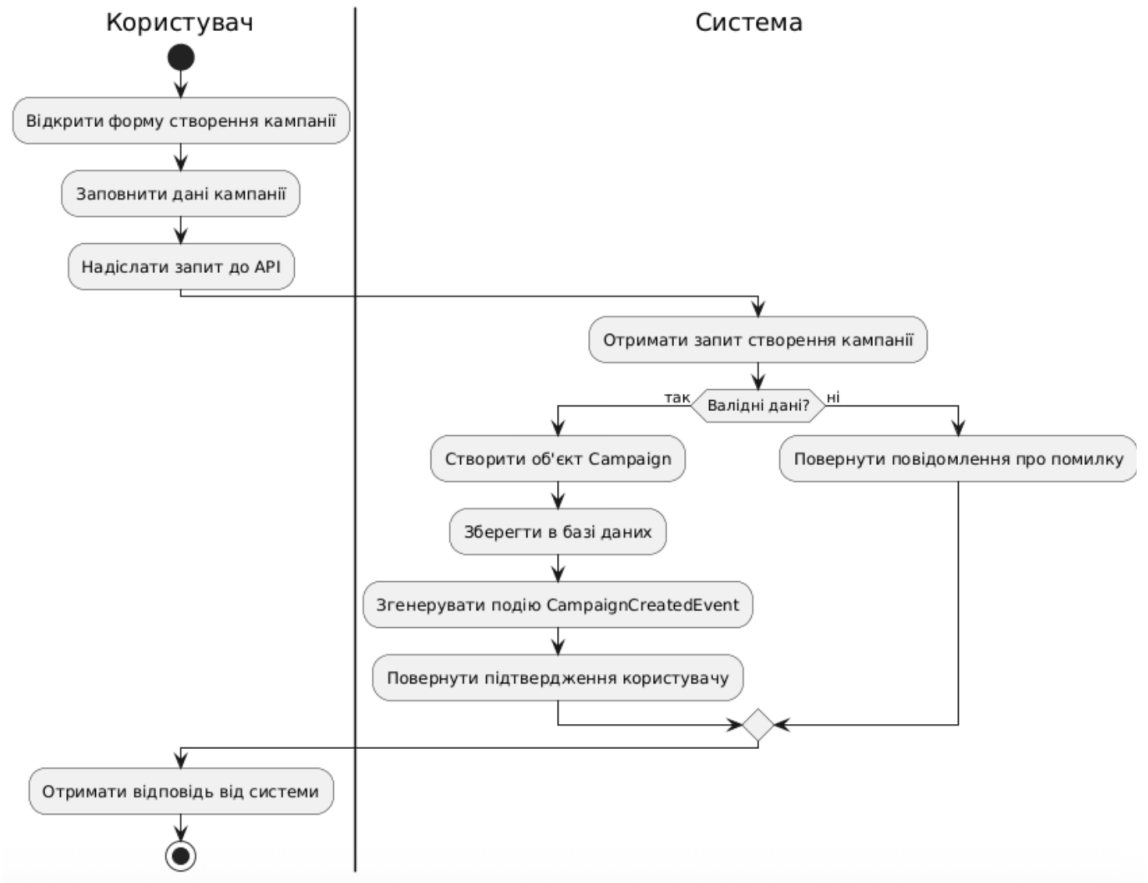


Рис. 2.4. Діаграма активності створення рекламної кампанії

Рекламна кампанія проходить кілька етапів від створення до остаточного завершення: «Очікує підтвердження» (створена, очікує підтвердження), «Підтверджена» (підтверджена обома сторонами), «Відхилена» (відхилена), «Скасована» (скасована), «Виконана» (виконана) та «Завершена» (завершена з остаточними результатами). Можливі переходи між станами та допустимі послідовності зображено на діаграмі станів (рис. 2.5).

Початковий стан кампанії «Очікує підтвердження». З цього стану можливі два переходи: до стану «Підтверджена» при підтвердженні співпраці другою стороною або до стану «Відхилена» при відмові. З підтвердженого стану кампанія може бути скасована (перехід до стану «Скасована») до моменту початку виконання або переведена в стан «Виконана» після виконання блогером зобов'язань. Зі стану «Виконана» кампанія переходить до стану «Завершена» після остаточної перевірки результатів та збору аналітики.

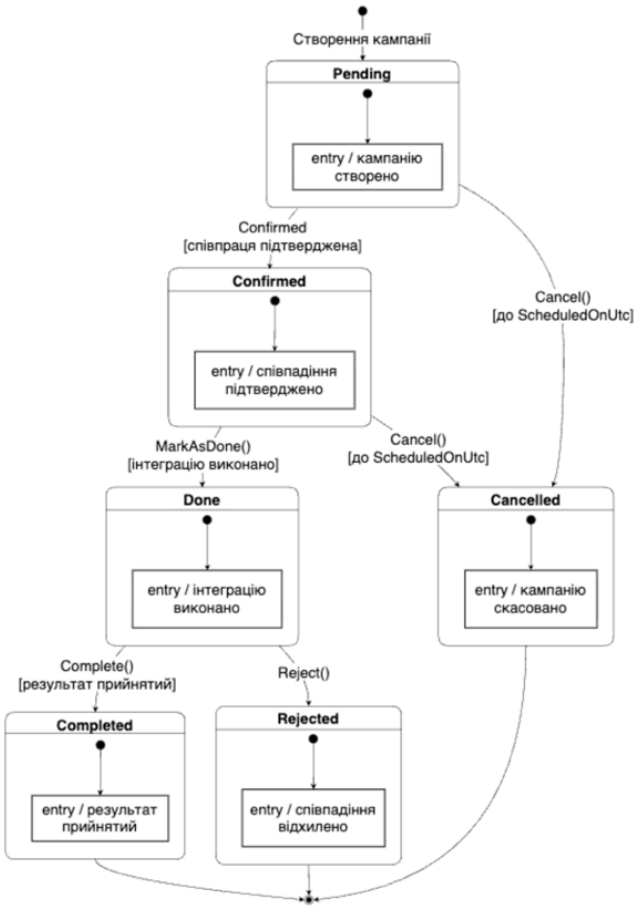


Рис. 2.5. Діаграма станів рекламної кампанії

Діаграма показує, що не всі переходи є допустимими: наприклад, неможливо скасувати кампанію після її фактичного виконання, а підтвердити або відхилити можна лише кампанію в стані «Очікує підтвердження». Такі обмеження забезпечуються через перевірки в доменних методах та на рівні бізнес-процесів.

Підключення облікового запису Instagram є одним із ключових сценаріїв роботи системи, оскільки саме через нього користувач надає платформі доступ до даних, необхідних для подальшого аналізу профілю, аудиторії та показників взаємодії. Послідовність взаємодії між компонентами системи під час виконання цього сценарію зображено на діаграмі послідовності.

Процес починається після того, як користувач проходить авторизацію на стороні Instagram і отримує код авторизації. Цей код передається до Users API за допомогою HTTP-запиту. На цьому етапі API не виконує всю операцію синхронно, оскільки підключення акаунта передбачає звернення до зовнішніх

сервісів, обмін коду на токен доступу, отримання даних профілю та збереження результату. Щоб уникнути тривалого очікування відповіді та зменшити залежність користувацького запиту від зовнішніх сервісів, операція виконується асинхронно. Повну послідовність взаємодії компонентів під час підключення акаунта зображено на діаграмі послідовності (рис. 2.6).

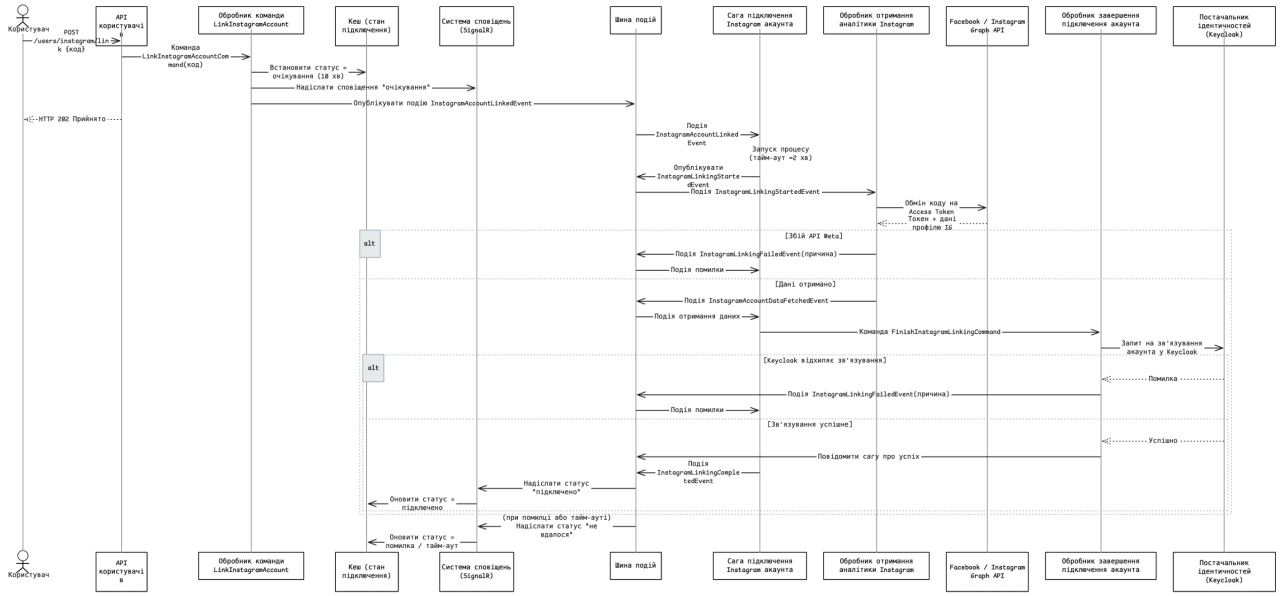


Рис. 2.6. Діаграма послідовності підключення Instagram акаунта

Після отримання запиту Users API формує команду «Link Instagram Account Command» і передає її до медіатора [3]. Далі команда обробляється відповідним обробником «Link Instagram Command Handler». Його основне завдання полягає не в повному завершенні процесу підключення, а в ініціалізації цього процесу. Обробник записує в кеш проміжний статус операції зі значенням pending, що дозволяє системі відстежувати поточний стан підключення. Одночасно через SignalR користувачу надсилається повідомлення про початок процесу, щоб інтерфейс міг одразу відобразити актуальний статус.

Після цього обробник «Link Instagram Command Handler» публікує інтеграційну подію «Instagram Account Linked Event» у шину повідомлень EventBus. Публікація події означає, що запит прийнято до обробки, але сама операція ще не завершена. Саме тому API повертає користувачу відповідь HTTP 202 Accepted, яка вказує, що запит успішно прийнято і буде оброблено у фоновому режимі.

Далі подія надходить до LinkInstagramSaga, яка виконує роль координатора довготривалого бізнес-процесу [18, 28]. Використання саги є доцільним, оскільки сценарій підключення Instagram-акаунта складається з кількох незалежних кроків, що виконуються різними компонентами системи. Сага створює контекст процесу, зберігає його стан і публікує подію «Instagram Linking Started Event», яка сигналізує іншим компонентам про необхідність розпочати отримання даних з Instagram.

На подію «Instagram Linking Started Event» реагує обробник «Analytics Fetch Handler». Він звертається до Meta Graph API, передає отриманий раніше код авторизації, виконує його обмін на токен доступу та отримує необхідну інформацію про Instagram-профіль користувача [19]. До таких даних можуть належати ідентифікатор акаунта, базові дані профілю, а також показники, які надалі використовуються модулем аналітики та пошуку партнерів.

Після успішного отримання даних «Analytics Fetch Handler» публікує подію «Instagram Account Data Fetched Event». Ця подія повідомляє сагу про те, що дані Instagram-акаунта отримано успішно і процес можна переводити до фінального етапу. У відповідь сага формує команду «Finish Instagram Linking Command», яка передається обробнику завершення підключення. На цьому етапі інформація про підключений Instagram-акаунт зберігається в системі, зокрема в Keycloak, який використовується для управління користувацькими обліковими записами та пов’язаними з ними атрибутами [14].

Після успішного збереження даних сага публікує подію «Instagram Linking Completed Event». На основі цієї події система оновлює статус підключення та надсилає користувачу повідомлення через SignalR. Завдяки цьому користувач бачить у вебінтерфейсі фінальний результат операції без необхідності вручну оновлювати сторінку або повторно надсилати запит.

У випадку помилки на будь-якому з етапів, наприклад під час обміну коду авторизації на токен, звернення до Meta Graph API або збереження даних у Keycloak, сага переводить процес у стан помилки та публікує подію «Instagram

Linking Failed Event». Після цього користувач отримує відповідне повідомлення через SignalR, а в кеші або стані процесу фіксується невдалий результат операції.

Платформа використовує подієво-орієнтовану архітектуру для забезпечення слабкої зв’язності між модулями [12, 17]. Замість прямих викликів між сервісами система фіксує важливі зміни стану у вигляді доменних подій, які надалі передаються через RabbitMQ [27]. Послідовність обробки доменних подій між модулями системи зображено на діаграмі послідовності (рис. 2.7). Наприклад, після виконання бізнес-логіки в модулі Campaigns формується подія «Campaign Created Event», яка повідомляє інші частини системи про створення кампанії.

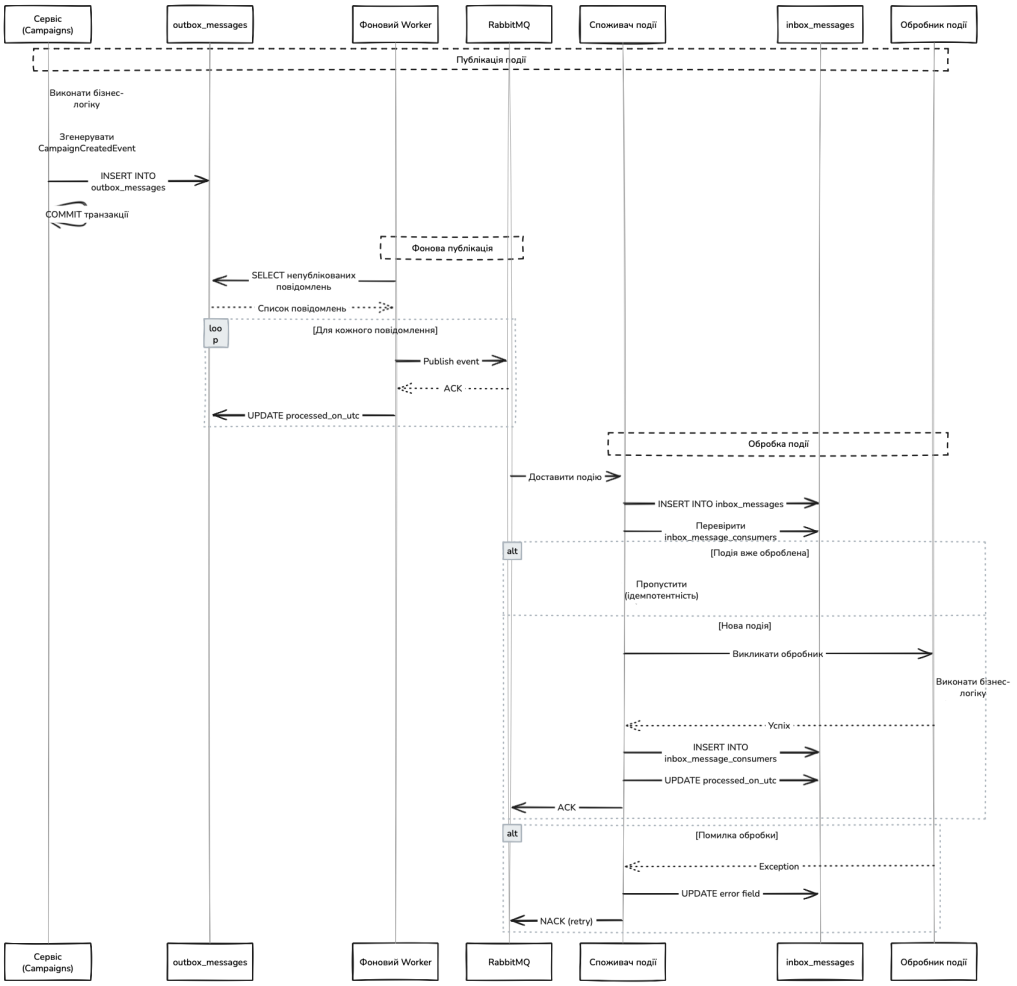


Рис. 2.7. Діаграма послідовності обробки доменних подій системи

Для гарантованої доставки повідомлень застосовується патерн Inbox/Outbox [29]. Перед публікацією подія зберігається в таблиці outbox_messages у межах тієї самої транзакції, що й основні зміни бізнес-

даних. Це дозволяє уникнути ситуації, коли зміни в базі даних уже зафіксовані, але повідомлення не було опубліковане через збій системи або недоступність брокера.

Публікацію подій виконує фоновий процес. Він періодично читає з outbox_messages неопубліковані повідомлення, надсилає їх до RabbitMQ і після отримання підтвердження позначає їх як оброблені, оновлюючи поле processed_on_utc. Якщо публікація не вдалася, повідомлення залишається необробленим і може бути відправлене повторно.

На стороні отримувача подія спочатку зберігається в таблиці inbox_messages, після чого передається відповідному обробнику. Перед виконанням бізнес-логіки система перевіряє, чи не було це повідомлення вже оброблене конкретним споживачем. Для цього використовується таблиця inbox_message_consumers. Якщо повідомлення вже оброблялося, воно пропускається, що запобігає дублюванню дій.

У разі успішної обробки система фіксує результат у inbox_message_consumers і позначає повідомлення як оброблене. Якщо виникає помилка, інформація про неї зберігається, а повідомлення може бути повторно доставлене для обробки. Таким чином, використання RabbitMQ разом із патернами Inbox та Outbox забезпечує надійний асинхронний обмін між модулями, захищає події від втрати та підтримує ідемпотентність операцій. Такий підхід дозволяє чітко розмежувати відповідальність між модулями: кожен модуль обробляє лише ті події, які стосуються його предметної області, не маючи прямих залежностей від інших частин системи. Це спрощує внесення змін до окремих модулів та зменшує ризик каскадних збоїв у разі недоступності одного з компонентів. Окрім того, наявність повної історії опублікованих та оброблених повідомлень у таблицях outbox_messages та inbox_messages дозволяє за потреби відтворити послідовність дій системи, що є корисним для діагностики проблем та аудиту бізнес-операцій.

Однією з ключових вимог до платформи є достовірність показників аудиторії блогерів, на основі яких рекламодавці приймають рішення про

співпрацю. Одноразове підключення облікового запису Instagram дозволяє отримати початкові дані, однак ці метрики змінюються з часом, зростає чи зменшується аудиторія, змінюється показник залученності. Тому для підтримання актуальності система виконує періодичну синхронізацію даних з Instagram.

Синхронізація реалізована як фонова задача, що виконується щодоби. В рамках кожного запуску система отримує дані всіх активних блогерів з підключеними Instagram-акаунтами через Instagram Graph API. Після отримання даних розраховуються агреговані показники аудиторії і залученності, а результати зберігаються як в MongoDB, так і в PostgreSQL [19]. Загальну послідовність кроків наведено на діаграмі активності (рис. 2.8).

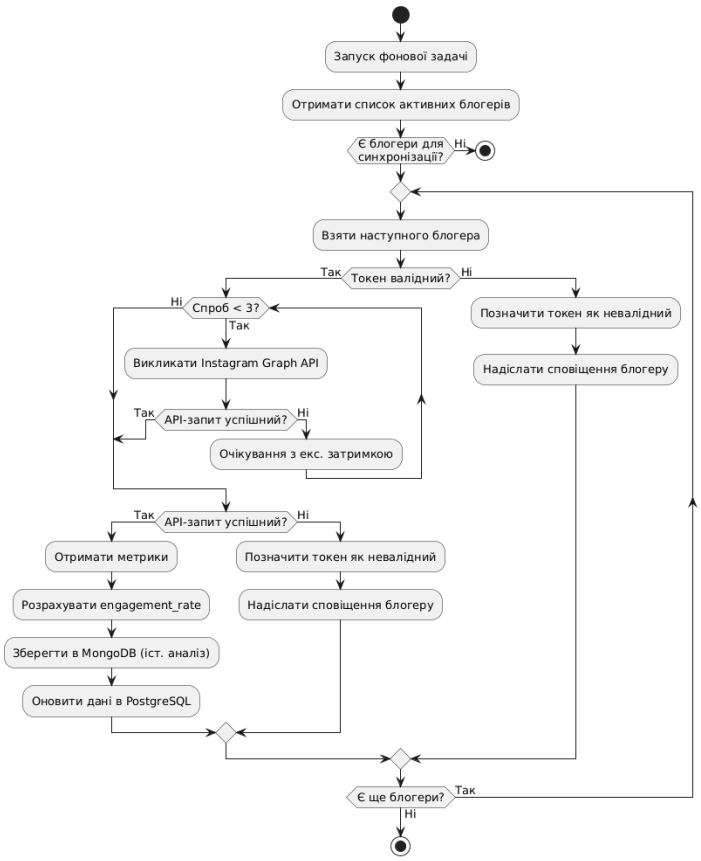


Рис. 2.8. Діаграма активності синхронізації аналітичних даних

Алгоритм синхронізації включає отримання списку активних блогерів з підключеними Instagram-акаунтами, перевірку валідності токенів доступу, виконання запитів до Instagram Graph API для отримання актуальних метрик (кількість підписників, середня кількість вподобань та коментарів), розрахунок

показника залученості аудиторії, збереження даних у MongoDB для історичного аналізу, оновлення агрегованих показників у PostgreSQL.

При помилках доступу до API система автоматично повторює запит з експоненційною затримкою. Після трьох невдалих спроб токен позначається як недійсний та надсилається сповіщення блогеру про необхідність повторної авторизації.

2.4 Реалізація функціоналу системи управління рекламниим кампаніями

Центральним елементом функціоналу системи управління рекламними кампаніями є процес створення та ініціалізації рекламної кампанії. Саме кампанія поєднує між собою основні сутності системи: користувачів у ролях замовника та виконавця, рекламну пропозицію (offer), умови співпраці (pact), а також подальші бізнес-процеси взаємодії між сторонами. Тому реалізацію основних принципів архітектури та прикладної логіки доцільно розглянути на прикладі створення рекламної кампанії зі статусом «Pending».

Реалізація сценарію створення кампанії охоплює всі основні рівні системи: транспортний рівень API, валідацію вхідних даних, прикладний рівень з обробкою команд, доменний рівень з бізнес-логікою, інфраструктурний рівень для роботи з базою даних та механізм публікації доменних подій. Така структура відповідає принципам Clean Architecture [16] та забезпечує розділення відповідальностей між компонентами системи.

Подальший опис побудовано за порядком проходження запиту крізь систему: від отримання HTTP-запиту до публікації подій про створену кампанію та оновлення пошукового індексу.

На рівні API взаємодія з клієнтським застосунком реалізована за допомогою Minimal API. Кожен ендпоінт оформлено як окремий клас, що успадковує базовий клас CampaignsEndpointBase, який реалізує спільну логіку для всіх ендпоінтів модуля кампаній: перевірку дозволів, обробку помилок та формування відповідей у стандартизованому форматі.

Відповідний HTTP-ендпоінт виконує прийом запиту, перевіряє права доступу користувача та передає керування до прикладного шару системи. Самі ендпоінти не містять бізнес-логіки. Їх функцією є трансляція HTTP-запиту в команду прикладного рівня та повернення результату клієнту.

```
internal sealed class PendCampaign : CampaignsEndpointBase
{
    protected override string[] RequiredPermissions =>
[Permissions.CreateCampaign];

    protected override RouteHandlerBuilder
MapEndpointInternal(IEndpointRouteBuilder app)
    {
        return app.MapPost(this.BuildRoute(string.Empty),
                async (
                    [FromBody] PendCampaignRequest request,
                    ISender sender,
                    CancellationToken cancellationToken) =>
                {
                    Result<Guid> result = await sender.Send(new
PendCampaignCommand(
                            request.Name,
                            request.Description,
                            request.ScheduledOnUtc,
                            new OfferId(request.OfferId)
                        ),
                        cancellationToken);

                    return result.Match(Results.Ok, ApiResult.Problem);
                })
            .WithTags(Tags.Campaigns)
            .WithName("PendCampaign")
            .WithSummary("Create pending campaign")
            .WithDescription("Creates a new campaign in pending status")
            .WithOpenApi();
    }

    internal sealed class PendCampaignRequest
    {
        public string Name { get; init; }
        public string Description { get; init; }
        public DateTimeOffset ScheduledOnUtc { get; init; }
        public Guid OfferId { get; init; }
    }
}
```

Перевірка дозволів (Permissions.CreateCampaign) обмежує доступ до функціоналу лише авторизованим користувачам з відповідними правами. Список дозволів формується на основі ролі користувача в системі: блогери отримують дозвіл на створення кампаній, тоді як інші ролі позбавлені цієї можливості.

Результат виконання команди передається через тип Result, який інкапсулює як успішне значення, так і інформацію про помилку. Використання типу Result замість винятків дозволяє явно моделювати потенційно невдалі

операції на рівні системи типів та зменшує ймовірність необробленої помилки під час виконання [1]. Метод Match перетворює Result у HTTP-відповідь: успішне значення повертається з кодом 200 OK, а помилка з відповідним кодом стану та описом проблеми відповідно до стандарту RFC 7807 [26].

Для реалізації бізнес-операцій у системі використовується підхід CQRS, відповідно до якого створення кампанії представлено у вигляді окремої команди [21]. CQRS передбачає розділення команд, що змінюють стан системи, та запитів, що лише читають дані. Команди представлені окремими класами-записами, які несуть всю необхідну інформацію та не мають жодної логіки. Передача команд між API та обробниками здійснюється через бібліотеку MediatR [17].

```
public sealed record PendCampaignCommand(
    string Name,
    string Description,
    DateTimeOffset ScheduledOnUtc,
    OfferId OfferId
) : ICommand<Guid>;
```

Команда інкапсулює всі необхідні дані для створення кампанії та не містить жодної логіки. Вона використовується як контракт між API-рівнем і прикладним шаром системи. Тип OfferId є строго типізованим ідентифікатором, який унеможливлює помилкове використання ідентифікатора однієї сутності замість іншої. Такий підхід застосовується в системах, побудованих за принципами Domain-Driven Design [3].

Виконання команди повертає Result із ідентифікатором створеної кампанії (Guid). У разі успіху клієнт отримує ідентифікатор для подальшої роботи з кампанією, а у разі помилки – структурований опис проблеми з кодом і повідомленням.

Перед виконанням прикладної логіки здійснюється перевірка коректності вхідних даних команди за допомогою механізму валідації. Валідація реалізована з використанням бібліотеки FluentValidation, яка надає декларативний синтаксис для опису правил перевірки. Кожен валідатор успадковує клас AbstractValidator та містить набір правил для конкретного типу команди.

```
internal sealed class PendCampaignCommandValidator :
AbstractValidator<PendCampaignCommand>
{
    public PendCampaignCommandValidator()
    {
        this.RuleFor(c => c.Name)
            .NotNull()
            .NotEmpty()
            .MinimumLength(5)
            .MaximumLength(Name.MaxLength);

        this
            .RuleFor(c => c.Description)
            .NotNull()
            .NotEmpty()
            .MaximumLength(Description.MaxLength);
    }
}
```

Валідація дозволяє запобігти обробці некоректних даних ще до звернення до бази даних та виконання доменної логіки. Правила перевірки включають перевірку на null, мінімальну та максимальну довжину текстових полів. Якщо валідація не пройдена, обробник команди не викликається, а клієнт отримує детальний опис кожного порушеного правила. Це дозволяє користувацькому інтерфейсу відобразити підказки біля невірно заповнених полів.

Валідація автоматично інтегрується в pipeline обробки команд через ValidationBehavior, який виконується перед обробником і перевіряє наявність зареєстрованих валідаторів для типу команди. Завдяки цьому додавання нової команди не вимагає явного виклику валідатора в обробнику.

Основна логіка створення рекламної кампанії зосереджена в обробнику команди. Саме тут відбувається перевірка бізнес-обмежень та ініціалізація доменного об'єкта кампанії. Обробник реалізує інтерфейс ICommandHandler і отримує необхідні залежності через конструктор. До таких залежностей належать репозиторії для роботи з агрегатами Campaign, Offer та User, а також контекст одиниці роботи, який забезпечує атомарність змін у базі даних.

Повний лістинг класу PendCampaignCommandHandler наведено в Додатку А (лістинг А.1).

Даний обробник отримує детальну інформацію про кампанію та публікує інтеграційну подію, яка може бути використана іншими модулями системи, наприклад для нотифікацій або аналітики. Обробник виконує перевірку існування рекламної пропозиції у базі даних, перевіряє, що користувач, який

створює кампанію, має на це право відносно обраної пропозиції, та що дата запуску кампанії знаходиться в майбутньому. У разі порушення будь-якої з умов обробник повертає типізовану помилку зі стандартизованим кодом, не звертаючись до бізнес-логіки доменного шару.

Якщо всі перевірки пройдені, обробник створює доменний об'єкт Campaign через статичний фабричний метод Pend, який інкапсулює доменну логіку створення кампанії у статусі pending. Фабричний метод публікує доменну подію «Campaign Pended Domain Event», яка фіксує факт створення кампанії в системі. Подія додається до колекції доменних подій агрегата і публікується після успішного збереження агрегата в базі даних.

Після створення доменного об'єкта обробник додає його до репозиторію та викликає метод SaveChanges одиниці роботи. Цей виклик ініціює транзакцію, у межах якої виконуються збереження агрегата в базі даних та запис доменних подій до таблиці outbox_messages. Таке об’єднання операцій в одну транзакцію є реалізацією патерна Transactional Outbox [22], який гарантує, що події не загубляться у разі збою системи.

Окремо реалізовано ORM-перехоплювач для синхронізації даних кампанії з пошуковим індексом. Перехоплювач реалізовано через механізм Entity Framework Core SaveChangesInterceptor, який дозволяє виконувати додаткову логіку при збереженні змін у базі даних [9]. У контексті кампаній перехоплювач фіксує зміни в агрегатах Campaign та формує події синхронізації для відправки до Elasticsearch [11], який використовується для повнотекстового пошуку кампаній за назвою, описом та іншими атрибутами.

Повний лістинг класу CampaignPendedDomainEventHandler наведено в Додатку А (лістинг А.2).

Повний лістинг класу CampaignsChangeCaptureInterceptor наведено в Додатку А (лістинг А.3).

Такий підхід дозволяє забезпечити актуальність пошукових даних без втручання в основну бізнес-логіку створення кампанії та відповідає принципам подієво-орієнтованої архітектури. Перехоплювач збирає зміни лише після

успішного завершення транзакції, тому до пошукового індексу потрапляють виключно ті дані, що фактично збережені в базі.

Подія CampaignPendedDomainEvent обробляється двома незалежними обробниками. CampaignPendedDomainEventHandler виконує операції в межах того ж модуля. CampaignPendedSearchSyncDomainEventHandler синхронізує дані кампанії з пошуковим індексом Elasticsearch, формуючи документ із полями назви, опису, статусу, ідентифікаторів учасників та дати створення.

Окрім доменних подій, обробник створення кампанії публікує інтеграційну подію «Campaign Pended Integration Event» через шину повідомлень RabbitMQ. Інтеграційні події призначені для модулів-споживачів за межами модуля Campaigns. Кожен споживач обробляє подію незалежно, що дозволяє модулям еволюціонувати окремо.

Для забезпечення ідемпотентності обробників вхідних подій застосовано шаблон Inbox. Кожне отримане повідомлення спершу записується до таблиці inbox_messages, а перед запуском обробника система перевіряє, чи не була ця подія вже опрацьована раніше. Такий підхід унеможливлює повторне виконання бізнес-логіки в разі повторного надходження тієї самої події, що особливо важливо в умовах асинхронного обміну повідомленнями, де доставлення «принаймні один раз» є типовою гарантією брокера [29].

Висновки до другого розділу

У другому розділі було виконано проєктування системи управління рекламними кампаніями в Instagram та обґрунтовано ключові архітектурні й структурні рішення. На основі аналізу предметної області визначено основні варіанти використання системи, що відображають взаємодію користувачів платформи у ролях замовників та виконавців рекламних кампаній, а також функціонал адміністративного управління.

Сформовано об’єктно-орієнтовану модель системи, яка охоплює ключові доменні сутності та їх взаємозв’язки. Побудована структура класів забезпечує цілісність предметної області та створює основу для подальшої реалізації бізнес-логіки.

Розроблено логічну структуру бази даних, що відповідає доменній моделі системи та забезпечує зберігання інформації про користувачів, рекламні пропозиції, кампанії та пов’язані з ними процеси. Запропонована схема бази даних дозволяє забезпечити узгодженість даних і підтримує подальше масштабування системи.

У розділі також розглянуто алгоритми роботи основних бізнес-процесів системи, зокрема процес створення та ініціалізації рекламної кампанії. Реалізація функціоналу продемонстрована на прикладі прикладної та доменної логіки, що підтверджує коректність обраних архітектурних підходів і технологічних рішень.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ УПРАВЛІННЯ РЕКЛАМНИМИ КАМПАНІЯМИ В INSTAGRAM

3.1 Порядок встановлення та налаштування параметрів платформи

Встановлення та налаштування параметрів системи передбачає підготовку серверного середовища, розгортання серверної та клієнтської частин, а також налаштування зв’язків між компонентами системи. Платформа Lanka реалізована як розподілена система з кількома взаємопов'язаними сервісами у контейнерах. До її складу входять серверна частина на ASP.NET Core, клієнтська частина на Angular, реляційна база даних PostgreSQL, документоорієнтована база даних MongoDB, кеш Redis, брокер повідомлень RabbitMQ, пошуковий рушій Elasticsearch та сервіс автентифікації Keycloak. Така багатокомпонентна архітектура вимагає узгодженого запуску всіх сервісів та коректного налаштування мережевих зв'язків між ними.

Для спрощення розгортання вся конфігурація серверного середовища описана у файлах проєкту та зберігається разом із вихідним кодом у системі контролю версій. Це дозволяє відтворити середовище розгортання у будь-який момент часу та забезпечує однакову поведінку системи в режимах розробки, тестування й експлуатації.

Розгортання платформи включає кілька послідовних кроків. На першому кроці на цільовому середовищі готується інфраструктура для запуску контейнерів. На другому кроці отримується вихідний код системи з репозиторію та налаштовуються параметри середовища відповідно до режиму експлуатації. На третьому кроці виконується збірка та запуск контейнерів через AppHost, який автоматично формує мережеві зв'язки між сервісами та налаштовує тома для збереження даних. Після запуску виконуються міграції баз даних, ініціалізується пошуковий індекс Elasticsearch та налаштовується сервіс автентифікації Keycloak.

Структура розгортання містить кілька логічних рівнів. Перший рівень становить клієнтська частина, яка обслуговується вебсервером і доступна користувачам через браузер. Другий рівень включає серверну частину, що

складається з API та шлюзу, які обробляють запити користувачів та координують роботу інших компонентів. Третій рівень охоплює інфраструктурні сервіси, які забезпечують зберігання даних та асинхронну взаємодію між модулями: бази даних PostgreSQL і MongoDB, кеш Redis, брокер повідомлень RabbitMQ та пошуковий рушій Elasticsearch. Окремо розташований сервіс автентифікації Keycloak, який керує обліковими записами та правами доступу користувачів.

Загальну структуру розгортання та зв'язки між компонентами системи зображено на діаграмі розгортання (рис. 3.1).

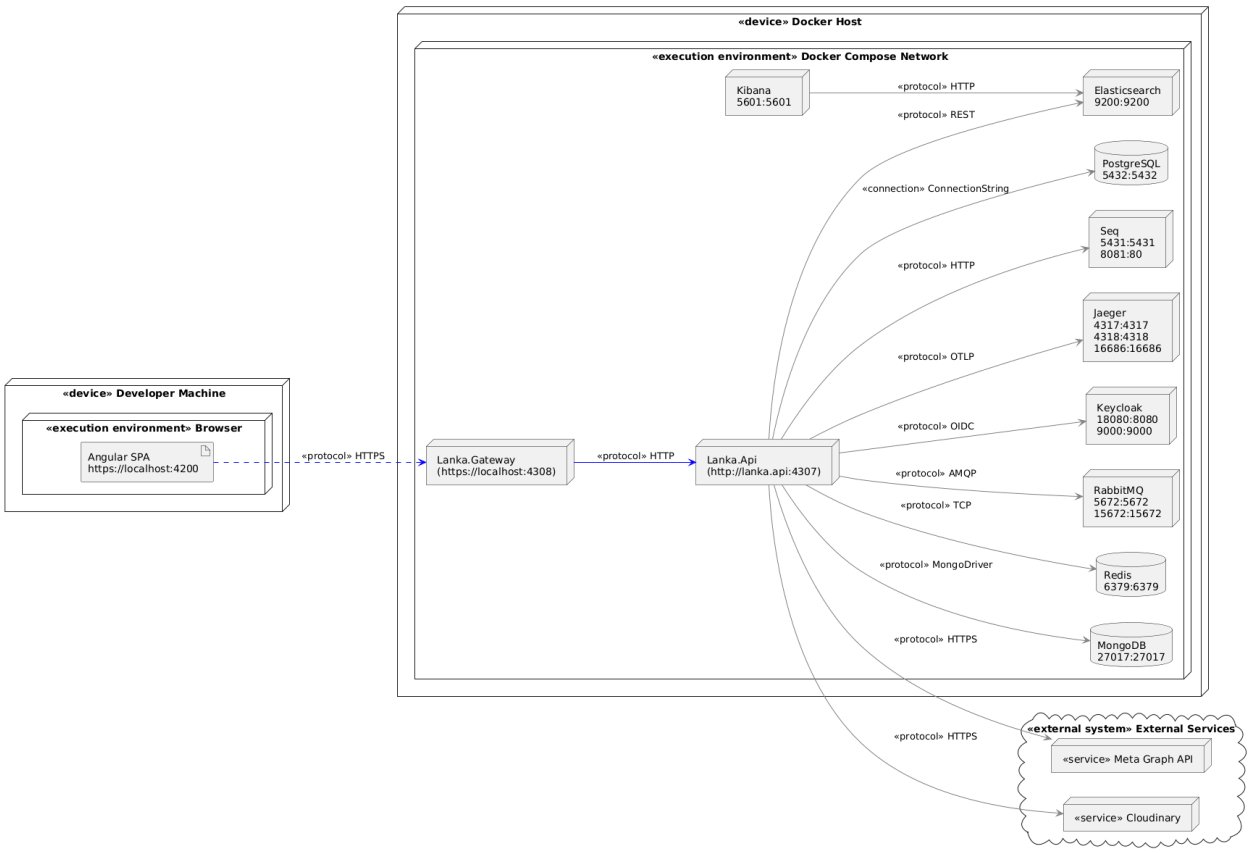


Рис. 3.1. Діаграма розгортання

Перед запуском платформи на цільовому середовищі мають бути встановлені засоби контейнеризації Docker та середовище виконання .NET. Docker забезпечує ізоляцію кожного сервісу платформи в окремому контейнері. Середовище виконання .NET необхідне для запуску серверної частини платформи та інструменту оркестрації Aspire.

Для роботи з системою контролю версій необхідний Git, через який отримуються оновлення вихідного коду. Усі необхідні інструменти доступні для основних операційних систем (Windows, Linux та macOS). Вихідний код системи

отримується з системи контролю версій, а запуск виконується з каталогу серверної частини проєкту, де розміщено конфігурацію для оркестрації сервісів. Структура репозиторію відокремлює серверну та клієнтську частини в різні папки. Файли конфігурації середовища (appsettings.json) розташовані в кореневому каталозі серверної частини та містять налаштування підключення до баз даних, інфраструктурних сервісів та зовнішніх API.

У режимі розробки використовується AppHost на основі Aspire, який забезпечує узгоджений запуск основних компонентів платформи: API, шлюзу, клієнтської частини, реляційної бази даних, MongoDB, Redis, RabbitMQ, Elasticsearch, Keycloak та допоміжних інструментів спостереження. Aspire є фреймворком оркестрації для .NET-застосунків, який дозволяє описати склад розподіленої системи у вигляді коду C#. AppHost-проєкт містить описи всіх компонентів платформи та зв'язків між ними, що автоматично перетворюються на конфігурацію розгортання.

Кожен компонент має своє призначення в архітектурі платформи. PostgreSQL [22] використовується як основна реляційна база даних для зберігання структурованої інформації про користувачів, рекламні пропозиції, кампанії, відгуки та фінансові операції. MongoDB [23] зберігає неструктуровані дані та архівні дані аналітики. Redis виконує роль розподіленого кешу для часто запитуваних даних та сесій користувачів, що зменшує навантаження на основні бази даних. RabbitMQ забезпечує асинхронний обмін повідомленнями між модулями системи та реалізує транспортний рівень для подій. Keycloak [20] виступає окремим сервісом автентифікації, який реалізує стандарти OAuth 2.0 і OpenID Connect та керує обліковими записами користувачів і ролями доступу.

Elasticsearch використовується для повнотекстового пошуку та фільтрації блогерів за низкою параметрів [5, 11]. Пошуковий індекс синхронізується з реляційною базою даних через механізм доменних подій: будь-яка зміна основних даних блогера або кампанії автоматично призводить до оновлення відповідного документа в Elasticsearch. Це дозволяє виконувати пошук за частковим співпадінням, нечіткими запитами та підтримує фасетну фільтрацію

за категоріями контенту, кількістю підписників, рівнем залученості й іншими атрибутами.

До параметрів належать рядки підключення до баз даних, облікові дані інфраструктурних сервісів, параметри автентифікації, ключі доступу до зовнішніх API та налаштування платіжної інтеграції. Параметри поділяються на дві категорії: конфігураційні (рядки підключення, URL-адреси сервісів, тайм-аути) та секретні (паролі, ключі API, токени доступу). Конфігураційні параметри зберігаються у файлах appsettings.json та відрізняються залежно від середовища розгортання, тоді як секретні параметри передаються через змінні середовища.

Параметри підключення до Meta Graph API містять ідентифікатор застосунку Facebook та секретний ключ, який використовується для обміну коду авторизації на токен доступу. Ці параметри отримуються при реєстрації застосунку в консолі розробника Meta. Для платіжної інтеграції необхідні API-ключі провайдера платежів.

У режимі розробки платформа запускається у конфігурації, призначеній для локальної розробки, демонстрування та тестування. Під час запуску автоматично застосовуються міграції бази даних та може виконуватися ідемпотентне наповнення системи тестовими даними. Міграції виконуються через інструмент Entity Framework Core Migrations [9], який порівнює поточну схему бази даних із цільовою та застосовує необхідні зміни. Кожна міграція описана у вигляді окремого файлу коду C#.

Тестові дані формуються за допомогою сидерів (data seeders), які створюють набори даних, придатних для демонстрації та тестування. Прикладами таких даних є тестові облікові записи блогерів і рекламодавців, набори рекламних пропозицій та сценарії кампаній на різних стадіях життєвого циклу.

Для спрощення перевірки функціональності у цьому режимі доступні допоміжні службові засоби: інтерактивна документація прикладного програмного інтерфейсу у форматі OpenAPI, засоби моніторингу, логи, трасування запитів та окремі службові ендпоінти для підготовки тестових

сценаріїв. Документація OpenAPI генерується автоматично на основі метаданих ендпоінтів і дозволяє виконувати тестові виклики API безпосередньо з браузера через інтерфейс Scalar.

Збірка системи виконується один раз засобами AppHost на основі Aspire, який формує образи контейнерів та конфігурацію оркестрації сервісів. Поведінка системи визначається під час запуску: застосунок перевіряє поточну конфігурацію оточення, і якщо це не режим розробки, допоміжні службові засоби, тестове наповнення даними та службові ендпоінти, призначені лише для розробки, автоматично вимикаються та не задіюються у Production середовищі.

Це досягається через перевірку змінної середовища ASPNETCORE_ENVIRONMENT, що автоматично визначається інфраструктурою .NET. У режимі Production інтерактивна документація OpenAPI та тестові ендпоінти стають недоступними, оскільки вони не призначені для використання кінцевими користувачами.

Для оновлення платформи до нової версії потрібно отримати актуальний вихідний код, оновити конфігураційні параметри та повторно виконати збірку засобами AppHost, після чого готові образи контейнерів розгортаються у цільовому середовищі.

3.2 Структура інтерфейсу системи управління рекламними кампаніями в Instagram

Клієнтська частина вебплатформи реалізована у вигляді односторінкового застосунку, що виконується у вебпереглядачі користувача. Доступ до функціональності системи надається лише авторизованим користувачам, тому взаємодія розпочинається зі сторінки реєстрації та входу. Нижче наведено основні екрани інтерфейсу системи з описом доступних користувачеві дій.

Взаємодія з платформою розпочинається з реєстрації облікового запису, оскільки доступ до функціональності системи мають лише авторизовані користувачі. Сторінку реєстрації наведено на рис. 3.2. Форма реєстрації містить поля для введення імені та прізвища користувача, електронної адреси, пароля та дати народження. Поле пароля підтримує прихований ввід, а для поля

електронної пошти виконується перевірка формату в режимі реального часу. У верхній частині сторінки розташовані перемикачі мови інтерфейсу (українська та англійська) та теми оформлення (світла/темна). Під формою міститься посилання на Політику конфіденційності, оскільки при реєстрації користувач підтверджує згоду на її умови.

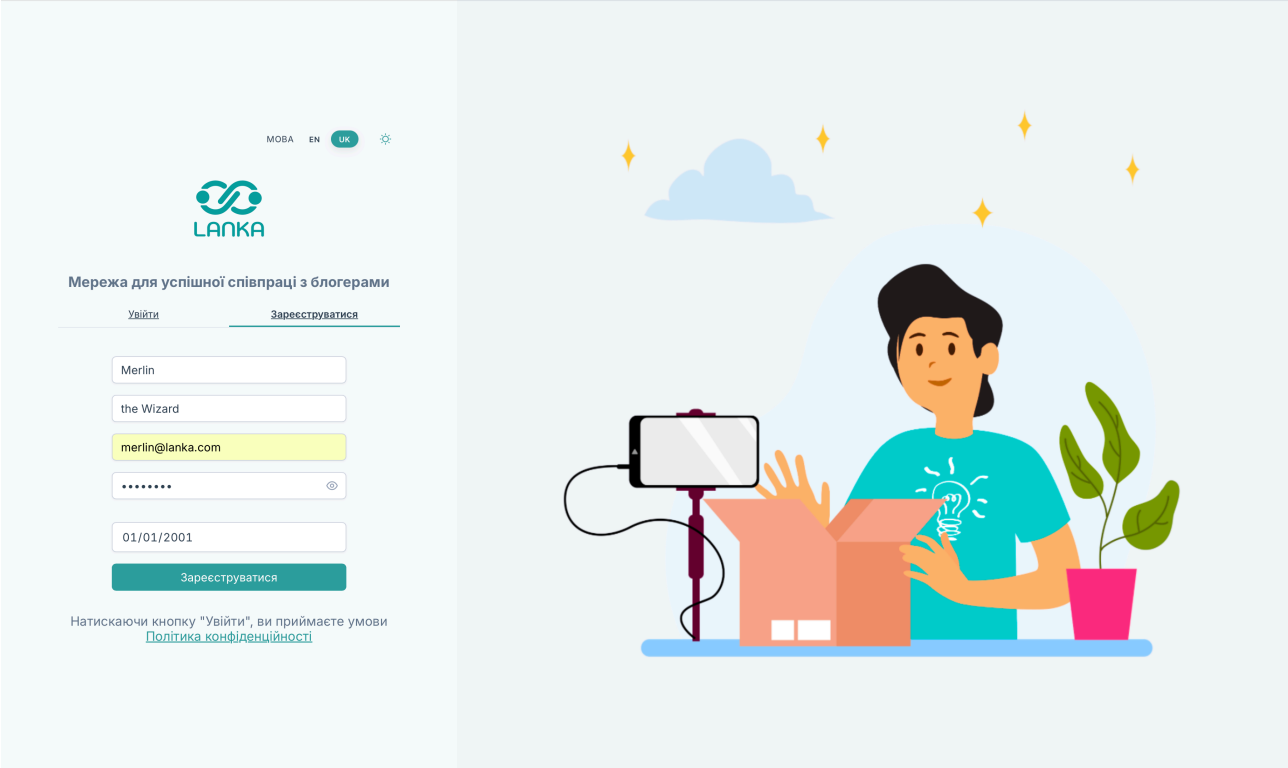


Рис. 3.2. Сторінка реєстрації облікового запису

Авторизація зареєстрованого користувача здійснюється на сторінці входу до облікового запису, наведеній на рис. 3.3. Для входу користувач вводить електронну адресу та пароль, вказані під час реєстрації. Перемикач у верхній частині форми дозволяє швидко перейти на сторінку реєстрації, якщо користувач ще не має облікового запису. У верхньому правому куті доступні перемикачі мови та теми оформлення, які зберігаються в профілі користувача після входу. Після успішної авторизації користувач автоматично перенаправляється на сторінку підключення Instagram-акаунта (якщо акаунт ще не підключено) або на основний екран платформи.

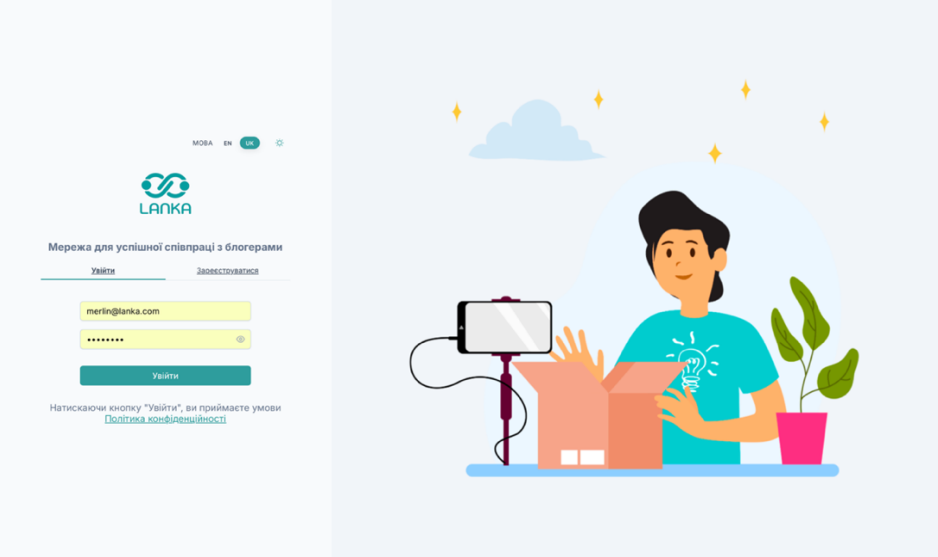


Рис. 3.3. Сторінка входу до облікового запису

Після реєстрації користувач має підключити обліковий запис Instagram, що є обов'язковим кроком: без такого підключення функції платформи недоступні, оскільки воно потрібне для отримання метрик та інших даних з Instagram. Сторінку підключення облікового запису наведено на рис. 3.4. Сторінка містить поточний статус підключення та кнопку «Підключити акаунт Instagram», яка запускає процес авторизації через Meta Graph API. Під формою приведено вимоги до акаунта: він має бути професійним (Бізнес або Creator), підключеним до сторінки Facebook із роллю Адміністратора, а під час авторизації потрібно надати дозволи на читання статистики. Дрібний надпис під кнопкою уточнює, що платформа отримує лише право читання показників професійного акаунта.

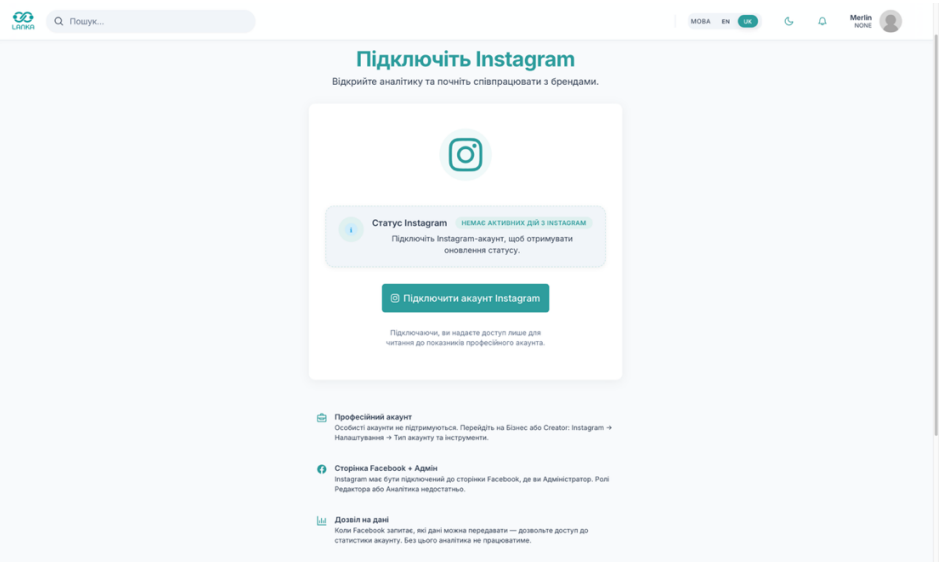


Рис. 3.4. Сторінка підключення облікового запису Instagram

Підключення виконується через обліковий запис Facebook Business, до якого прив’язано відповідний обліковий запис Instagram. Сторінку входу до Facebook Business наведено на рис. 3.5. Це офіційне вікно авторизації Meta, в якому користувач підтверджує вхід до свого акаунту. У верхній частині відображаються логотипи Meta та Lanka, що вказує на взаємодію між платформами. Після підтвердження користувач переходить до наступних етапів авторизації Meta Graph API.

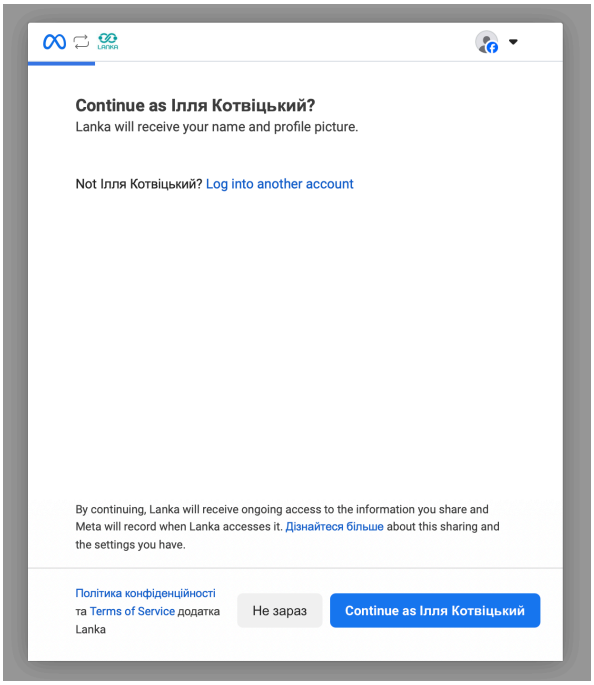


Рис. 3.5. Сторінка входу до облікового запису Facebook Business

Далі користувач обирає бізнес-сторінку, з якою пов’язано обліковий запис Instagram (рис. 3.6). Сторінка пропонує два варіанти надання доступу: до всіх поточних і майбутніх сторінок або лише до обраних користувачем. Під опціями міститься перелік бізнес-сторінок користувача з їхніми ідентифікаторами та іконками, кожну можна обрати поставивши відповідний прапорець. Рекомендується обирати варіант з обмеженим доступом, щоб зберегти контроль над тим, які саме ресурси Facebook будуть доступні платформі в майбутньому. Сторінка є частиною стандартного діалогу надання дозволів Meta.

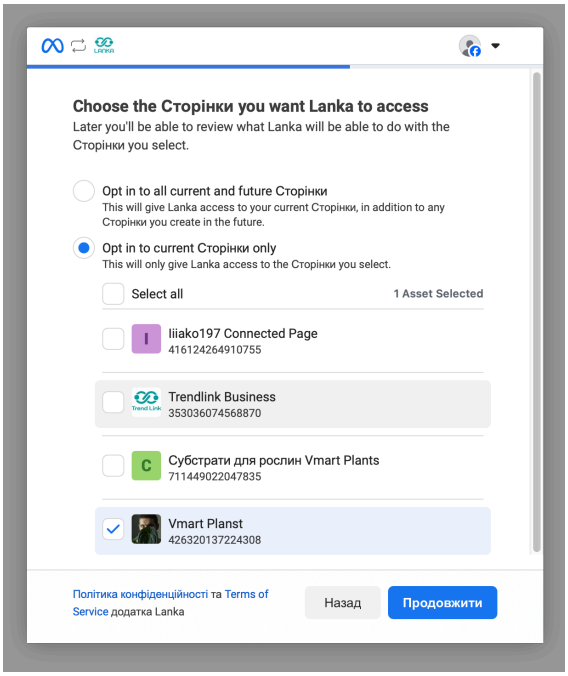


Рис. 3.6. Вибір бізнес-сторінки

Після цього здійснюється вибір відповідної компанії Facebook Business (рис. 3.7). Аналогічно до попереднього екрана, користувачу пропонується вибір між доступом до всіх компаній або лише до обраних. У списку відображаються всі компанії, до яких поточний користувач Facebook має доступ, кожна з власним ідентифікатором. Обирається саме та компанія, до якої прив’язаний бізнес-акаунт Instagram, що підключається. Після підтвердження процес переходить до сторінки вибору конкретного Instagram-акаунта.

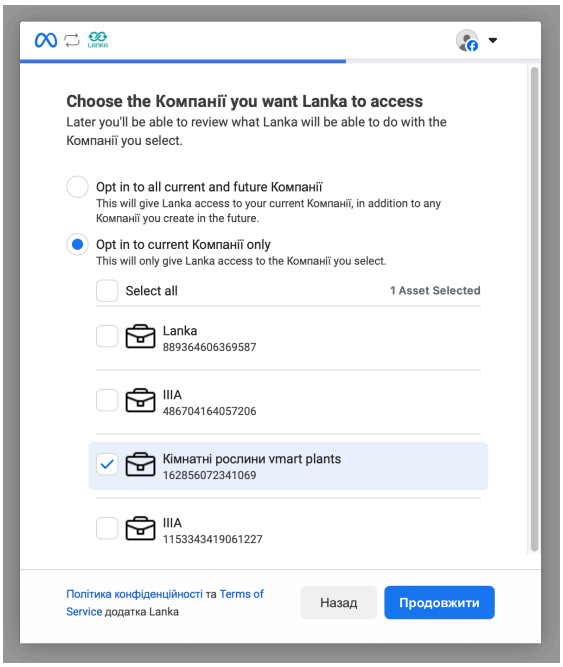


Рис. 3.7. Вибір компанії Facebook Business

Наступним кроком користувач обирає обліковий запис Instagram, який буде підключено до платформи (рис. 3.8). Відображається список доступних акаунтів, пов'язаних із вибраною бізнес-сторінкою Facebook. Після вибору потрібного акаунта система ініціює запит на отримання токена доступу та метаданих Instagram-профілю. Важливо, що підключення можливе лише для бізнес-акаунтів або акаунтів автора вмісту, оскільки лише вони надають доступ до аналітичних даних через Meta Graph API.

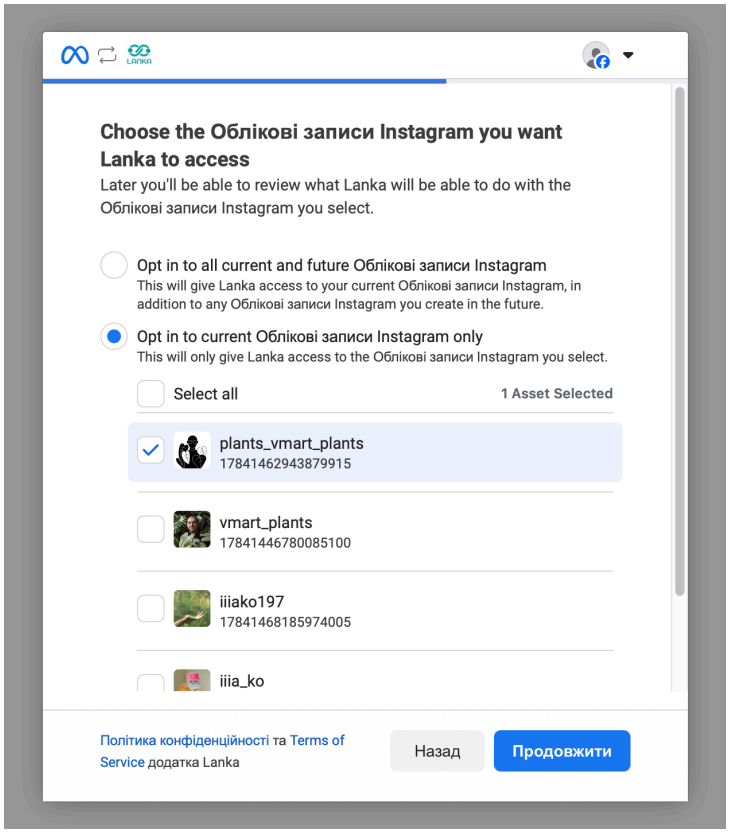


Рис. 3.8. Вибір облікового запису Instagram

Для отримання даних з облікового запису Instagram користувач надає згоду на відповідні дозволи (рис. 3.9). На цьому екрані відображається перелік прав доступу, які запитує платформа: читання публікацій, аналітики, даних аудиторії та ідентифікаційної інформації профілю. Надання дозволів є обов'язковим, оскільки без них система не може отримати метрики залученості, демографію та інші показники, необхідні для роботи модуля аналітики. Після підтвердження дозволів Meta повертає платформі код авторизації, який далі обмінюється на довгостроковий токен доступу.

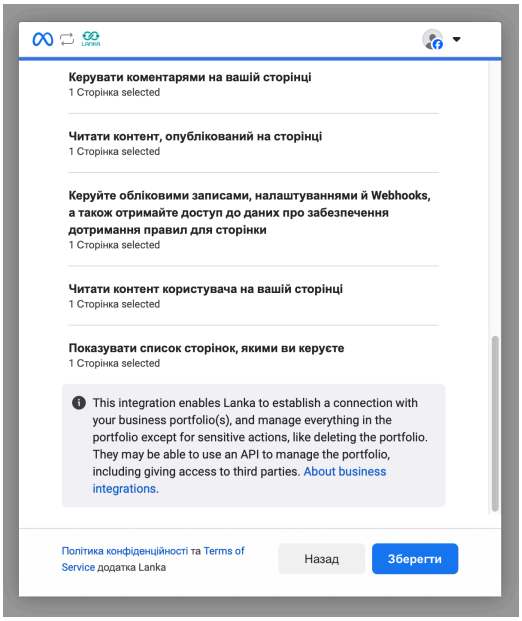


Рис. 3.9. Надання згоди на дозволи доступу до даних Instagram

Процес підключення облікового запису Instagram відображається в системі в реальному часі (рис. 3.10). Після отримання коду авторизації платформа виконує його обмін на токен доступу через Meta Graph API та паралельно отримує дані профілю, кількість підписників і показники аналітики. Оскільки ці операції виконуються асинхронно, інтерфейс відображає індикатор очікування та статус кожного кроку через механізм реального часу (SignalR). Це дає змогу користувачу бачити хід процесу без необхідності оновлювати сторінку вручну.

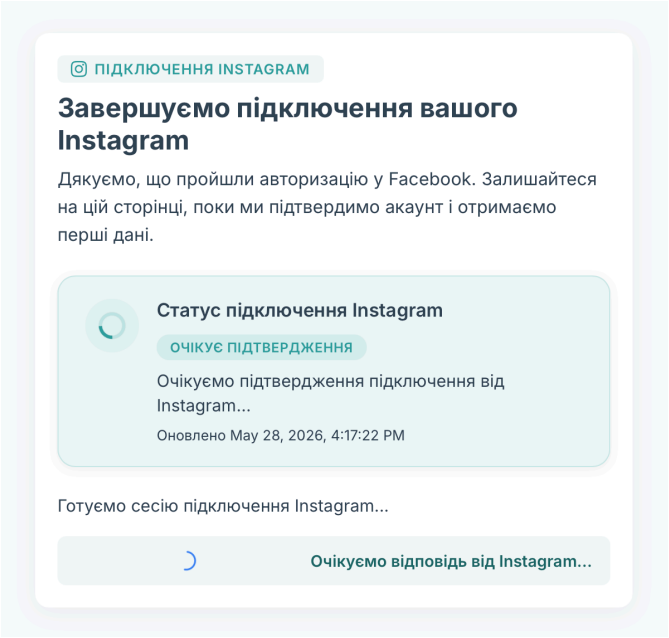


Рис. 3.10. Відображення процесу підключення облікового запису Instagram

У разі виникнення помилки під час підключення система відображає відповідне повідомлення, після чого процес можна повторити (рис. 3.11). Помилка може виникнути, зокрема, якщо термін дії коду авторизації вичерпано, Meta Graph API повернув відмову або виникла проблема з передачею даних. У такому разі сага підключення переходить у стан помилки та надсилає через SignalR відповідне сповіщення інтерфейсу. Користувач отримує опис причини помилки та кнопку для повторного запуску процесу без необхідності починати авторизацію заново.

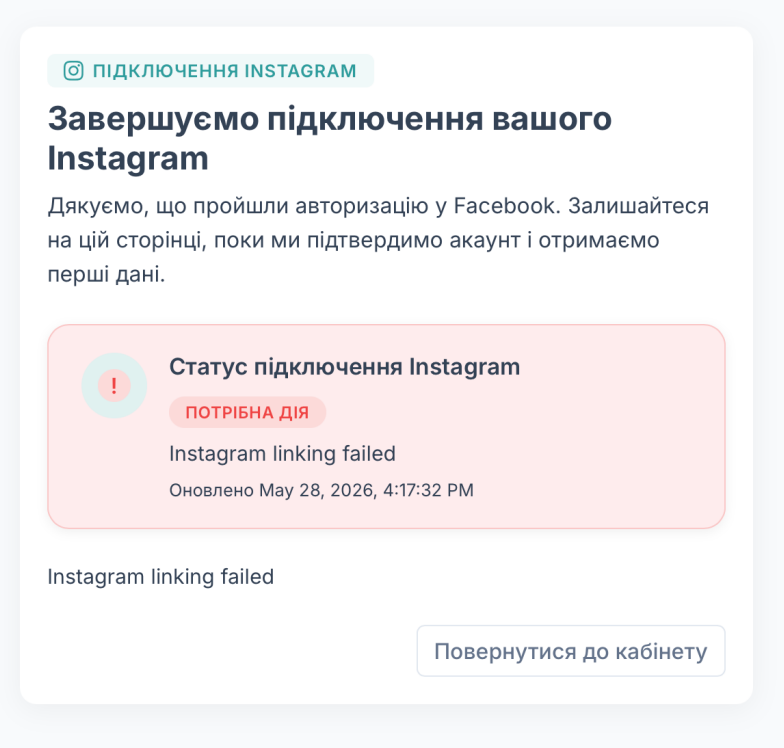


Рис. 3.11. Відображення помилки під час підключення облікового запису

Після успішного підключення облікового запису користувачеві стає доступною сторінка профілю (рис. 3.12). На ній відображається ім'я та фото профілю, категорія контенту, опис, а також агреговані показники підключеного Instagram-акаунта. Профіль є публічним у межах платформи: інші учасники можуть переглядати його перед тим, як надіслати заявку на рекламну кампанію. Таким чином, заповнений профіль безпосередньо впливає на кількість вхідних пропозицій, які отримує блогер.

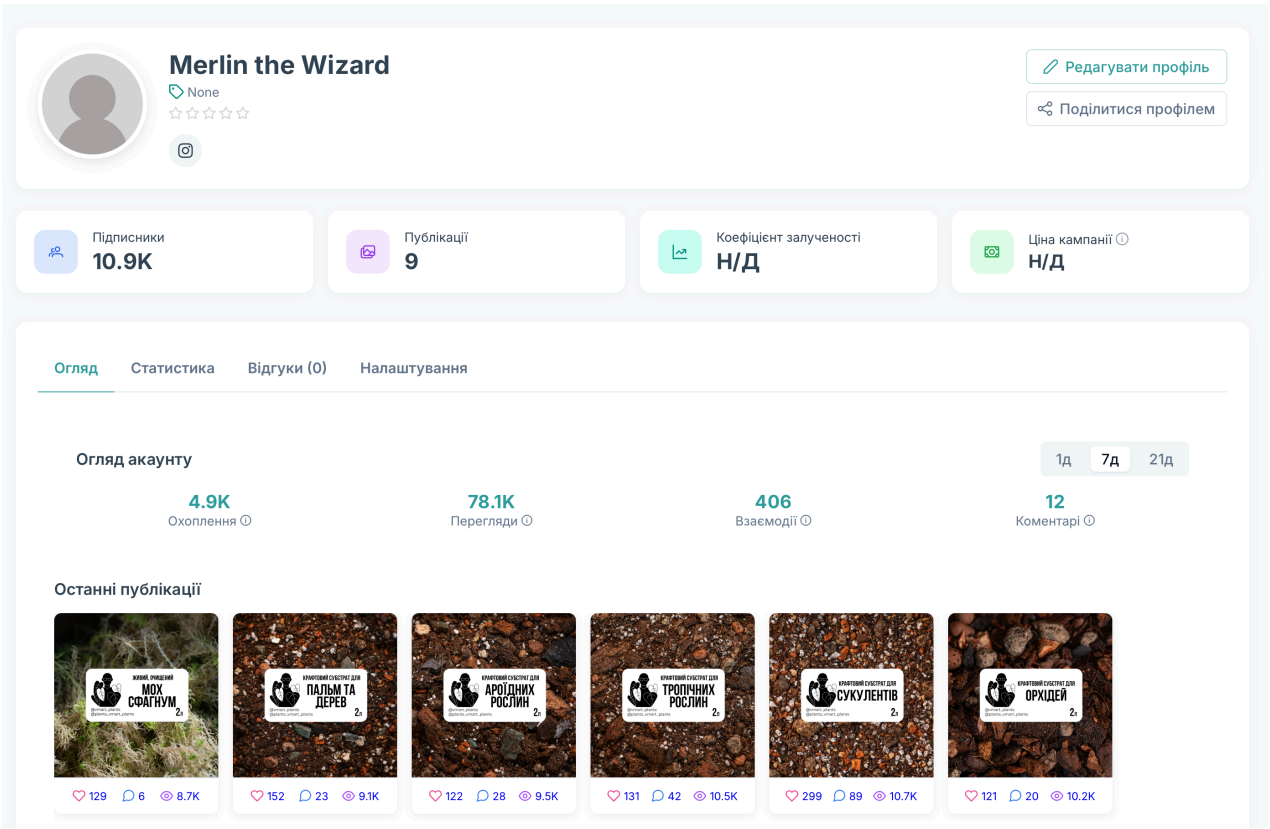


Рис. 3.12. Сторінка профілю користувача після підключення

На сторінці налаштувань користувач може змінити фото профілю, ім’я, категорію контенту та опис (рис. 3.13). Категорія контенту дозволяє відобразити спеціалізацію блогера (наприклад, спорт, краса, технології, подорожі) і використовується при пошуку та фільтрації в системі. Опис профілю бачать рекламодавці і він впливає на перше враження про блогера. Усі зміни зберігаються після підтвердження, після чого оновлений профіль відразу стає доступним для перегляду іншим користувачам.

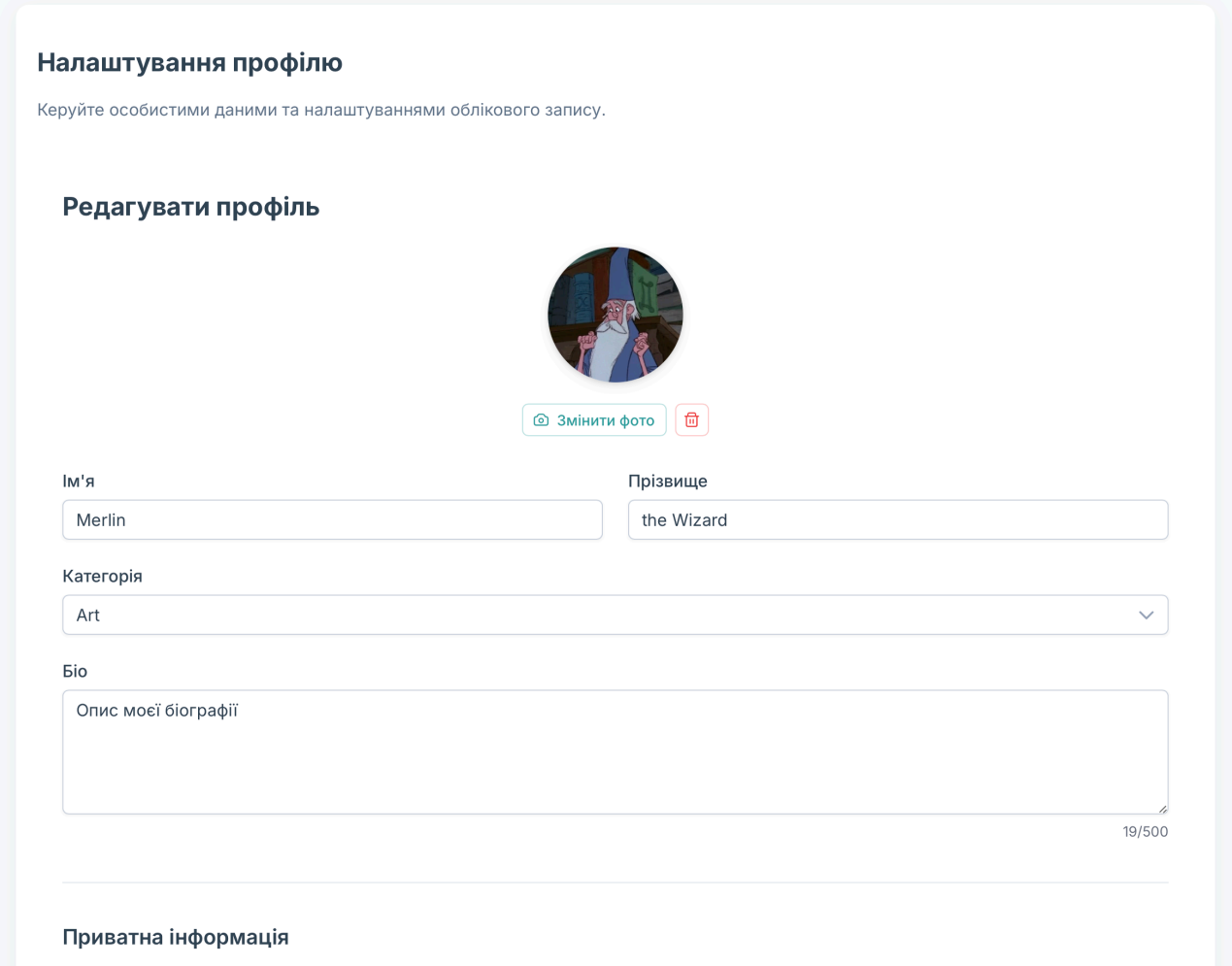


Рис. 3.13. Сторінка налаштувань профілю

У розділі налаштування оплат користувач зазначає номер IBAN та обирає валюту для отримання виплат за надані послуги, а також має можливість видалити обліковий запис (рис. 3.14). Ці налаштування необхідні для формування платіжних реквізитів у рамках рекламних кампаній. Вказаний IBAN зберігається в профілі користувача та автоматично підставляється при оформленні виплат. Видалення акаунта виконується через окрему кнопку в цьому ж розділі і є незворотною операцією: після підтвердження обліковий запис разом із усіма пов'язаними даними буде видалено з системи.

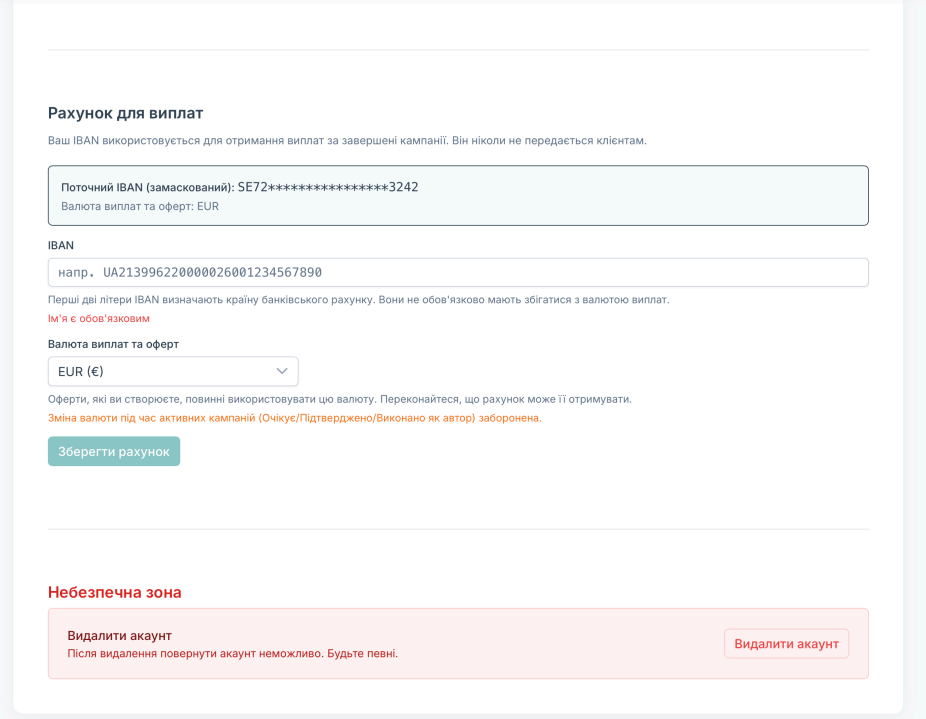


Рис. 3.14. Налаштування оплат

Сторінка пакту містить умови співпраці, які користувач визначає самостійно і які інші користувачі бачать перед початком рекламної кампанії. Доки пакт не створено, відображається пропозиція його сформувати, оскільки без пакту участь у рекламних кампаніях неможлива (рис. 3.15). Пакт визначає загальні правила роботи: терміни оплати, формат звітностей, додаткові вимоги до контенту. Ця інформація відображається в профілі блогера і впливає на рішення рекламодавця надіслати заявку. Пропозиції можуть бути створені лише після того, як пакт затверджено.

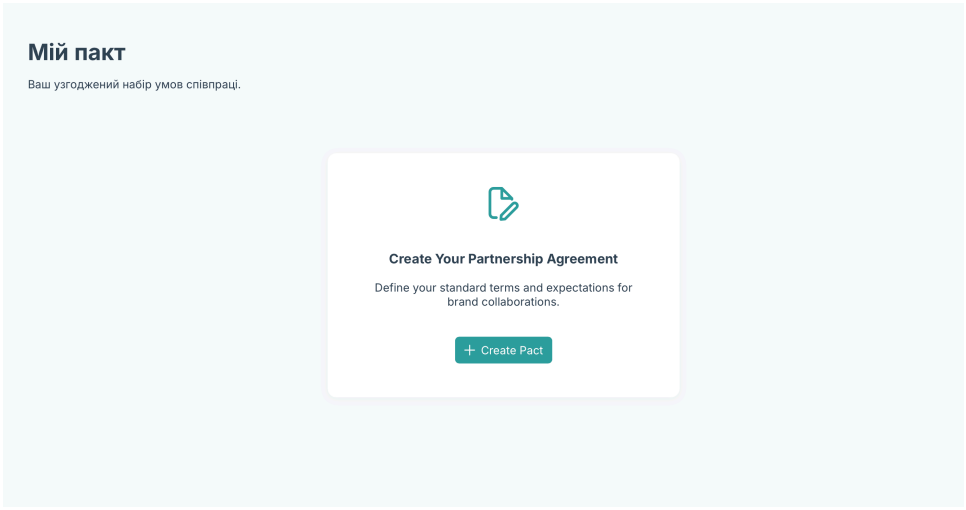


Рис. 3.15. Сторінка пакту без створених умов співпраці

Після натискання кнопки створення пакту відкривається редактор умов співпраці у форматі Markdown із можливістю попереднього перегляду та збереження змін (рис. 3.16). Використання Markdown дозволяє структурувати умови з заголовками, списками і виділенням тексту. Попередній перегляд відображає відрендерений результат прямо в редакторі, що спрощує перевірку кінцевого вигляду до збереження. Після збереження пакт відразу стає доступним для перегляду іншими користувачами.

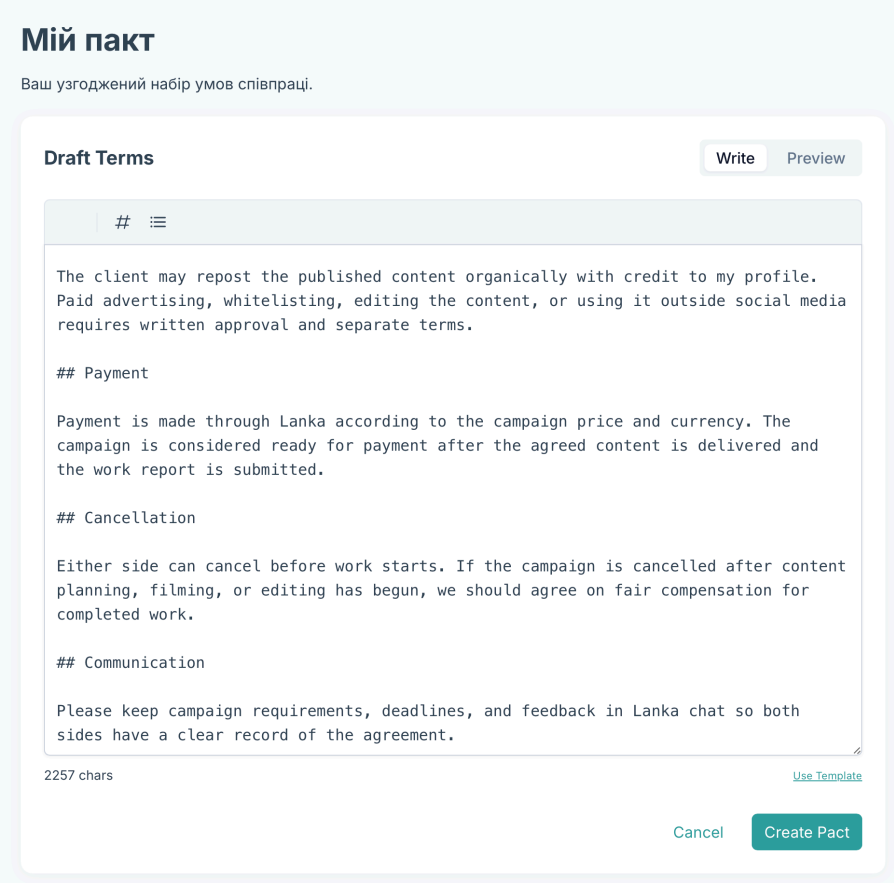


Рис. 3.16. Редагування пакту

Після створення пакту користувачеві стає доступним формування пропозицій, які він може надавати як виконавець рекламних кампаній (рис. 3.17). На сторінці відображується сам пакт з його текстом, а також перелік створених пропозицій з назвами, цінами і описами. Пакт є базовою офертою блогера і визначає загальні умови, в межах яких окремі пропозиції конкретизують деталі рекламної інтеграції. Рекламодавцям відображаються ці пропозиції перед тим, як надіслати заявку на кампанію.

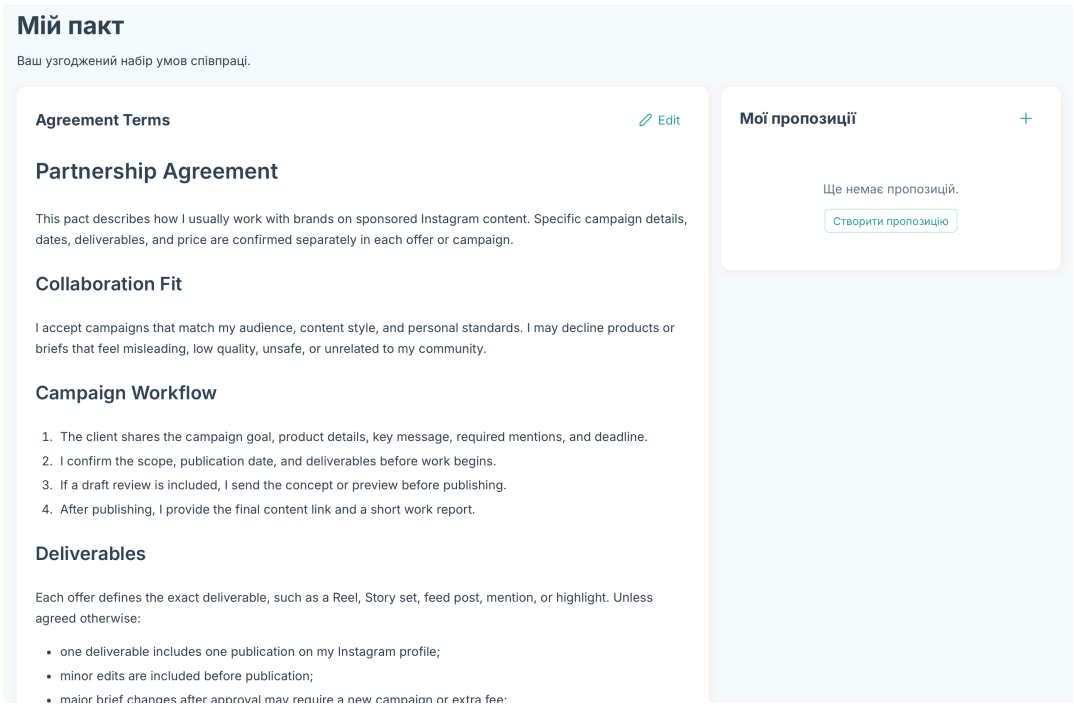


Рис. 3.17. Сторінка зі створеним пактом

У меню створення пропозиції користувач зазначає назву, ціну та опис того, як саме буде виконано рекламну інтеграцію (рис. 3.18). Назва пропозиції відображається рекламодавцям при перегляді профілю. Опис рекламної інтеграції дозволяє рекламодавцю зрозуміти, який саме формат інтеграції запропоновано і чи він відповідає його тематиці. Ціна встановлюється блогером самостійно і вказується в деталях кампанії.

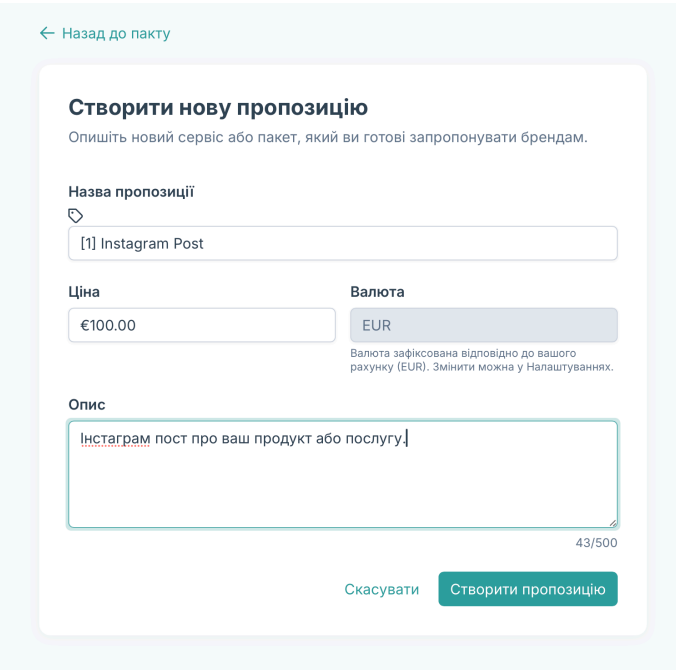


Рис. 3.18. Меню створення пропозиції

Профіль користувача містить кілька вкладок. На вкладці «Огляд» відображаються останні публікації в Instagram, загальна статистика за періоди в 1, 7 та 21 день, а також календар рекламних кампаній (рис. 3.19). Статистика включає кількість підписників, публікацій, охоплення та рівень залученості. Дані постійно оновлюються відповідно до розкладу синхронізації Instagram. Календар відображає поточні кампанії з позначками статусу, що дозволяє швидко оцінити завантаженість блогера для нових замовлень.

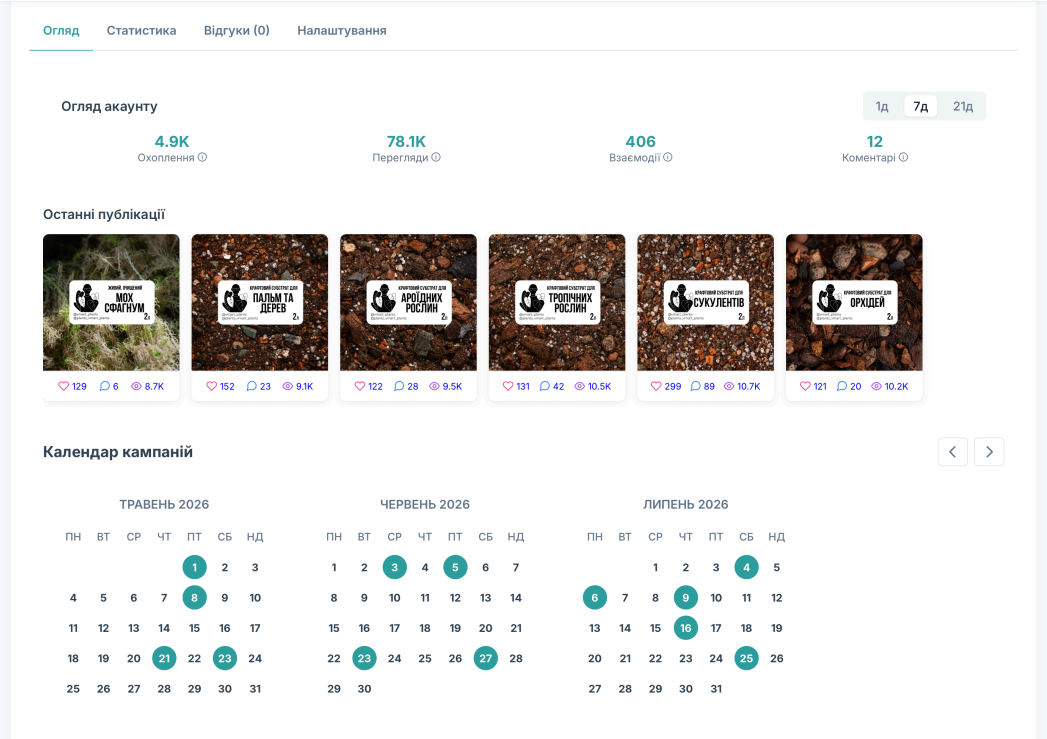


Рис. 3.19. Профіль користувача, вкладка «Огляд»

На вкладці «Статистика» доступний перегляд показників аудиторії за періоди в 1, 7 та 21 день, зокрема переважної статі, основної вікової групи та локації за країнами або містами. Географічний розподіл аудиторії наведено на рис. 3.20, а демографічний розподіл на рис. 3.21. Ці дані отримуються через Meta Graph API та зберігаються під час кожної синхронізації. Перемикання між часовими проміжками дозволяє рекламодавцям порівняти динаміку показників і виявити тренди. Наявність точних демографічних даних є ключовою перевагою платформи перед конкурентами, де ці дані або недоступні, або наводяться без перевірки автентичності.

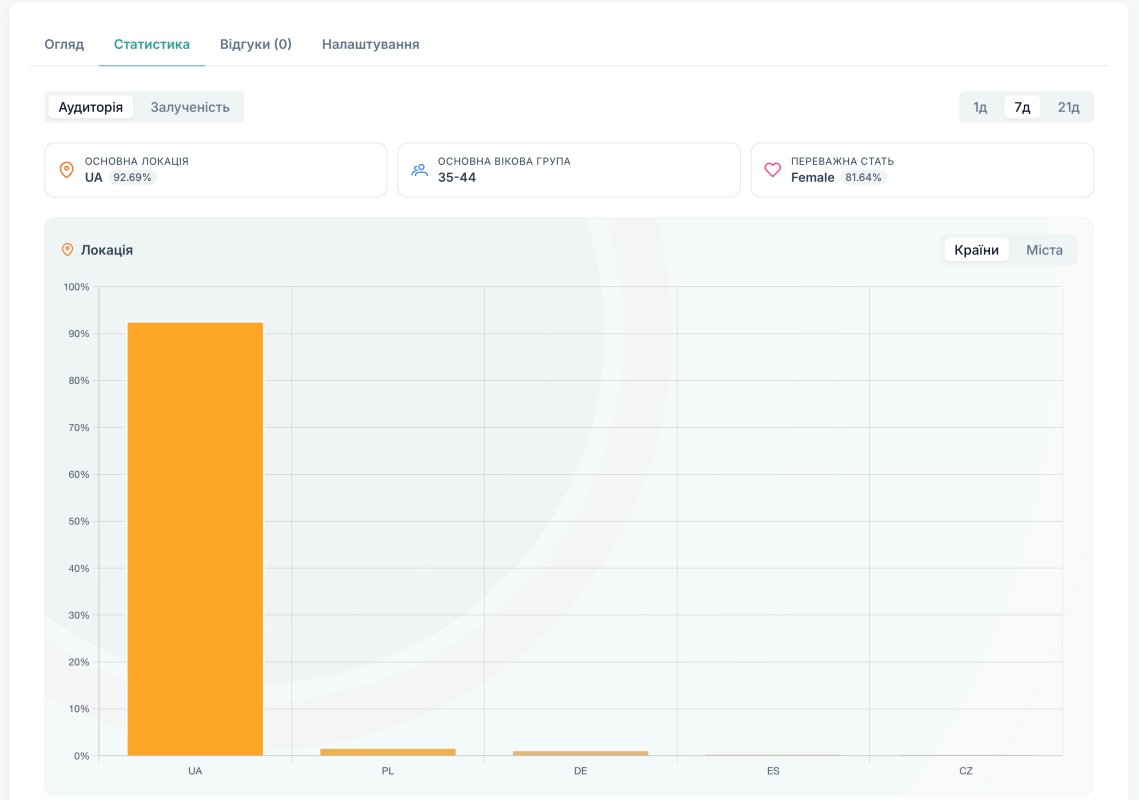


Рис. 3.20. Профіль користувача, вкладка «Статистика»: демографія аудиторії

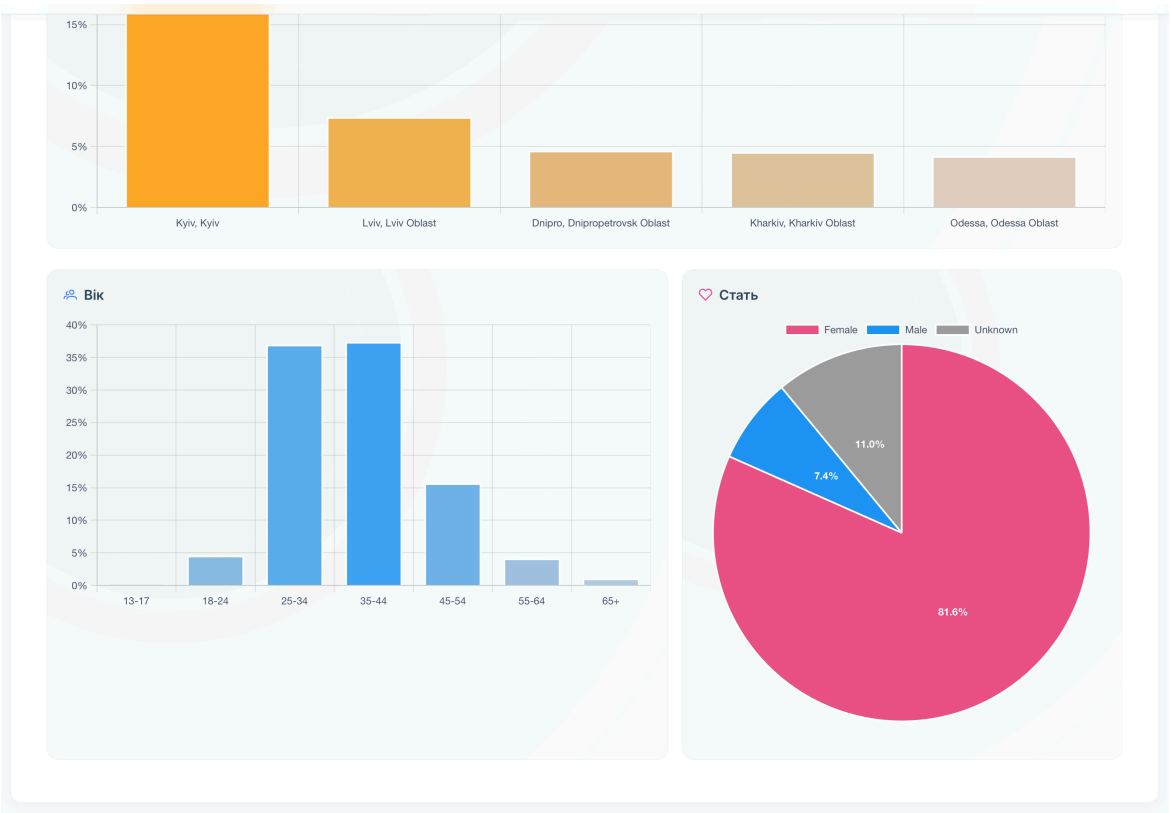


Рис. 3.21. Профіль користувача, вкладка «Статистика»: демографія аудиторії

Окремо на вкладці «Статистика» подано показники залученості, ефективності та взаємодії з аудиторією. Для кожного показника передбачено підказку, що відображає його значення та оцінку стану за шкалою «хороший», «середній» або «низький» (рис. 3.22). Така інтерпретація полегшує аналіз для рекламодавців, які можуть не розумітися на числових порогах кожного конкретного показника. Рекламодавець одразу бачить, чи відповідає інфлюенсер очікуванням щодо якості аудиторії та активності, ще до прийняття рішення про кампанію.

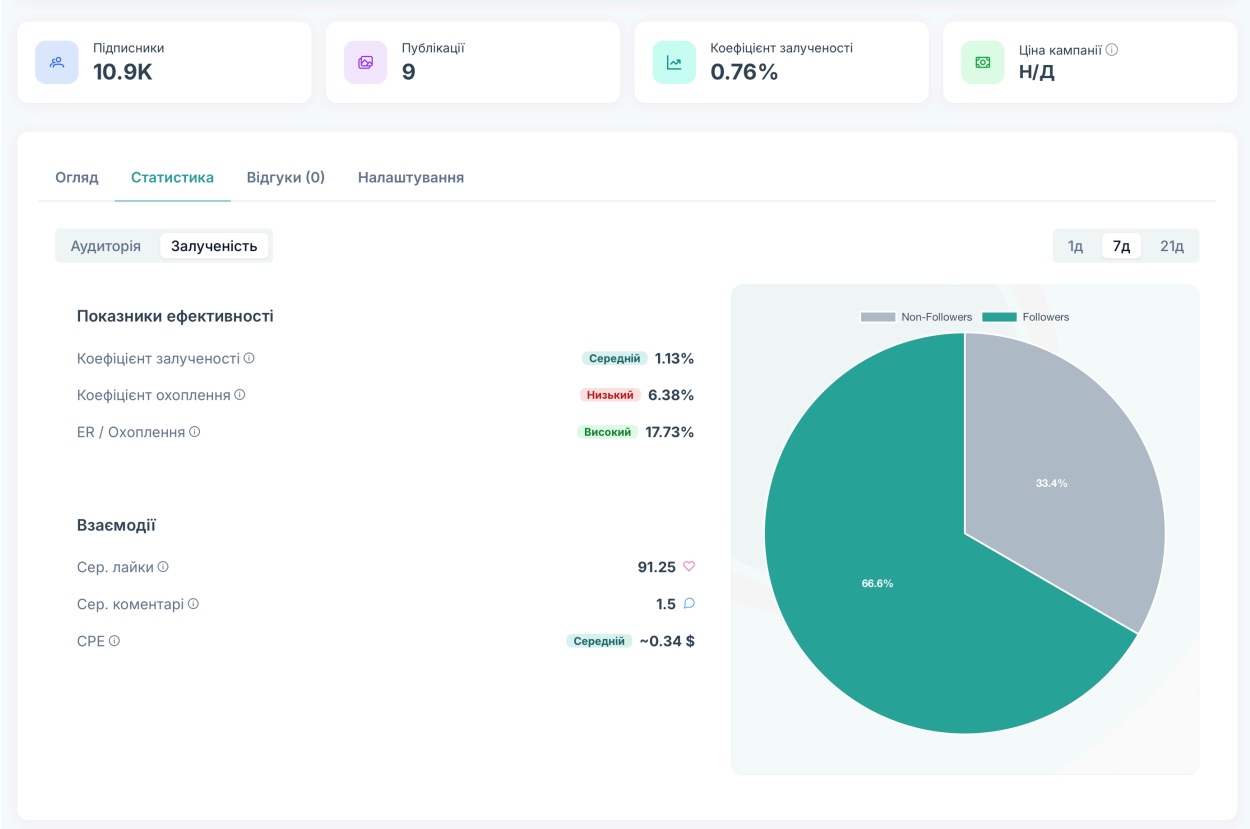


Рис. 3.22. Профіль користувача, вкладка «Статистика»: показники залученості та ефективності

На вкладці «Відгуки» користувач може переглянути відгуки, залишені замовниками за результатами рекламних кампаній (рис. 3.23). Кожен відгук містить оцінку, коментар від замовника та посилання на відповідну кампанію. Ці відгуки доступні іншим рекламодавцям при перегляді профілю блогера і слугують важливим критерієм при виборі партнера. Відгуки фіксуються в системі після завершення кампанії та є незмінними, що забезпечує їх достовірність.

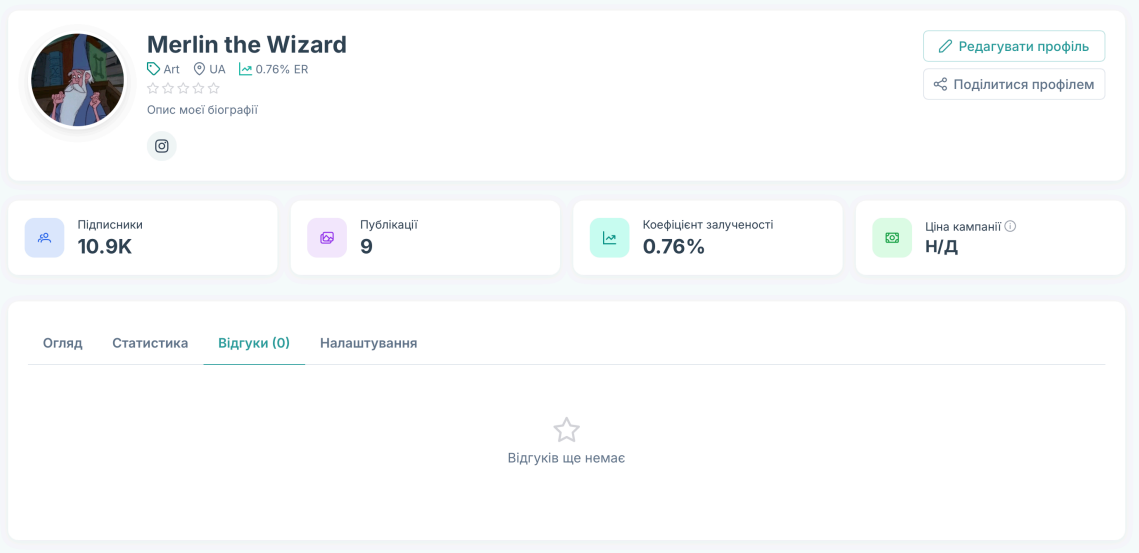


Рис. 3.23. Профіль користувача, вкладка «Відгуки»

На вкладці швидких налаштувань відображаються опис профілю та коротке зведення показників аудиторії, а також передбачено повторне оновлення підключення облікового запису Instagram на випадок, якщо сервіс перестає отримувати дані (рис. 3.24). Це може статися, якщо термін дії токена доступу вичерпався або інтеграцію було відкликано на стороні Meta. Кнопка оновлення запускає процес повторного підключення без необхідності виходити з платформи, що зручно при технічних збоях із синхронізацією даних.

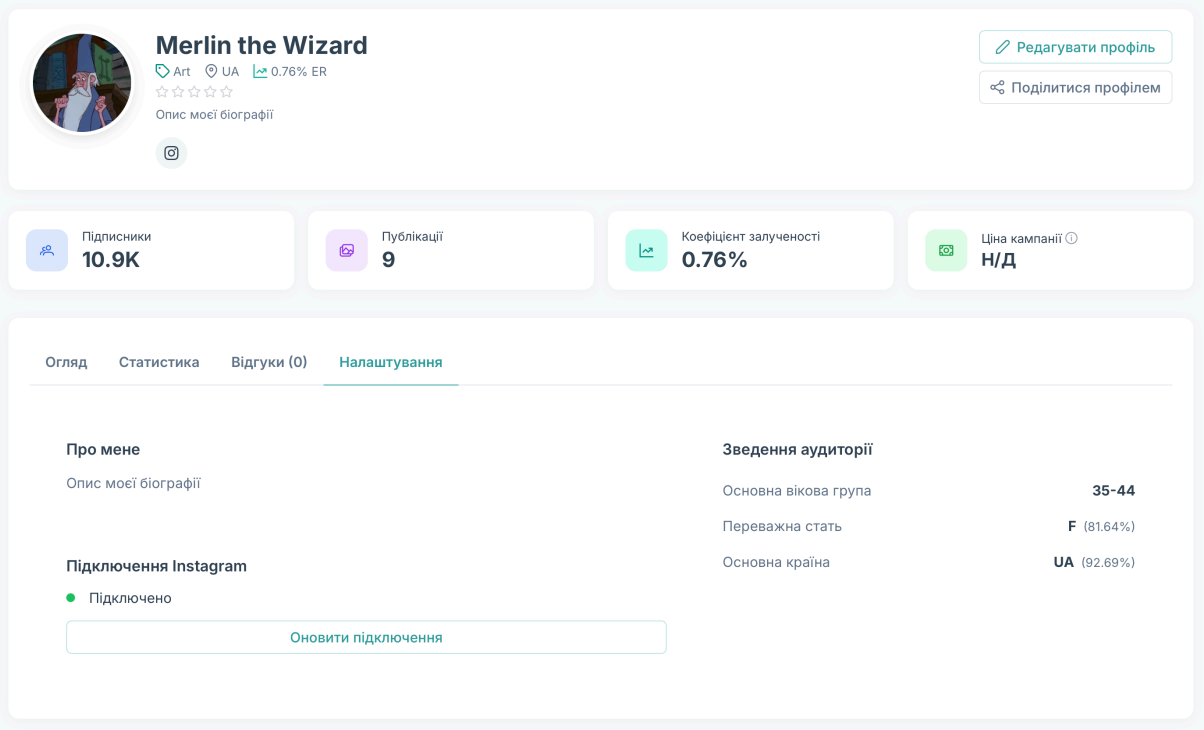


Рис. 3.24. Профіль користувача, вкладка швидких налаштувань

Система передбачає відображення сповіщень про події, пов’язані з рекламними кампаніями та взаємодією між користувачами (рис. 3.25). Сповіщення надходять у реальному часі через SignalR та відображаються у панелі у верхньому правому куті екрана з коротким описом події. Натиснувши на сповіщення, користувач переходить безпосередньо до відповідної сторінки: деталей кампанії, чату або профілю. Серед типів подій передбачено нову заявку, зміну статусу кампанії, нове повідомлення в чаті або новий відгук.

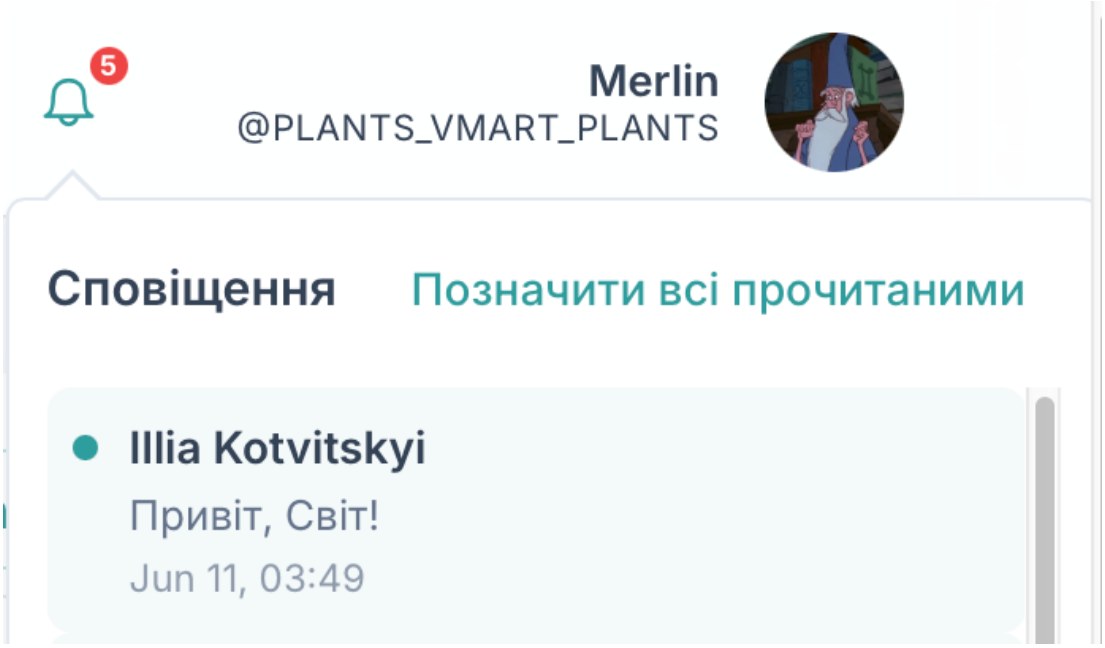


Рис. 3.25. Відображення сповіщень у системі

На сторінці пошуку блогерів користувач може ввести пошуковий запит та застосувати фільтри за категорією контенту, кількістю підписників, рівнем залученості аудиторії тощо (рис. 3.26). Пошук реалізовано через Elasticsearch, що забезпечує швидке повнотекстове знаходження блогерів за ім'ям, описом та категорією. Фільтри дозволяють звузити результати до аудиторії конкретного розміру або якісних показників, що особливо важливо при пошуку мікроінфлюенсерів із вузькою тематикою. Результати відображаються у вигляді карток із ключовими показниками кожного блогера.

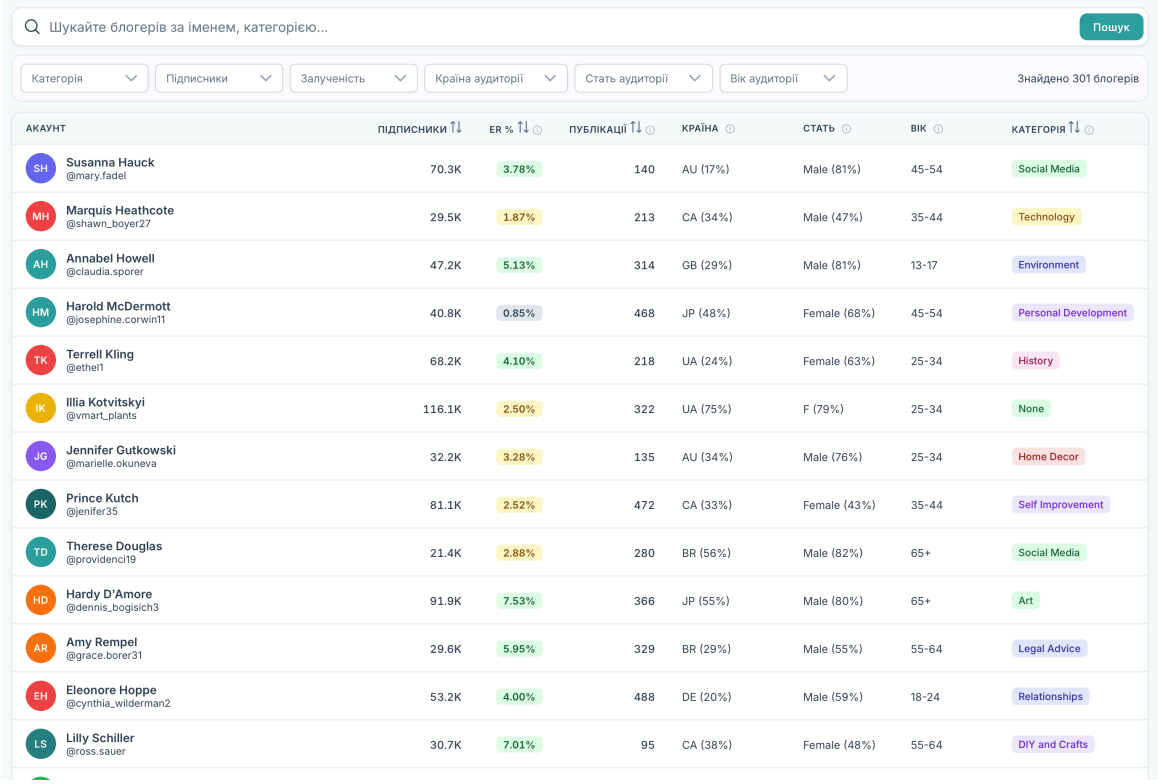


Рис. 3.26. Сторінка пошуку блогерів

За результатами пошуку система також пропонує схожі облікові записи блогерів (рис. 3.27). Рекомендації базуються на категорії контенту та показниках аудиторії, схожих до вже переглянутих профілів. Це допомагає рекламодавцю відкрити альтернативи, яких він міг не знайти через прямий пошук, та розширити список потенційних партнерів для кампанії.

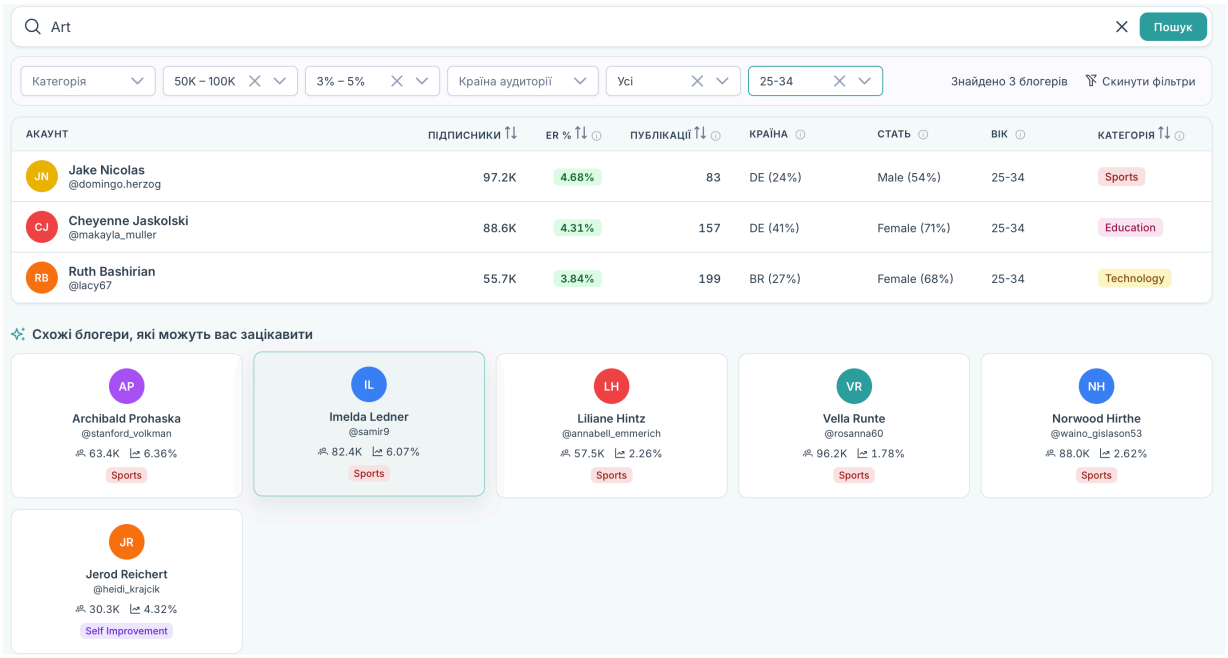


Рис. 3.27. Пропозиція схожих результатів пошук

Сторінка профілю іншого користувача надає аналогічні можливості перегляду: статистику охоплень, демографію аудиторії та анонімізовані відгуки. Загальний вигляд профілю наведено на рис. 3.28, статистику залученості аудиторії на рис. 3.29, а відгуки на рис. 3.30. При перегляді чужого профілю рекламодавець бачить ті самі дані, що й на власній сторінці блогера, але без можливості їх редагувати. Це забезпечує прозорість при прийнятті рішення щодо запуску кампанії, оскільки всі аналітичні дані отримані з офіційного API Instagram.

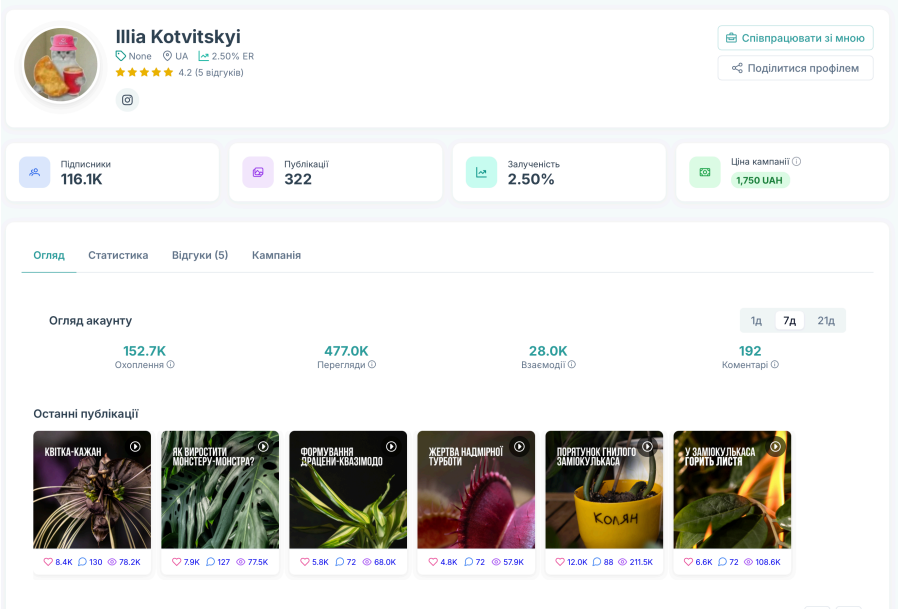


Рис. 3.28. Профіль іншого користувача: загальний вигляд

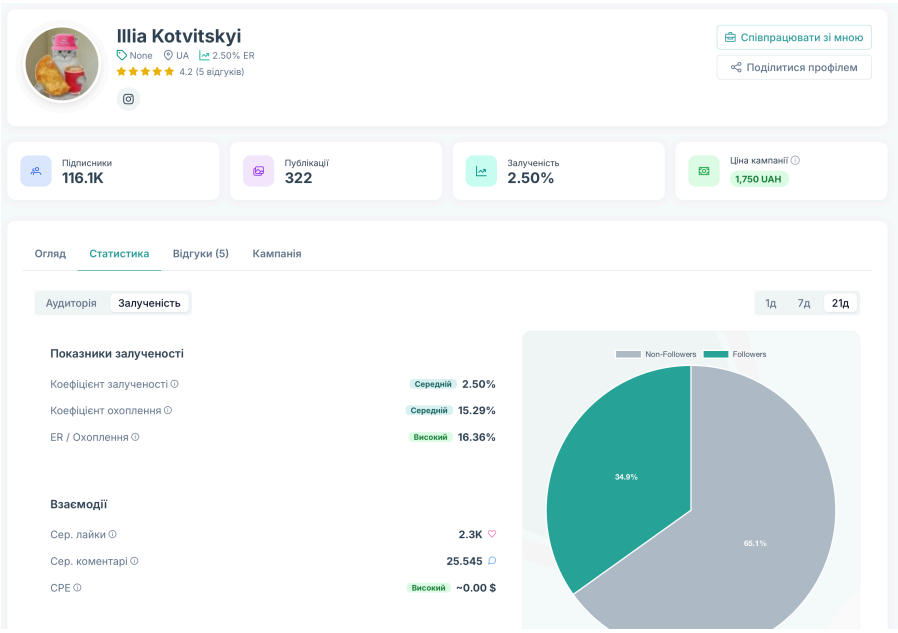


Рис. 3.29. Профіль іншого користувача: статистика залученості аудиторії

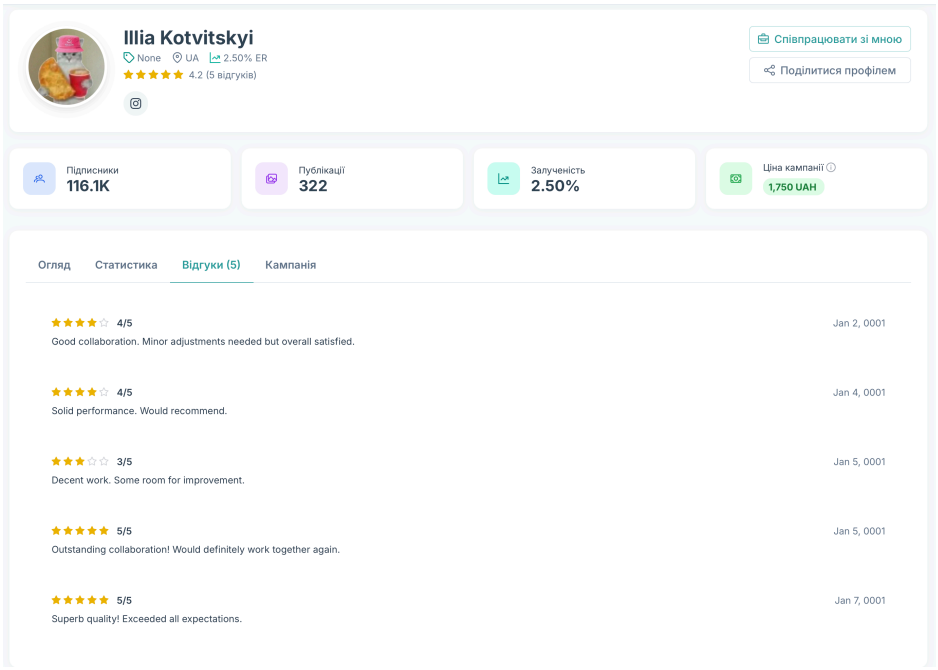


Рис. 3.30. Профіль іншого користувача: відгуки

На вкладці рекламних кампаній у профілі іншого користувача можна ознайомитися з його пактом та переглянути перелік пропозицій, які він надає. Передбачено листування з блогером щодо обраної пропозиції або початок рекламної кампанії за нею (рис. 3.31). Для кожної пропозиції відображаються назва, опис і ціна. Рекламодавець може обрати одну з них і одразу розпочати переговори через вбудований чат або одразу ініціювати кампанію. Така структура скорочує час від знайомства з блогером до укладання угоди про рекламну інтеграцію.

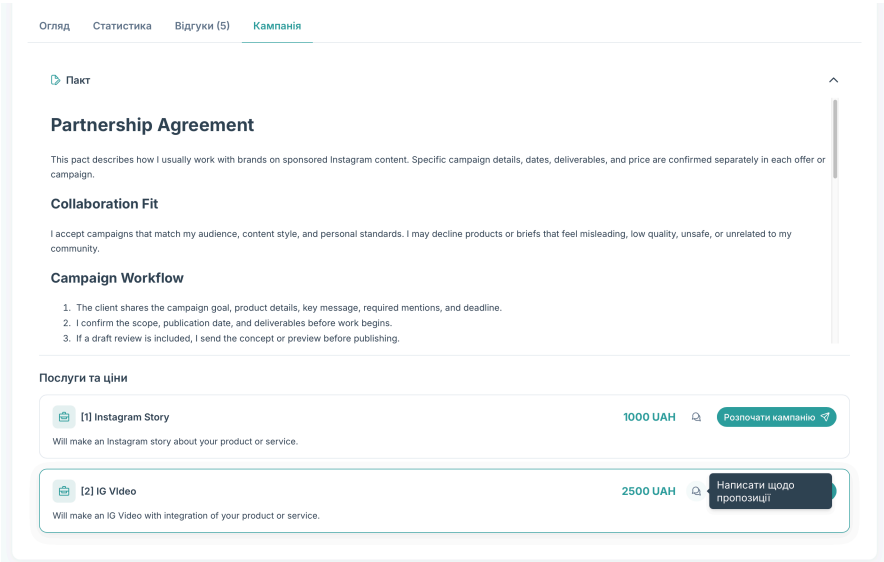


Рис. 3.31. Профіль іншого користувача, вкладка рекламних кампаній

На сторінці чатів користувач може переглядати історію листування та спілкуватися з іншими користувачами (рис. 3.32). Кожна гілка листування прив'язана або до конкретної пропозиції, або до рекламної кампанії, що дозволяє зберігати контекст перемовин. Учасники можуть прикріплювати файли, наприклад, технічне завдання або медіаматеріали. Нові повідомлення відображаються в реальному часі без необхідності оновлювати сторінку.

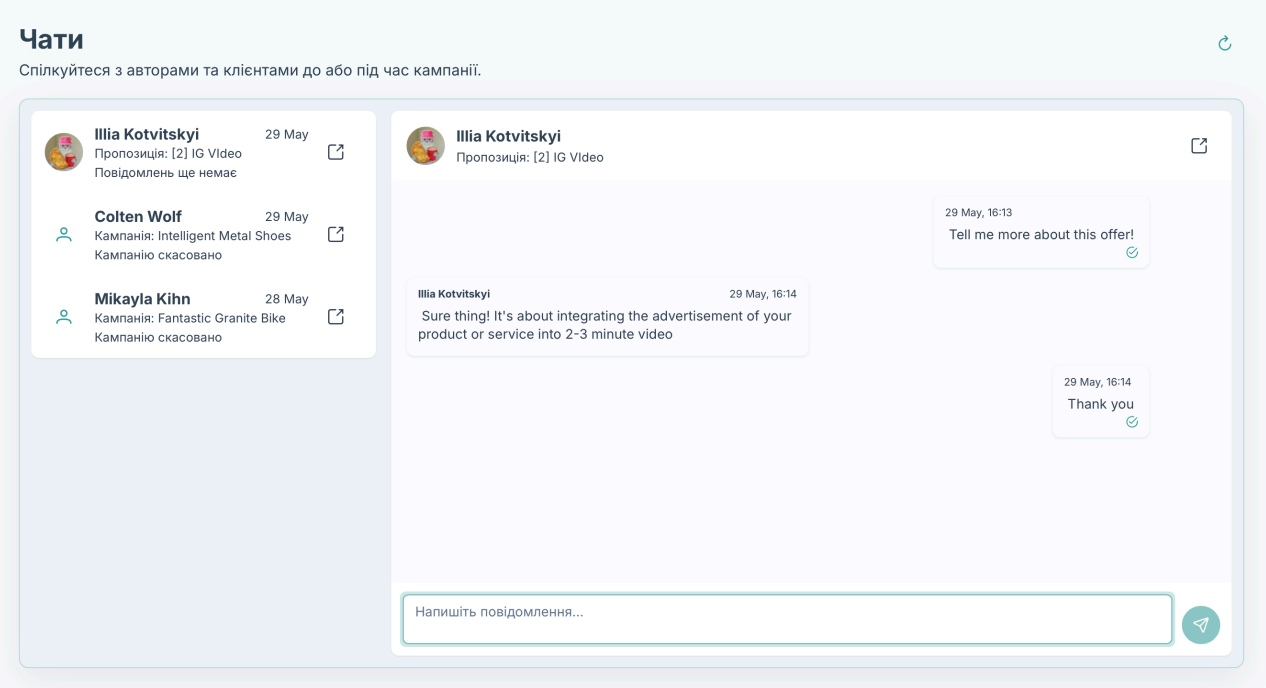


Рис. 3.32. Сторінка чатів

Створення рекламної кампанії здійснюється у модальному вікні (рис. 3.33). Рекламодавець обирає пропозицію блогера, вводить назву та опис кампанії, а також зазначає орієнтовну дату виконання. Після підтвердження кампанія отримує статус Pending і відображається у переліку активних кампаній обох учасників. Блогер може прийняти або відхилити заявку, перш ніж кампанія перейде до активної фази.

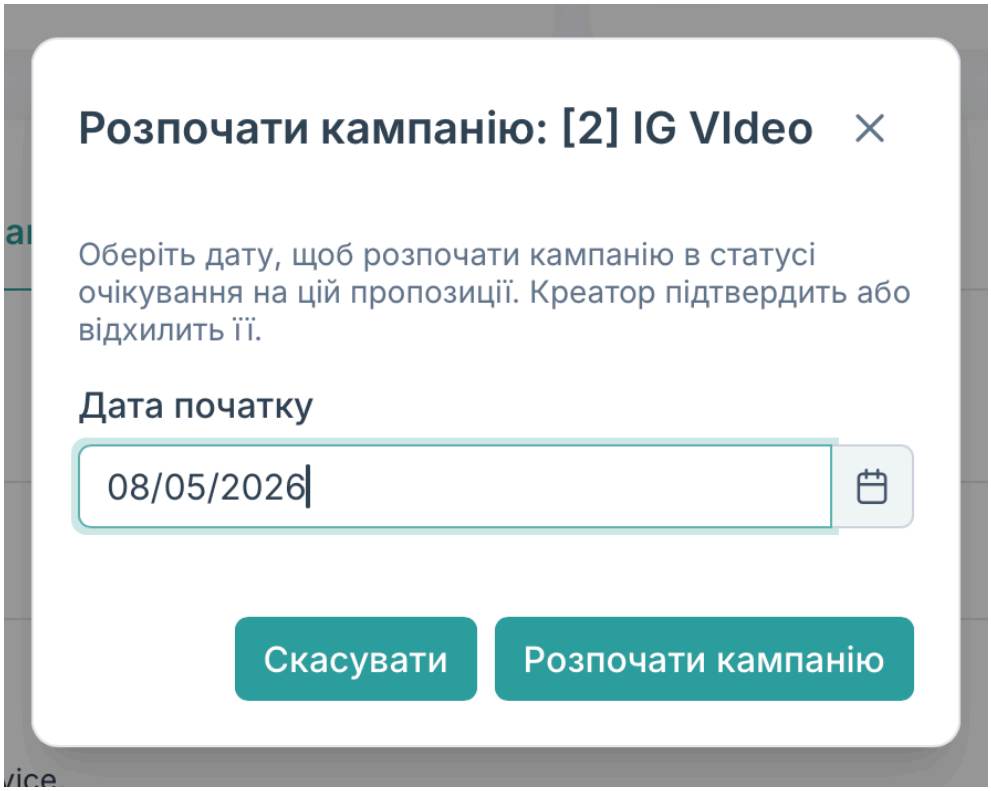


Рис. 3.33. Модальне вікно створення рекламної кампанії

Сторінка кампаній підтримує два режими перегляду. У режимі «Календар» відображається календар поточного місяця з можливістю вибору кампаній і переходу до їх деталей, а також фільтрування за роллю користувача: виконавця або замовника (рис. 3.34, 3.35). Кожна кампанія виводиться на день своєї запланованої дати виконання. Деталізований перегляд дня (рис. 3.35) відображає перелік кампаній, що заплановані або виконуються у цей день. Це дозволяє зручно керувати розкладом та уникати накладання термінів кількох кампаній.

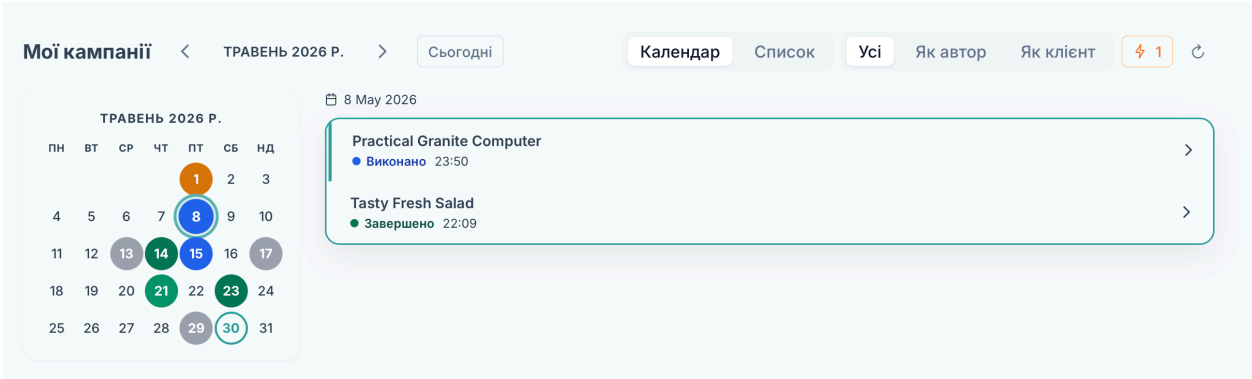


Рис. 3.34. Сторінка кампаній, режим «Календар»

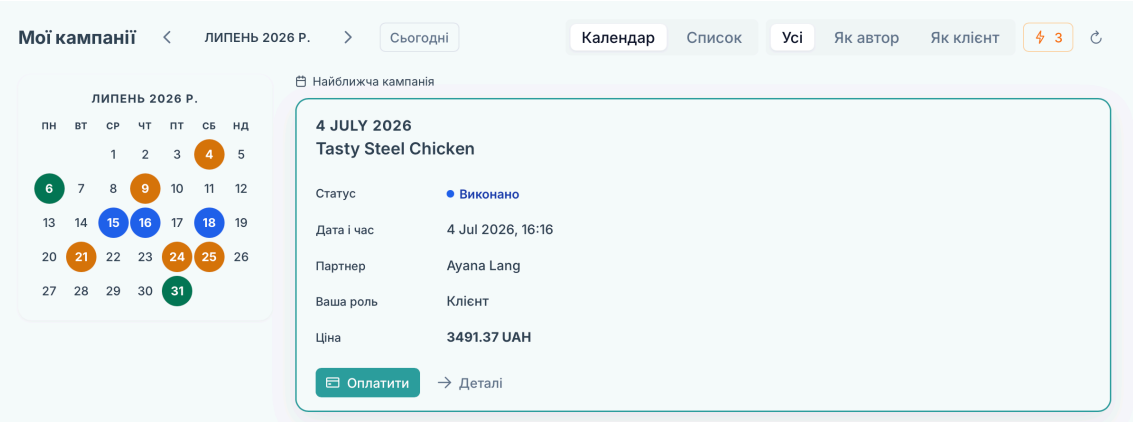


Рис. 3.35. Сторінка кампаній, режим «Календар»: деталізований вигляд

У режимі «Список» рекламні кампанії відображаються у вигляді списку з посторінковою навігацією. Передбачено фільтрування за назвою, статусом, роллю користувача (виконавець або замовник), а також за доступністю певної дії, наприклад, можливістю залишити відгук, завершити чи скасувати кампанію (рис. 3.36). Кожен рядок списку містить основну інформацію про кампанію: назву, поточний статус, дату виконання та ім'я контрагента. Режим списку зручний при роботі з великою кількістю кампаній, оскільки дозволяє швидко переглянути всі позиції та знайти потрібну за допомогою комбінованих фільтрів.

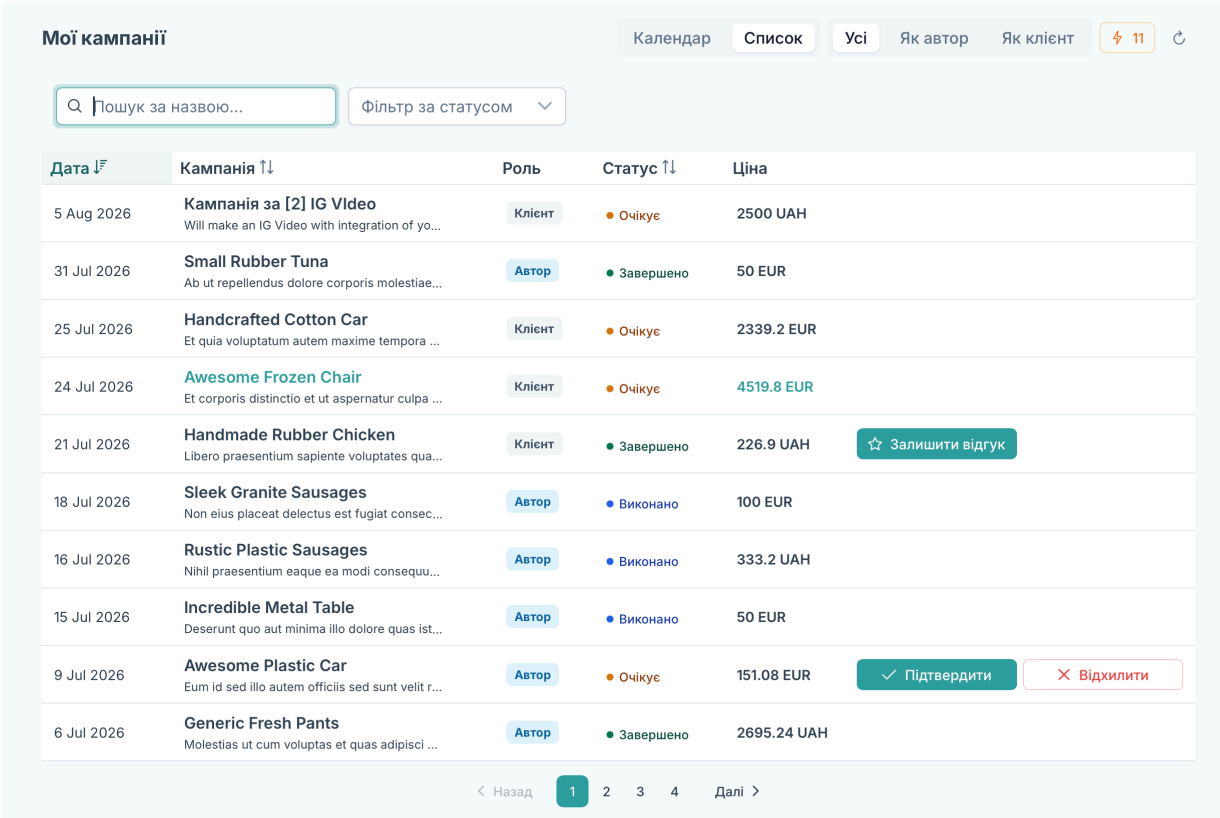


Рис. 3.36. Сторінка кампаній, режим «Список»

Сторінка деталей кампанії містить повну інформацію про неї: назву, опис, ціну, терміни, учасників та хронологію виконаних дій. Кожному учаснику доступні лише призначені йому дії: зокрема, підтвердити кампанію, тобто погодити пропозицію замовника про її початок, може лише виконавець (рис. 3.37). Хронологія дій відображає всі ключові переходи між статусами із датами, що дозволяє відстежувати прогрес кампанії та розв'язувати спірні ситуації. На сторінці також доступні дії для переходу на наступний статус кампанії залежно від поточного стану: завершити, скасувати, надіслати звіт або залишити відгук.

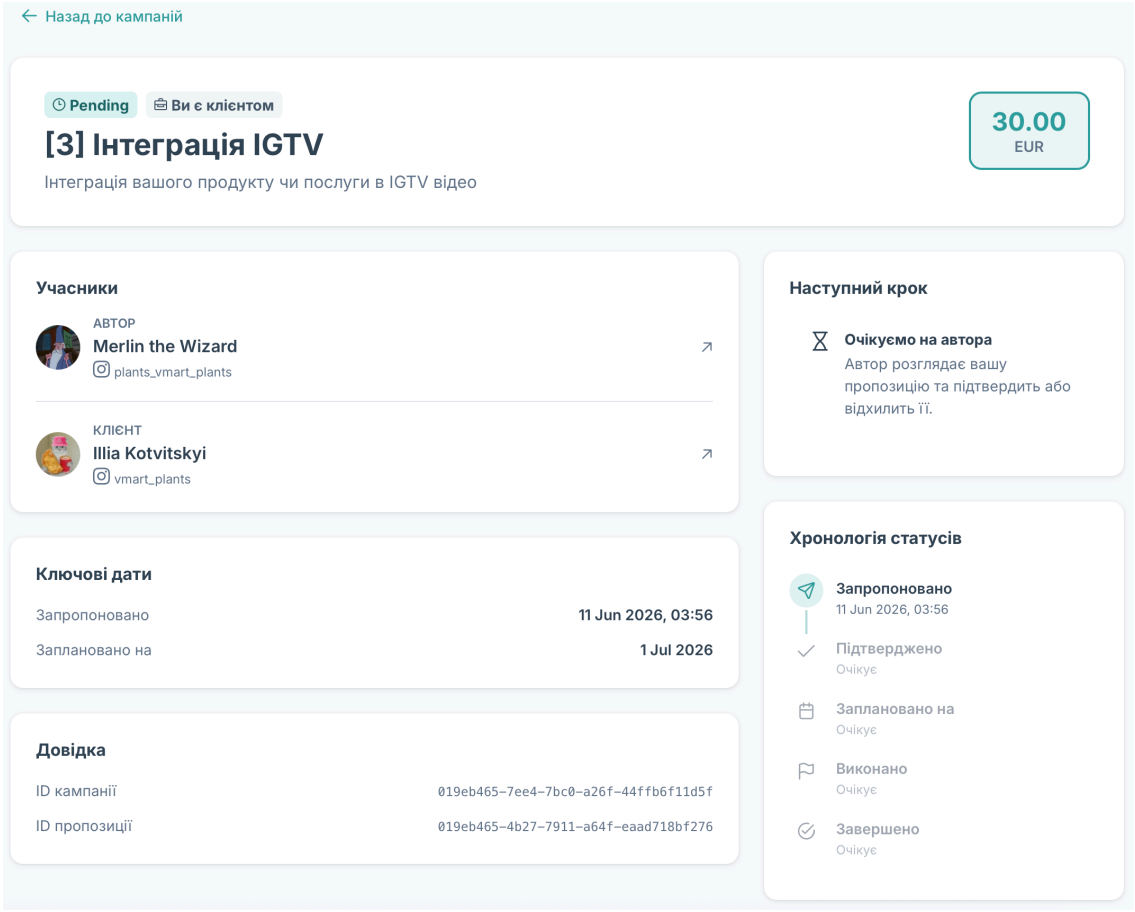


Рис. 3.37. Сторінка деталей рекламної кампанії

Створення звіту про виконану в межах кампанії роботу здійснюється у модальному вікні (рис. 3.38). Виконавець вказує опис виконаних дій та може додати посилання на публікацію або інший матеріал, що підтверджує виконання рекламної інтеграції. Після подачі звіту замовник отримує сповіщення та може перевірити результати перед переходом до виплати. Звіт є обов'язковою умовою для переведення кампанії у фінальний статус та проведення оплати.

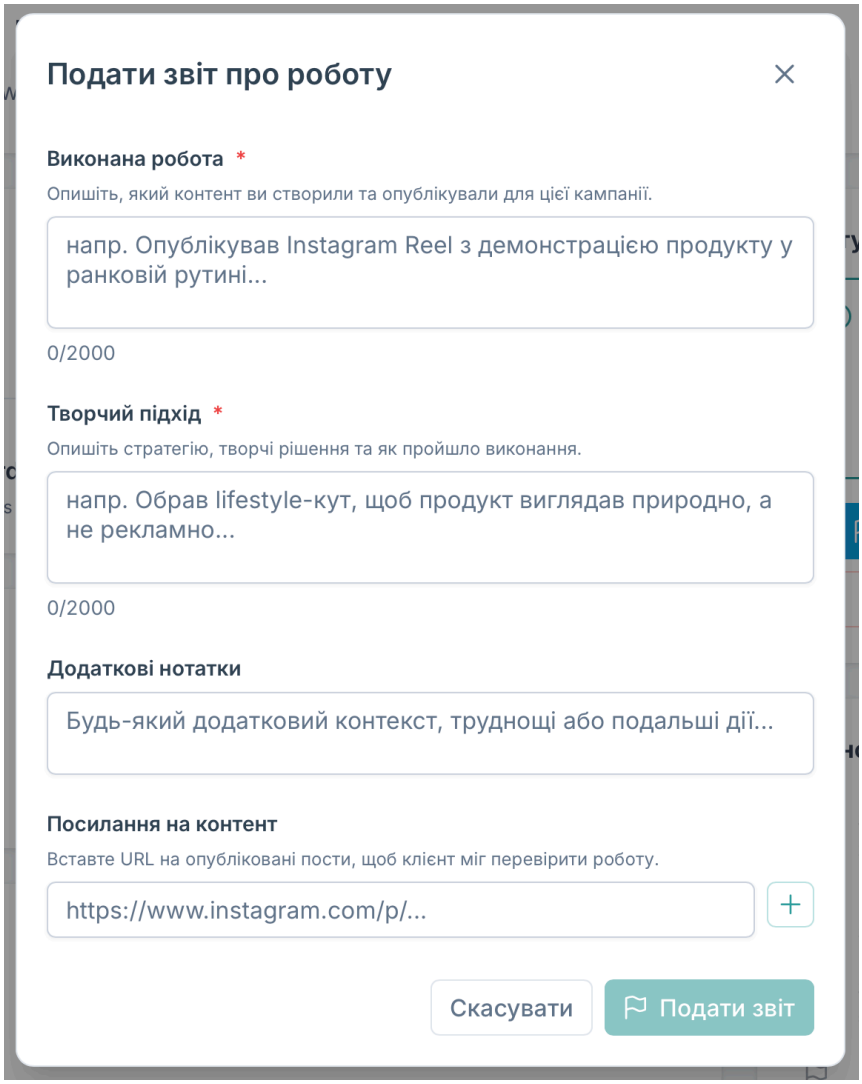


Рис. 3.38. Модальне вікно створення звіту про виконану роботу

Раніше створений звіт можна змінити у відповідному модальному вікні редагування (рис. 3.39). Редагування доступне лише до моменту, поки замовник не підтвердив прийняття роботи та не ініціював виплату. Це дозволяє виконавцю виправити помилки або доповнити звіт актуальними посиланнями та матеріалами, якщо первісна версія потребує уточнень.

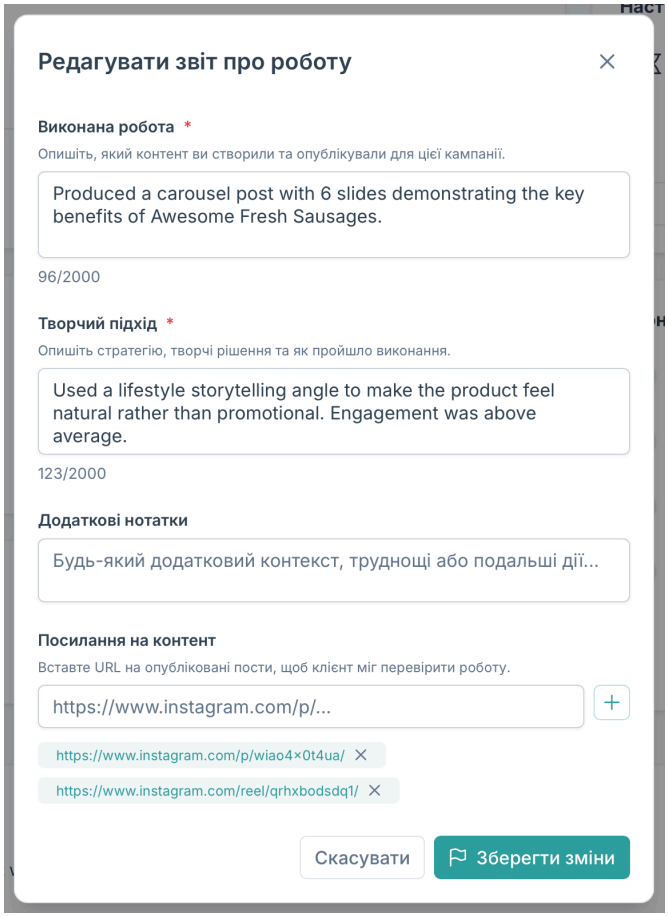


Рис. 3.39. Модальне вікно редагування звіту про виконану роботу

Оплата за виконану роботу здійснюється через відповідне меню (рис. 3.40). Замовник підтверджує суму оплати відповідно до умов пропозиції та ініціює транзакцію через інтегровану платіжну систему. Після успішного проведення платежу кампанія отримує статус Completed, а виконавець отримує виплату на вказаний ним IBAN-рахунок. Деталі платежу зберігаються в системі і доступні для перегляду в розділі фінансових операцій.

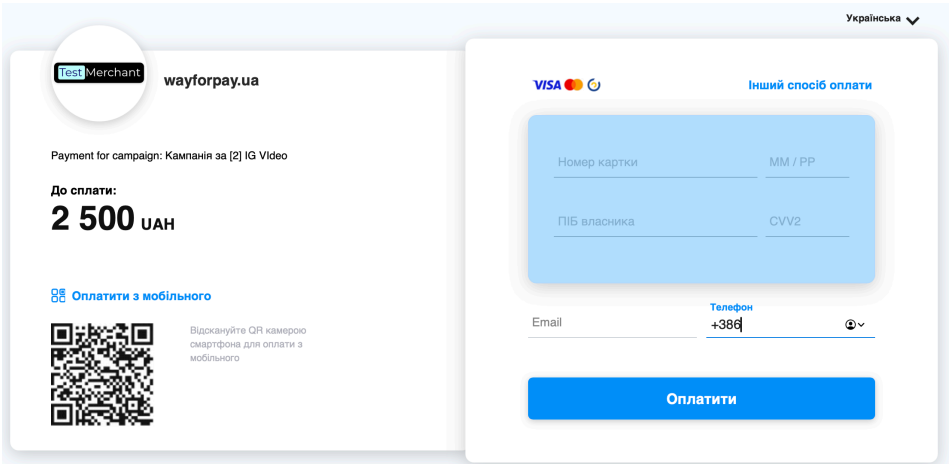


Рис. 3.40. Меню оплати за виконану роботу

Після завершення кампанії замовник може залишити відгук у модальному вікні його створення (рис. 3.41). Відгук містить числову оцінку та текстовий коментар щодо якості роботи виконавця. Після збереження він з'являється у профілі блогера та є доступним для перегляду іншим рекламодавцям. Відгук не підлягає видаленню або редагуванню, що гарантує достовірність репутаційних даних на платформі.

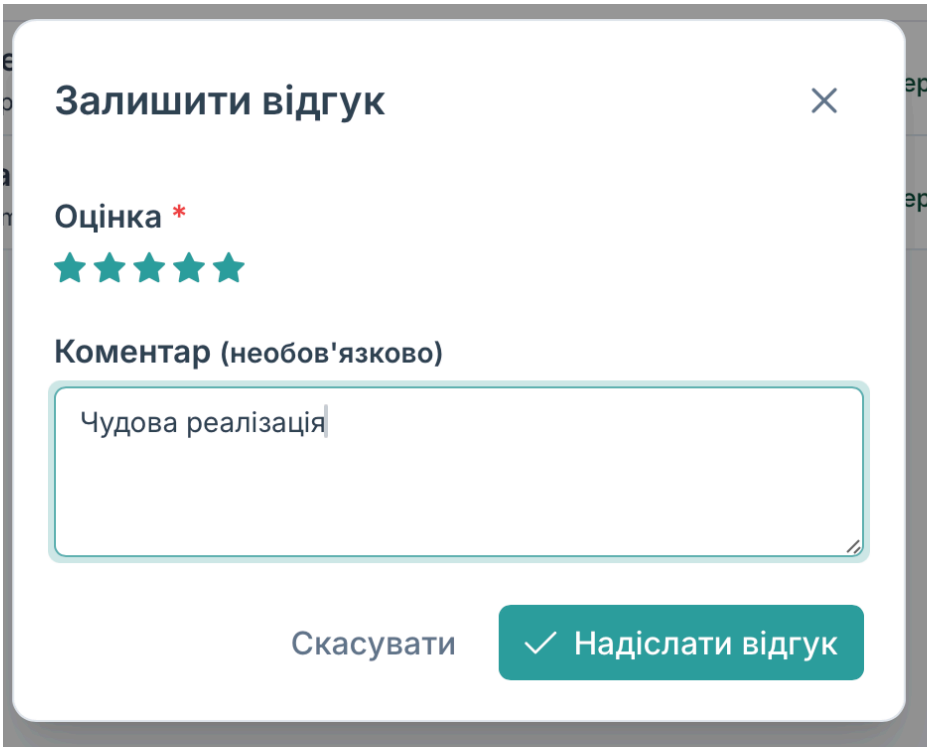


Рис. 3.41. Модальне вікно створення відгуку

3.3 Тестування платформи

Для підтвердження коректності роботи системи та відповідності реалізованої функціональності визначеним вимогам було проведено тестування платформи. Зважаючи на побудову системи за модульною монолітною архітектурою, тестування організовано окремо для кожного функціонального модуля, що дозволяє локалізувати можливі помилки та перевіряти модулі незалежно один від одного.

Автоматизоване тестування реалізовано засобами платформи .NET і охоплює три категорії перевірок. Юніт-тести доменного рівня перевіряють коректність бізнес-логіки та правил предметної області, зокрема переходів станів рекламної кампанії та валідації даних. Юніт-тести прикладного рівня охоплюють обробники команд і запитів, які реалізують сценарії використання системи.

Архітектурні тести контролюють дотримання встановлених правил модульної архітектури, унеможливлюючи небажані залежності між модулями та порушення меж шарів застосунку.

Загальну структуру та результати виконання автоматизованих тестів наведено на рис. 3.42.



Рис. 3.42. Структура та результати виконання автоматизованих тестів

Автоматизованими тестами охоплено чотири основні модулі системи: модуль користувачів, модуль рекламних кампаній, модуль пошуку та підбору партнерів та модуль аналітики. Загальна кількість тестів становить 314, усі вони виконані успішно.

Успішне проходження всіх тестів підтверджує коректність реалізації основних сценаріїв роботи платформи та дотримання архітектурних обмежень, закладених на етапі проєктування.

Окрім автоматизованого тестування, проведено ручну перевірку клієнтської частини. Перевірено адаптивність інтерфейсу під час перегляду на пристроях із різною шириною екрана: за зменшення ширини viewport елементи

сторінки перебудовуються в одноколонкову розкладку, зберігаючи доступність основних дій. Вигляд сторінки профілю на мобільному пристрої наведено на рис. 3.43.

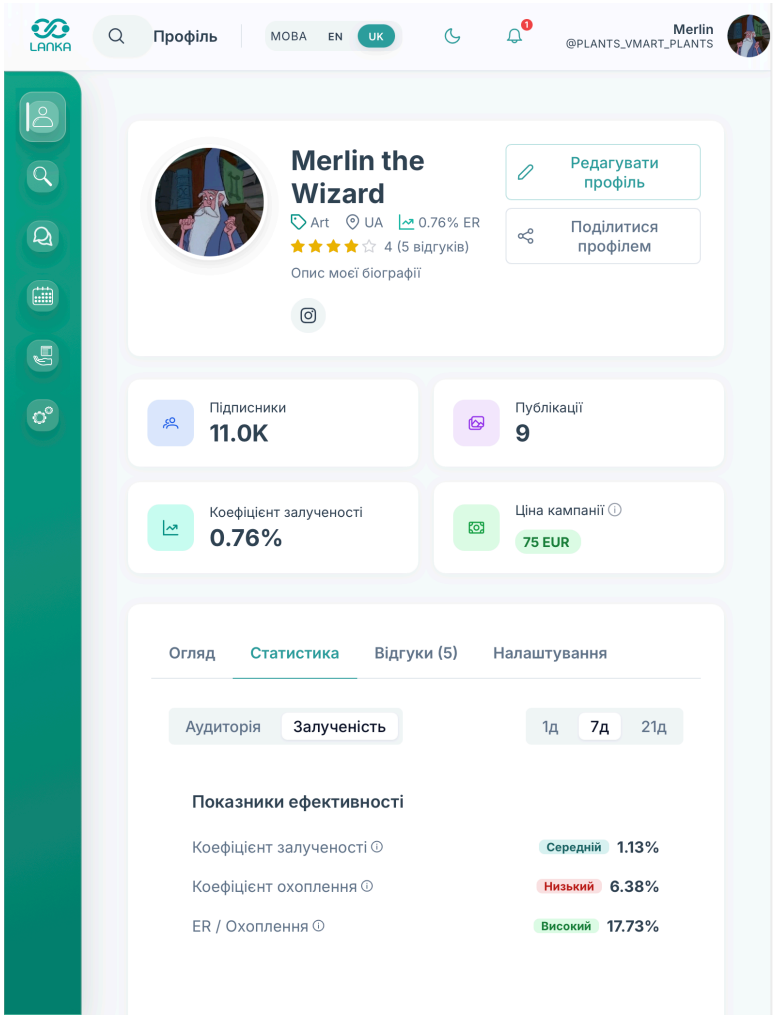


Рис. 3.43. Адаптивний вигляд сторінки профілю на мобільному пристрої

Також виконано аудит доступності (accessibility) інтерфейсу за допомогою інструментів веброзробника. Перевірку пройдено успішно: із 22 контрольних правил доступності 21 виконано повністю, а одне позначено як попередження, що відповідає підсумковій оцінці 95%. Результати аудиту наведено на рис. 3.44.

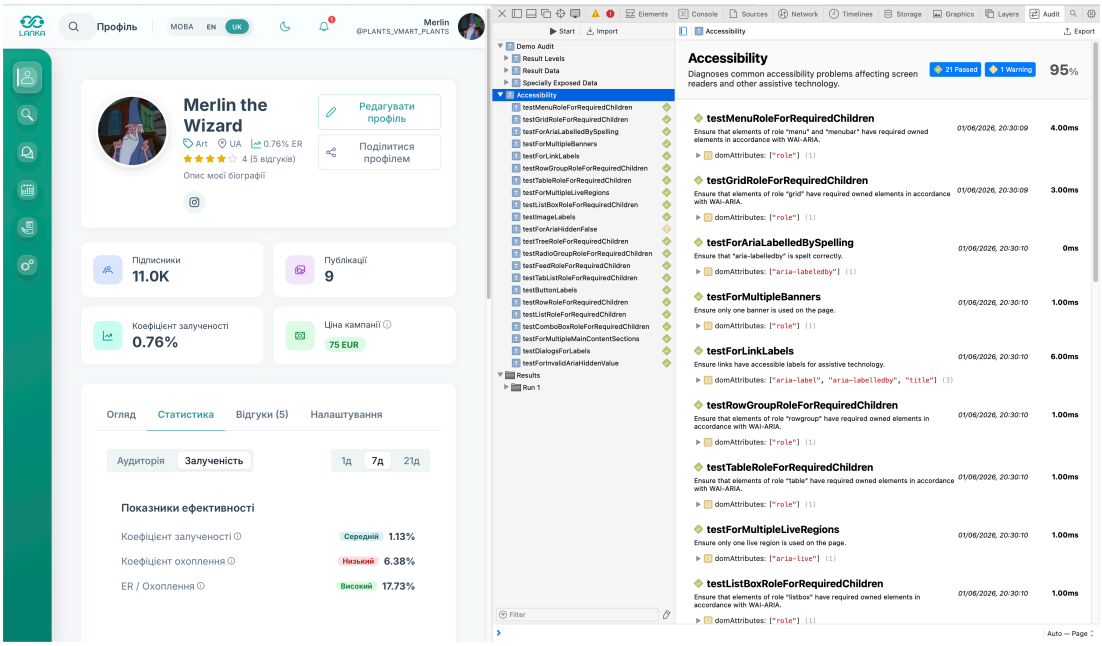


Рис. 3.44. Результати аудиту доступності інтерфейсу

Висновки до третього розділу

У третьому розділі описано встановлення, налаштування та практичне використання розробленої вебплатформи управління рекламними кампаніями в Instagram.

Розглянуто розгортання системи із застосуванням контейнеризації та оркестрації, а також налаштування запуску для різних середовищ. Показано, що система збирається один раз, а її поведінка визначається конфігурацією оточення, що спрощує встановлення й оновлення.

Описано структуру інтерфейсу та основні сценарії роботи: реєстрацію, підключення Instagram-акаунта, налаштування профілю, створення пропозицій, пошук партнерів, перегляд аналітики, листування, супровід кампаній, формування звітів і відгуків.

Проведено автоматизоване й ручне тестування платформи, зокрема перевірку доменного та прикладного рівнів, архітектурних обмежень, адаптивності інтерфейсу й доступності. Результати підтвердили коректність основних сценаріїв роботи системи.

Отримані результати свідчать про працездатність платформи та її відповідність функціональним вимогам.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було розроблено вебплатформу управління рекламними кампаніями в соціальній мережі Instagram. Проаналізовано предметну область цифрового маркетингу та виконано огляд існуючих програмних продуктів для організації співпраці між брендами та інфлюенсерами. На основі порівняння найбільш релевантних аналогів визначено їхні переваги, недоліки та функціональні можливості, що дало змогу сформувати вимоги до власного рішення та обґрунтувати доцільність його розробки.

Обґрунтовано вибір архітектури та інструментальних засобів реалізації, спроєктовано структуру програмної платформи та модель даних. Для платформи було обрано модульну монолітну архітектуру, що забезпечує логічне розділення функціональності, спрощує супровід системи та створює передумови для її подальшого розвитку.

Реалізовано серверну та клієнтську частини вебплатформи. Розроблена система забезпечує виконання основних функцій, необхідних для організації рекламної співпраці між блогерами та брендами у межах основних сценаріїв рекламної співпраці. Також було продемонстровано інтерфейс платформи й порядок роботи з основними сценаріями, що підтвердило логічність структури системи та зручність взаємодії користувача з її функціональністю.

Проведено тестування розробленого програмного продукту із застосуванням юніт-тестів доменного та прикладного рівнів, а також архітектурних тестів. Юніт-тести дали змогу перевірити правильність роботи окремих бізнес-правил, сервісів і компонентів системи, а архітектурні тести підтвердили дотримання визначених залежностей між модулями та відповідність реалізації обраному архітектурному підходу.

У результаті виконання кваліфікаційної роботи було досягнуто поставлену мету і спроєктовано та реалізовано вебплатформу для централізованого управління рекламними кампаніями в Instagram.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Котвіцький І.В., Савіцький Р.С. Типозабезпечена обробка помилок з патерном Result. Збірник тез VII Міжнародної конференції молодих учених «Актуальні питання розвитку інформаційних технологій», 19 листопада 2025 року. Дніпро: ДВНЗ «Приазовський державний технічний університет», 2025.
2. Котвіцький І.В., Савіцький Р.С. Узгодженість внутрішніх домовленостей як фундамент якості в інженерії програмного забезпечення. Матеріали VIII Всеукраїнської науково-практичної інтернет-конференції молодих вчених та студентів «Сучасні інформаційні системи та технології», 24 листопада 2025 року. Хмельницький: Хмельницький національний університет, 2025.
3. Bogard J. MediatR Wiki [Електронний ресурс]. URL: https://github.com/jbogard/MediatR/wiki (дата звернення: 20.05.2026).
4. Conde R., Casais B. Micro, macro and mega-influencers on Instagram: The power of persuasion via the parasocial relationship. Journal of Business Research. 2023. Vol. 158. Article 113708. DOI: 10.1016/j.jbusres.2023.113708.
5. Elastic. Elasticsearch Guide [Електронний ресурс]. URL: https://www.elastic.co/guide/ (дата звернення: 20.05.2026).
6. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2003. 560 p.
7. Fowler M. Modular Monolith: Microservices Alternative [Електронний ресурс]. URL: https://martinfowler.com/articles/modular-monolith.html (дата звернення: 20.05.2026).
8. Fowler M. Monolith First [Електронний ресурс]. URL: https://martinfowler.com/bliki/MonolithFirst.html (дата звернення: 20.05.2026).
9. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

10. Google LLC. Angular Documentation [Електронний ресурс]. URL: https://angular.dev/overview (дата звернення: 20.05.2026).
11. Grzybek K. Modular Monolith: A Primer [Електронний ресурс]. URL: https://www.kamilgrzybek.com/blog/posts/modular-monolith-primer (дата звернення: 20.05.2026).
12. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston : Addison-Wesley, 2003. 736 p.
13. Influencer Marketing Hub. Influencer Marketing Benchmark Report 2024 [Електронний ресурс]. URL: https://influencermarketinghub.com/influencer-marketing-benchmark-report/ (дата звернення: 20.05.2026).
14. Keycloak Documentation [Електронний ресурс]. URL: https://www.keycloak.org/documentation (дата звернення: 20.05.2026).
15. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2017. 616 p.
16. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
17. MassTransit Documentation [Електронний ресурс]. URL: https://masstransit.io/documentation (дата звернення: 20.05.2026).
18. MassTransit. Saga Pattern [Електронний ресурс]. URL: https://masstransit.io/documentation/patterns/saga (дата звернення: 20.05.2026).
19. Meta Platforms, Inc. Instagram Graph API [Електронний ресурс]. URL: https://developers.facebook.com/docs/instagram-platform/instagram-graph-api/ (дата звернення: 20.05.2026).
20. Microsoft. ASP.NET Core documentation [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 20.05.2026).
21. Microsoft. CQRS Pattern [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs (дата звернення: 20.05.2026).

22. Microsoft. Entity Framework Core Documentation [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 20.05.2026).
23. Microsoft. TypeScript Documentation [Електронний ресурс]. URL: https://www.typescriptlang.org/docs/ (дата звернення: 20.05.2026).
24. MongoDB, Inc. MongoDB Manual [Електронний ресурс]. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 20.05.2026).
25. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 612 p.
26. Nottingham M., Wilde E. RFC 7807: Problem Details for HTTP APIs [Електронний ресурс]. URL: https://datatracker.ietf.org/doc/html/rfc7807 (дата звернення: 20.05.2026).
27. RabbitMQ Documentation [Електронний ресурс]. URL: https://www.rabbitmq.com/documentation.html (дата звернення: 20.05.2026).
28. Richardson C. Pattern: Saga [Електронний ресурс]. URL: https://microservices.io/patterns/data/saga.html (дата звернення: 20.05.2026).
29. Richardson C. Pattern: Transactional Outbox [Електронний ресурс]. URL: https://microservices.io/patterns/data/transactional-outbox.html (дата звернення: 20.05.2026).
30. Statista. Influencer marketing market size worldwide from 2015 to 2025 [Електронний ресурс]. URL: https://www.statista.com/statistics/1092819/global-influencer-market-size/ (дата звернення: 20.05.2026).
31. TechEmpower. Web Framework Benchmarks. Round 22 [Електронний ресурс]. URL: https://www.techempower.com/benchmarks/ (дата звернення: 20.05.2026).
32. The PostgreSQL Global Development Group. PostgreSQL Documentation. Version 16 [Електронний ресурс]. URL: https://www.postgresql.org/docs/16/ (дата звернення: 20.05.2026).

33. Young G. CQRS Documents [Електронний ресурс]. URL: https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf (дата звернення: 20.05.2026).

ДОДАТКИ

Додаток А

Лістинг вихідного коду

Лістинг А.1:

```
internal sealed class PendCampaignCommandHandler
    : ICommandHandler<PendCampaignCommand, Guid>
{
    private readonly IPactRepository _pactRepository;
    private readonly ICampaignRepository _campaignRepository;
    private readonly IOfferRepository _offerRepository;
    private readonly IUnitOfWork _unitOfWork;
    private readonly IDateTimeProvider _dateTimeProvider;
    private readonly IUserContext _userContext;

    public PendCampaignCommandHandler(
        IPactRepository pactRepository,
        ICampaignRepository campaignRepository,
        IOfferRepository offerRepository,
        IUnitOfWork unitOfWork,
        IDateTimeProvider dateTimeProvider,
        IUserContext userContext
    )
    {
        this._pactRepository = pactRepository;
        this._campaignRepository = campaignRepository;
        this._offerRepository = offerRepository;
        this._unitOfWork = unitOfWork;
        this._dateTimeProvider = dateTimeProvider;
        this._userContext = userContext;
    }

    public async Task<Result<Guid>> Handle(PendCampaignCommand request,
CancellationToken cancellationToken)
    {
        Offer? offer = await this._offerRepository.GetByIdAsync(
            request.OfferId,
            cancellationToken
        );

        if (offer is null)
        {
            return Result.Failure<Guid>(OfferErrors.NotFound);
        }

        Pact? creatorPact = await this._pactRepository.GetByIdWithOffersAsync(
            offer.PactId,
            cancellationToken
        );

        if (creatorPact is null)
        {
            return Result.Failure<Guid>(OfferErrors.NotFound);
        }

        var clientId = new BloggerId(this._userContext.GetUserId());

        if (clientId == creatorPact.BloggerId)
        {
            return Result.Failure<Guid>(CampaignErrors.SameUser);
        }

        if (!creatorPact.HasOffer(offer.Name))
        {
```

```
            return Result.Failure<Guid>(OfferErrors.NotFound);
        }

        if (
            await this._campaignRepository.IsAlreadyStartedAsync(
                offer,
                request.ScheduledOnUtc,
                cancellationToken
            )
        )
        {
            return Result.Failure<Guid>(CampaignErrors.AlreadyStarted);
        }

        try
        {
            Result<Campaign> result = Campaign.Pend(
                request.Name,
                request.Description,
                request.ScheduledOnUtc,
                offer,
                clientId,
                creatorPact.BloggerId,
                this._dateTimeProvider.UtcNow
            );

            if (result.IsFailure)
            {
                return Result.Failure<Guid>(result.Error);
            }

            Campaign campaign = result.Value;

            this._campaignRepository.Add(campaign);

            await this._unitOfWork.SaveChangesAsync(cancellationToken);

            return campaign.Id.Value;
        }
        catch (ConcurrencyException)
        {
            return Result.Failure<Guid>(CampaignErrors.AlreadyStarted);
        }
    }
}
```

Лістинг А.2:

```
internal sealed class CampaignPendedDomainEventHandler
    : DomainEventHandler<CampaignPendedDomainEvent>
{
    private readonly ISender _sender;
    private readonly IEventBus _eventBus;
    private readonly INotificationRepository _notificationRepository;
    private readonly IChatThreadRepository _chatThreadRepository;
    private readonly IChatMessageRepository _chatMessageRepository;
    private readonly IUnitOfWork _unitOfWork;
    private readonly IChatNotificationService _chatNotificationService;


    public CampaignPendedDomainEventHandler(
        ISender sender,
        IEventBus eventBus,
        INotificationRepository notificationRepository,
```

```
        IChatThreadRepository chatThreadRepository,
        IChatMessageRepository chatMessageRepository,
        IUnitOfWork unitOfWork,
        IChatNotificationService chatNotificationService)
    {
        this._sender = sender;
        this._eventBus = eventBus;
        this._notificationRepository = notificationRepository;
        this._chatThreadRepository = chatThreadRepository;
        this._chatMessageRepository = chatMessageRepository;
        this._unitOfWork = unitOfWork;
        this._chatNotificationService = chatNotificationService;
    }

    public override async Task Handle(
        CampaignPendedDomainEvent notification,
        CancellationToken cancellationToken = default
    )
    {
        Result<CampaignResponse> result = await this._sender.Send(
            new GetCampaignQuery(notification.CampaignId.Value),
            cancellationToken
        );

        if (result.IsFailure)
        {
            throw new LankaException(nameof(GetCampaignQuery),
CampaignErrors.NotFound);
        }

        var occurredAt = new DateTimeOffset(notification.OccurredOnUtc,
TimeSpan.Zero);

        this._notificationRepository.Add(Notification.Create(
            new BloggerId(result.Value.CreatorId),
            result.Value.Id,
            "New campaign proposal",
            $"You received a new campaign proposal '{result.Value.Name}'",
            occurredAt
        ));

        Result<ChatThread> threadResult = await
CampaignChatThreadResolver.GetOrCreateAsync(
            this._chatThreadRepository,
            result.Value,
            occurredAt,
            cancellationToken);

        if (threadResult.IsFailure)
        {
            throw new LankaException(nameof(CampaignChatThreadResolver),
threadResult.Error);
        }

        var systemMessage = ChatMessage.CreateSystemMessage(
            threadResult.Value.Id,
            "CAMPAIGN_REQUESTED",
            occurredAt
        );

        this._chatMessageRepository.Add(systemMessage);

        await this._unitOfWork.SaveChangesAsync(cancellationToken);

        await this._chatNotificationService.SendMessageAsync(
```

```
            SystemChatNotificationFactory.Create(systemMessage),
            cancellationToken
        );

        await this._eventBus.PublishAsync(
            new CampaignNotificationIntegrationEvent(
                Guid.CreateVersion7(),
                notification.OccurredOnUtc,
                result.Value.CreatorId,
                result.Value.Id,
                result.Value.Name,
                "Pending"
            ),
            cancellationToken
        );

        await this._eventBus.PublishAsync(
            new CampaignPendedIntegrationEvent(
                notification.Id,
                notification.OccurredOnUtc,
                notification.CampaignId.Value,
                result.Value.OfferId,
                result.Value.ClientId,
                result.Value.CreatorId
            ),
            cancellationToken
        );
    }
}
```

Лістинг А.3:

```
internal sealed class CampaignsChangeCaptureInterceptor :
ChangeCaptureInterceptorBase
{
    protected override CapturedChangeData? ExtractCapturedData(IChangeCaptured
entity)
    {
        return entity switch
        {
            Blogger b => new CapturedChangeData(
                nameof(Blogger),
                $"{b.FirstName.Value} {b.LastName.Value}",
                CreateBloggerSearchContent(b),
                CreateBloggerSearchTags(b),
                new Dictionary<string, object>
                {
                    { "BirthDate", b.BirthDate.Value.ToString("yyyy-MM-dd",
CultureInfo.InvariantCulture) },
                    { "Category", b.Category.Name },
                    { "InstagramUsername", b.InstagramMetadata.Username ??
string.Empty },
                    { "FollowersCount", b.InstagramMetadata.FollowersCount ?? 0
},
                    { "MediaCount", b.InstagramMetadata.MediaCount ?? 0 },
                    { "EngagementRate", b.InstagramMetadata.EngagementRate ??
0.0 },
                    { "AudienceTopAgeGroup",
b.InstagramMetadata.AudienceTopAgeGroup ?? string.Empty },
                    { "AudienceTopGender", b.InstagramMetadata.AudienceTopGender
?? string.Empty },
                    { "AudienceTopGenderPercentage",
b.InstagramMetadata.AudienceTopGenderPercentage ?? 0.0 },
                    { "AudienceTopCountry",
```

```
b.InstagramMetadata.AudienceTopCountry ?? string.Empty },
                    { "AudienceTopCountryPercentage",
b.InstagramMetadata.AudienceTopCountryPercentage ?? 0.0 },
                }),

            Campaign c => new CapturedChangeData(
                nameof(Campaign),
                c.Name.Value,
                c.Description.Value,
                ["campaign", c.Status.ToString()],
                new Dictionary<string, object>
                {
                    { "Status", c.Status.ToString() },
                    { "PriceAmount", c.Price.Amount },
                    { "PriceCurrency", c.Price.Currency.ToString() },
                    { "CreatorId", c.CreatorId.Value.ToString() },
                    { "ClientId", c.ClientId.Value.ToString() },
                    { "ScheduledOnUtc", c.ScheduledOnUtc.ToString("o",
CultureInfo.InvariantCulture) },
                    { "PendedOnUtc", c.PendedOnUtc.ToString("o",
CultureInfo.InvariantCulture) },
                }),

            Offer o => new CapturedChangeData(
                nameof(Offer),
                o.Name.Value,
                o.Description.Value,
                ["offer"],
                new Dictionary<string, object>
                {
                    { "PriceAmount", o.Price.Amount },
                    { "PriceCurrency", o.Price.Currency.ToString() },
                }),

            Pact p => new CapturedChangeData(
                nameof(Pact),
                "Pact",
                p.Content.Value,
                ["pact"]),

            Review r => new CapturedChangeData(
                nameof(Review),
                $"Review ({r.Rating.Value}/5)",
                r.Comment.Value,
                ["review"],
                new Dictionary<string, object>
                {
                    { "Rating", r.Rating.Value },

                }),


            _ => null,
        };
    }

    private static string CreateBloggerSearchContent(Blogger blogger)
    {
        var content = new StringBuilder(blogger.Bio.Value);

        if (!string.IsNullOrWhiteSpace(blogger.Category.Name))
        {
            content.Append(" | Category: ").Append(blogger.Category.Name);
        }
```

```
        if (!string.IsNullOrWhiteSpace(blogger.InstagramMetadata.Username))
        {
            content.Append(" | Instagram:
@").Append(blogger.InstagramMetadata.Username);
        }

        if (blogger.InstagramMetadata.FollowersCount.HasValue)
        {
            content.Append(" | Followers:
").Append(blogger.InstagramMetadata.FollowersCount.Value);
        }

        return content.ToString();
    }

    private static string[] CreateBloggerSearchTags(Blogger blogger)
    {
        var tags = new List<string> { "blogger", "profile",
blogger.Category.Name };

        if (!string.IsNullOrWhiteSpace(blogger.InstagramMetadata.Username))
        {
            tags.Add("Instagram");
            tags.Add(blogger.InstagramMetadata.Username);
        }

        return tags.ToArray();
    }

    protected override string? GetItemType(IChangeCaptured entity)
    {
        return entity switch
        {
            Blogger => nameof(Blogger),
            Campaign => nameof(Campaign),
            Offer => nameof(Offer),
            Pact => nameof(Pact),
            Review => nameof(Review),
            _ => null,
        };
    }
}
```