Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально – науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему **"Розроблення чатботу з просування рекламних послуг "telega.store"**

Виконав: студент IV курсу, групи КН-41
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Коваляк В. Б.
(прізвище та ініціали)

Керівники **Головата С.Б, Мокрицька О.В.**
(прізвище та ініціали)

Рецензент Колесник К.
(прізвище та ініціали)

Львів – 2025 року

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Борецька І.Б.
"10" червня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Коваляк Володимир Богданович
(прізвище, ім'я, по батькові)

1.Тема роботи Розроблення чатботу з просування рекламних послуг "telega.store

керівники роботи асистент кафедри КН Головата Софія Богданівна, доцент кафедри КН Мокрицька Ольга Володимирівна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "15" листопада 2024 р. № С-882
2. Строк подання студентом роботи 10.06.2025 р.
3. Вихідні дані до роботи Формулювання задачі та її формалізація. Аналіз попередніх досліджень. Огляд програмних засобів для реалізації поставленого завдання.

4.Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1.Стан проблемної області
2.Інформаційне та математичне забезпечення
3.Програмне та технічне забезпечення
4. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайди доповіді, актуальність теми, постановка завдання, аналіз отриманих результатів, висновки

6.Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних та інших джерел згідно досліджуваної теми. Збір потрібних матеріалів.	19.11.2024-19.12.2024 р.	виконано
2	Постановка задачі та її формалізація.	19.12.2024-27.12.2024 р.	виконано
3	Вибір та обгрунтування методів і засобів розв'язання завдання.	27.12.2024-27.01.2025 р	виконано
4	Програмна реалізація системи.	27.01.2025-27.03.2025 р.	виконано
5	Оформлення опису створеної програми.	27.03.2025-27.05.2025 р.	виконано
6	Аналіз отриманих результатів виконання програми.	29.05.2025 р.	виконано
7	Здача пояснювальної записки на перевірку та виправлення виявлених помилок.	29.05.2025-10.06.2025 р.	виконано

Студент Коваляк (підпис) Коваляк В. Б. (прізвище та ініціали)

Керівник проекту (роботи) (підпис) Головата С.Б. (прізвище та ініціали)

Керівник проекту (роботи) (підпис) Мокрицька О.В. (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 109 сторінок пояснювальної записки, 38 рисунків, 1 таблицю та 14 джерел.

У ході виконання завдання було розроблено систему торгівлі рекламою у месенджері, що спрямована на автоматизацію діяльності та скорочення навантаження на продавців реклами, адміністраторів та менеджерів каналів.

Проведено дослідження існуючих аналогів програмного забезпечення в публічному доступі та їх критичний аналіз. Спроектована та розроблена інформаційно-біржева система просування реклами на базі месенджеру Telegram, що забезпечує надійні, швидкі та зручні купівлю-продаж реклами по найвигіднішим умовам, відкладений постинг, постійний збір та оновлення статистичних даних про існуючі в базі Telegram-канали.

Ключові слова: Реклама, Telegram – бот, месенджер, чат, рекламний пост, рекламний таймінг, ER, підписник, приріст.

ABSTRACT

The thesis contains 109 pages of explanatory note, 38 figures, 1 tables and 14 sources.

In the course of the task, we developed an advertising trading system in the messenger aimed at automating activities and reducing the workload of advertising sellers, administrators and channel managers.

A study of existing software analogues in the public domain and their critical analysis was conducted. An information and exchange system for promoting advertising based on the Telegram messenger was designed and developed, which provides reliable, fast and convenient purchase and sale of advertising on the most favourable terms, deferred posting, constant collection and updating of statistical data on existing Telegram channels.

Keywords: Advertising, Telegram - bot, messenger, chat, advertising post, advertising timing, ER, subscriber, growth.

Технічне завдання

- Необхідно розробити інформаційно – біржевий сервіс з просування рекламних послуг “telega.store” на базі месенджеру.
- Головними вимогами до системи є:

~ забезпечувати авторизацію й аутентифікацію користувач;

~ система ролей користувачів в системі;

~ перегляд бази Telegram – каналів;

~ зв’язок з власниками Telegram – каналів (або їх менеджерами);

~ перегляд повідомлень про продаж реклами;

~ перегляд повідомлень про купівлю реклами;

~ створення запису про купівлю реклами;

~ перегляд заявок на власний запис про купівлю реклами;

~ створення заявки у повідомленні про продаж реклами;

~ видалення власної заявки;

~ видалення власного повідомлення про купівлю реклами.

- Розроблена система, повинна містити весь перерахований вище функціонал.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ER	Engagement Rate (коефіцієнт залучення – відношення кількості переглядів до кількості підписників)
API	Application Programming Interface (прикладний програмний інтерфейс)
CI	Continuous Integration (безперервна інтеграція)
CD	Continuous Deployment (безперервне розгортання)
DB	Database (БД – база даних)
REST	Representational State Transfer (передача стану представлення)
EF	Entity Framework

ВСТУП

Актуальність роботи

Зараз дуже важко просувати свої послуги і сервіси без реклами. Тим не менш, якщо для просування сайту може бути достатньо зробити грамотну SEO-оптимізацію, а для YouTube-каналу правильно вказати ранжируванні теги, то з Telegram-каналами все набагато складніше. У месенджері немає ніяких рекомендацій, і Вас ніхто крім знайомих не зможе запросити у новий канал. Виходячи з цього, для просування своїх каналів адміністратори вдаються до закупівлі реклами у інших. Крім каналів через месенджер просувають також ігри, сайти та різноманітні послуги. Знайти відповідний канал за прийнятною ціною іноді буває дуже важко. Не дивлячись на те, що месенджер є досить сучасним і просунутим, він не призначений для розміщення реклами і його специфіка перешкоджає зручним купівлі та продажу. По-перше, рекламодавець повинен спочатку підібрати канал для розміщення, далі канал називає вартість реклами та свої вимоги щодо контенту поста. Після другого етапу велика частина кандидатів відсіюється. Якщо рекламодавцю пощастить знайти необхідний канал, його чекає етап запиту реквізитів, оплати, її перевірки та після розміщення поста перевірка виконання умов публікації вручну. Для спрощення та скорочення усіх вищеназваних процесів і потрібен ресурс, де можна буде ознайомитись з базою Telegram-каналів, їх цінами, замовити рекламу по найвигіднішій ціні. Тому основною метою цього програмного продукту є надання можливості рекламодавцям знайти потрібну площадку для розміщення їх рекламних постів, а рекламістові або ж продавцеві, навпаки, знайти клієнта, який погодиться придбати публікацію.

За офіційними даними станом на 2019 рік щомісячно налічувалося 30 мільярдів переглядів у більше 120 тисяч комерційних каналів із загальною кількістю підписників понад 80 мільярдів. За опитуванням адміністраторів деяких Telegram-каналів було з’ясовано, що середня ціна за одну тисячу переглядів рекламного поста сягає 50 - 500 грн. Отже, з умови, що 10% контенту каналу складає реклама, можемо припустити, що щомісячний грошовий обіг з реклами у месенджері складає 0.15 - 1.5 трильйонів гривень, що в переводі у долари складатиме 5.5 - 55 мільярдів доларів. Наразі купівля-продаж реклами у звичайному режимі включає в себе наступні пункти:

- р*екламодавець*: пошук відповідного каналу через інші канали або за допомогою сторонніх ресурсів;
- *продавець реклами*: очікування запитів по рекламі;
- Рекламодавець: особисте повідомлення менеджеру для уточнення цін, вільних місць;
- *продавець реклами*: відповідь на особисте повідомлення;
- *рекламодавець*: створення та надсилання поста, формування кнопок, реакцій;
- *продавець реклами*: консультування стосовно поста, створення фінального варіанту реклами;
- *рекламодавець*: запит реквізитів, оплата реклами;
- *продавець реклами*: надання реквізитів;
- *продавець реклами*: перевірка оплати;
- *продавець реклами*: формування поста в боті для відкладених постів;
- *рекламодавець*: перевірка виконання умов реклами;
- *рекламодавець*: самостійний аналіз результатів реклами.

Далі представлена цільова схема купівлі-продажу реклами, що, в свою чергу, складається з чотирьох пунктів замість дев’яти:

- *рекламодавець*: пошук каналу / поста за допомогою зручного фільтру;
- *рекламодавець*: надсилання запиту на рекламу;
- *продавець реклами*: вибір найкращої пропозиції, прийняття запиту;
- *рекламодавець*: оплата реклами.

Решта пунктів скорочуються та полегшуються за рахунок автоматизації з використанням Telegram-бота, а саме: пошук каналу, перевірка оплати реклами, відкладена публікація поста, перевірка виконання умов.

Основною перевагою даної схеми є те, що вона працює в обидві сторони між рекламодавцем і продавцем реклами та забезпечують найвигідніші умови. Таким чином, тема розробки сервісу з купівлі-продажу реклами у Telegram є дійсно актуальною як для рекламодавців, так і для продавців реклами, тобто адміністраторів каналів.

Метою дипломної роботи є розробка інформаційно-біржевого сервісу з просування рекламних послуг «telega.store» на базі месенджеру Telegram.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

1. виконати аналіз предметної області;
2. вивчити існуючі рішення і зробити постановку задачі;
3. визначити функціональні вимоги до системи;
4. спроектувати логічну і фізичну структуру системи;
5. провести тестування системи.

Об’єкт дослідження – процеси купівлі – продажу реклами у Telegram.

Предмет дослідження – алгоритми і розробка інформаційної системи з просування реклами у месенджері з використанням Telegram-бота та клієнта для збору інформації.

Наукова новизна роботи полягає у тому, що з’явилась можливість купівлі та продажу реклами з найвигіднішими умовами та максимальною ефективністю у найкоротші терміни.

Практична цінність роботи – розроблення методів та засобів для просування реклами у месенджері.

Розділ 1 Стан проблемної області

1.1. Реклама у Telegram. Роль в сучасному світі та основні аспекти торгівлі

Рекла́ма (лат. reclamare — «гукати раз-у-раз, знову викликати, повторно вигукувати»), забуте вихвала — популяризація товарів, видовищ, послуг і т. ін. з метою привернути увагу покупців, споживачів, глядачів, замовників тощо, поширення інформації про когось, щось для створення популярності, а також візуальна та інша медіа-продукція, що використовуються як засіб привертання уваги потенційних споживачів.

Одиницею реклами у месенджері Telegram є рекламний пост – публікація на каналі, яка зазвичай складається з медіа-контенту (від 1 до 5 елементів: GIF, статичне зображення, відео, документ), текст-поста (макс. кількість символів – 4096; форматування: жирний, курсив, закреслення, підкреслення), кнопки (клікабельні посилання, розташування у форматі ряд-стовпець до 3 кнопок в ряд). Також рекламні пости публікуються з певним таймінгом, що позначається двома числами через слеш (напр. 1/24, 2/48, 12/12 і тд): перше число – кількість годин в топі (коли реклама в каналі – останній пост і перше, що бачить зацікавлений підписник, зайшовши на канал), загальна кількість годин від часу публікації до видалення.

Також важливими параметрами каналу, які впливають на вартість реклами крім таймінгу є такі поняття, як охоплення (кількість переглядів) та ER (engagement rate – коефіцієнт зацікавленості, відношення охоплень за 24 години до загальної кількості підписників, чим вище, тим краще).

Провівши опитування деяких адміністраторів та менеджерів рекламних каналів у Telegram була з’язована цінність реклами у розмірі 50-500 грн за 1000 переглядів. Ця вартість безпосередньо залежить від тематики каналу. Наприклад у каналів на ігрову тематику або лайфхаки ціна за рекламу нижча за інші, оскільки аудиторія таких каналів налічує в основному дітей та підлітків, які, в свою чергу, є неплатоспроможними. На противагу їм, на каналах про заробіток та віддалену роботу кожен перегляд є на порядок цінніший, а отже і заробляють власники таких каналів у рази більше.

Було розглянуто аналоги сервісів для купівлі-продажу реклами у месенджері, яких є надзвичайно мало, і функціонують вони зовсім недавно.

1.2. Сервіси – аналоги

На рисунку 1.2.1 зображено сторінку каталогу Telegram-каналів сервісу Telega.in.

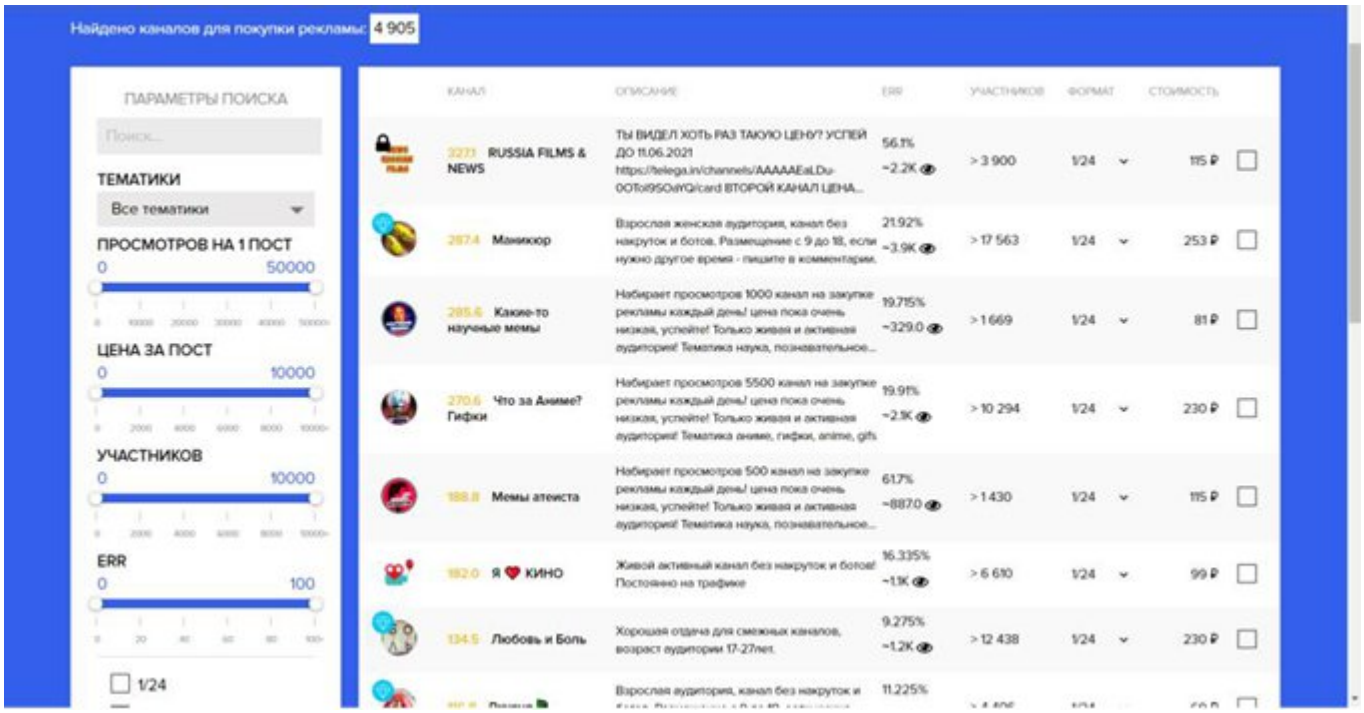

Рисунок 1.2.1. Сервіс-аналог «Telega.in»

На рисунку 1.2.2. сторінка блогів ще одного сервісу-аналогу Epicstars.com

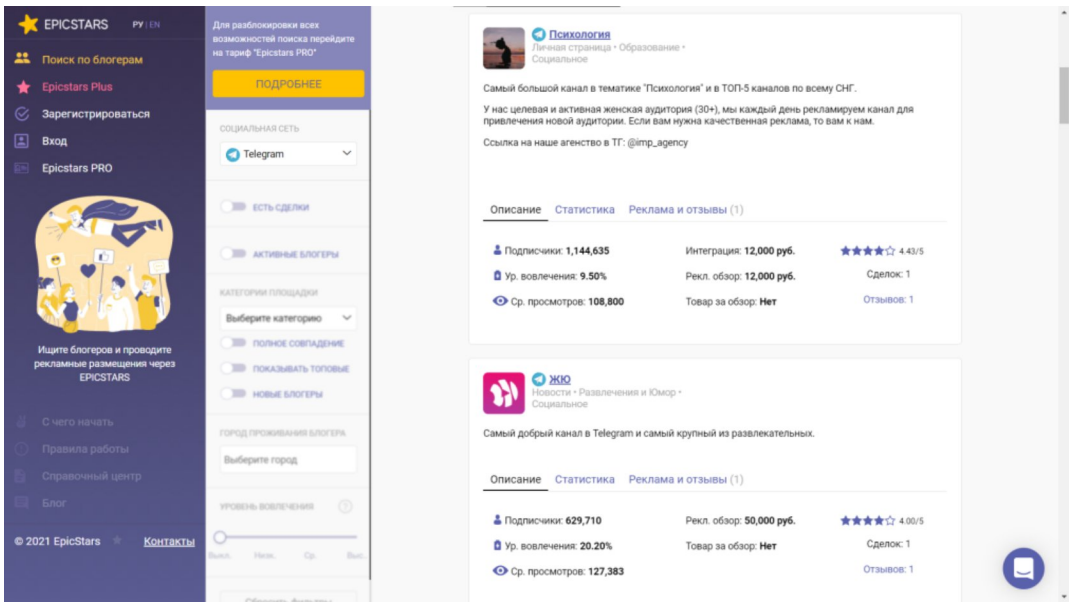

Рисунок 1.2.2 Сервіс-аналог «Epicstars.com»

У вищеназваних сервісах було виявлено наступні недоліки:

- при додаванні нового канала у БД адміністратору необхідно чекати три дні, поки канал пройде ручну модерацію;
- біржа має обмеження в мінімум 500 підписників на каналі;

- в аналітиці каналу немає кількості переглядів за тиждень, динаміки приросту підписників;
- сервіс немає функціоналу для подачі рекламних постів, що давало б можливість власникам каналів для пошуку і подачі заявок на рекламу;
- немає можливості для відкладеного постингу реклами та додавання каналу за допомогою бота;
- немає автоматизованої перевірки виконання умов розміщення реклами.

Розділ 2 Інформаційне та математичне забезпечення

2.1 Логічна структура системи

Система «Біржа реклами» призначена для того, щоб допомогти рекламодавцям та продавцям реклами у Telegram в автоматизації їхньої роботи. Зареєструвавшись у сервісі через Telegram-акаунт або ж номер телефону людина стає користувачем з безкоштовною підпискою за замовчуванням із можливістю переходу на платний пакет. Система пропонує розміщення постів про купівлю-продажу рекламних постів, а також відправлення у відповідь на них заявок. Крім цього є можливість для купівлі-продажу Telegram-каналів із гарантом. Користувачі можуть оформити платну підписку для того, щоб мати необмежену кількість постів та заявок.

Основним призначенням системи є автоматизація купівлі-продажу рекламних постів та Telegram-каналів зі зручним пошуком на фільтром, аналіз Telegram-каналів та автоматичне створення відкладених постів.

Також ця система може бути використана для відслідковування розвитку каналів за наступними параметрами: приріст і фактичне число підписників за день/тиждень, охоплень тощо.

Система складається з чотирьох основних підсистем:

- купівля-продаж реклами;
- купівля-продаж каналів;
- визначення категорій і повноважень користувачів;
- модерація БД.

2.1.1. Підсистема купівлі – продажу реклами

На рисунку 2.1.1 зображена модульна структура підсистеми купівлі-продажу реклами.

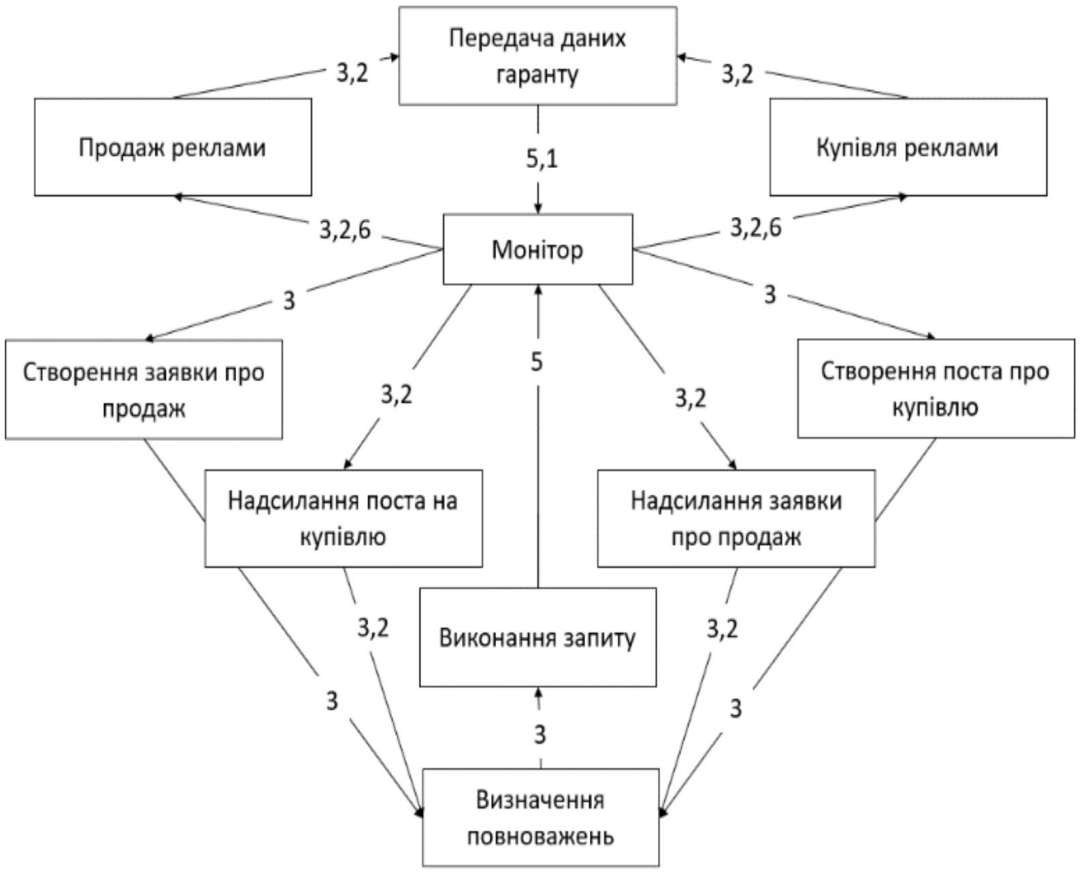

Рисунок 2.1.1 Модульна структура підсистеми купівлі-продажу реклами (1 - відкладений пост, звіт; 2 - дані поточного користувача; 3 - дані поста про купівлю, дані заявки про продаж; 4 - дані заявки про продаж; 5 - результат запиту; 6 - результат грошового переказу)

2.1.2. Підсистема купівлі – продажу каналів

Ця підсистема є частиною системи «Біржа реклами», що відповідає за процесом додавання каналів у сервіс через Telegram або вручну, зберігання інформації, перевірка правильності введення даних, обробки запитів про купівлю та продаж через гаранта, що приймає грошовий переказ та дані для входу на канал в якості адміністратора. Після перевірки гарант проводить обмін.

Вхідними даними підсистеми є: Telegram-канал, вартість, грошовий переказ. Підсистема видає оброблені дані: створений канал, результат купівлі-продажу.

Підсистема «Купівлі-продажу реклами» може функціонувати окремо від всієї системи і включає такі функції:

- введення, виведення й редагування інформації від інформаційних об'єктів підсистеми;
- збереження інформації, що надійшла від підсистеми;
- перевірку адміністратора Telegram-каналу;
- перевірку правильності введення даних;
- перевірку виконання грошового переказу;
- перевірку правильних даних, що надійшли до гаранта;
- передачу даних продавцеві та покупцеві.

Модульна структура підсистеми показана на рисунку 2.1.2.1.

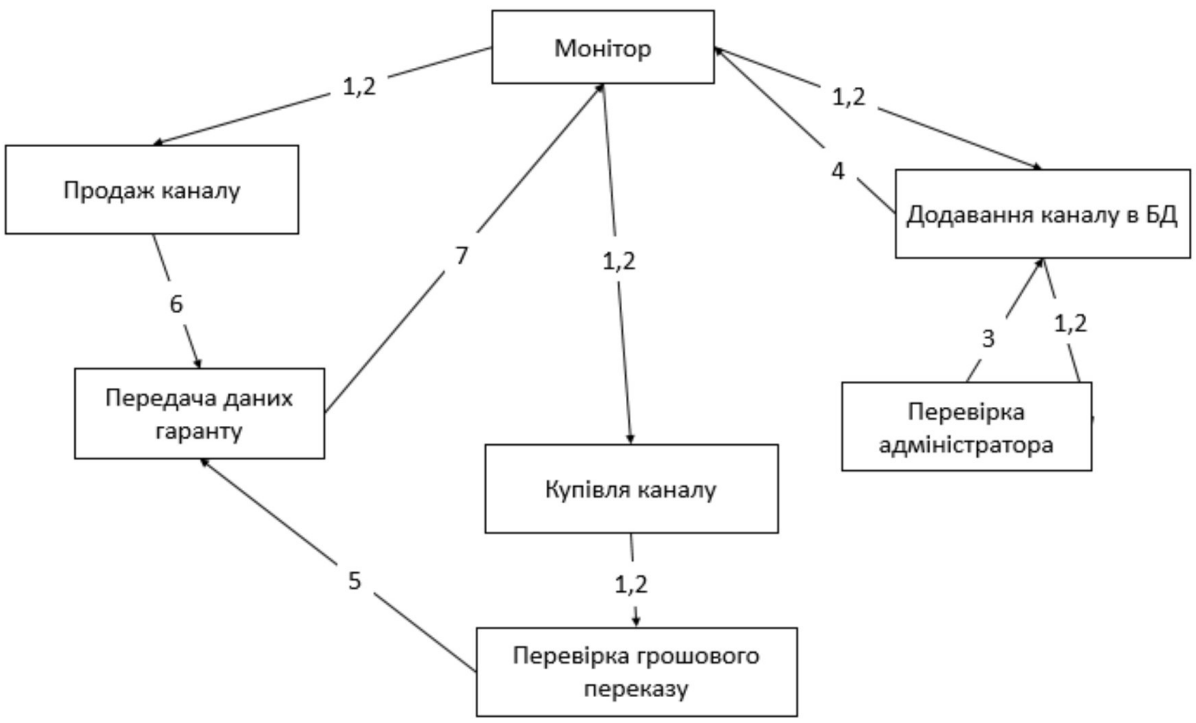

Рисунок 2.1.2.1 Модульна структура підсистеми купівлі-продажу каналів (1 - дані поточного користувача; 2 - дані каналу; 3 - результат перевірки адміністратора; 4 - дані про доданий канал, що генерує бот; 5 - результат грошового переказу; 6 - дані для входу на канал; 7 - результат запиту, звіт)

Робота модулів підсистеми. Монітор – викликає модуль введення й одержує від нього дані для виконання подальших завдань; звертається до модуля додавання каналу в базу даних, передаючи йому вихідні дані; після додавання каналу він передає модулю перевірки адміністратора дані адміністратора, після перевірки модуль відправляє результат попередньому модулю додавання, який, в свою чергу, за

допомогою бота парсить всі необхідні дані для аналізу каналу і повертає створений підтверджений канал монітору, що виводить результат на дисплей.

Купівля каналу – одержує запит від монітора, передає дані на перевірку грошового переказу, який, в свою чергу після виконання свого завдання в позитивному результаті передає дані про грошовий переказ гаранту.

Продаж каналу – одержує запит від монітора, передає дані для входу на канал гаранту.

Гарант після одержання результатів від продажу та купівлі каналу робить перевірку достовірності даних та передає результат своїх дій монітору, що виводить на дисплей результат запиту та звіт.

2.1.3. Підсистема обробки запитів та визначення категорій користувачів

Ця підсистема призначена для визначення категорії, повноважень і обробки запитів користувачів служби зайнятості. Зокрема, вона виконує такі функції:

- реєстрацію нових користувачів;
- визначення прав доступу зареєстрованого користувача;
- обробку запитів;
- прийом реєстраційних даних від модераторів та користувачів;
- прийом реєстраційних даних від адміністратора;
- запис даних у таблицю користувачів;
- запис даних у таблиці постів та заявок;
- запис даних у таблицю категорій каналів.
- Відповідно до виконуваних функцій система працює з такими даними:
- реєстраційними даними користувачів;
- реєстраційними даними модераторів;
- персональними даними користувачів;
- ідентифікаційними даними користувачів;
- запитом;
- даними про категорії каналів.

Модульна структура підсистеми показана на рисунку 2.1.3.1.

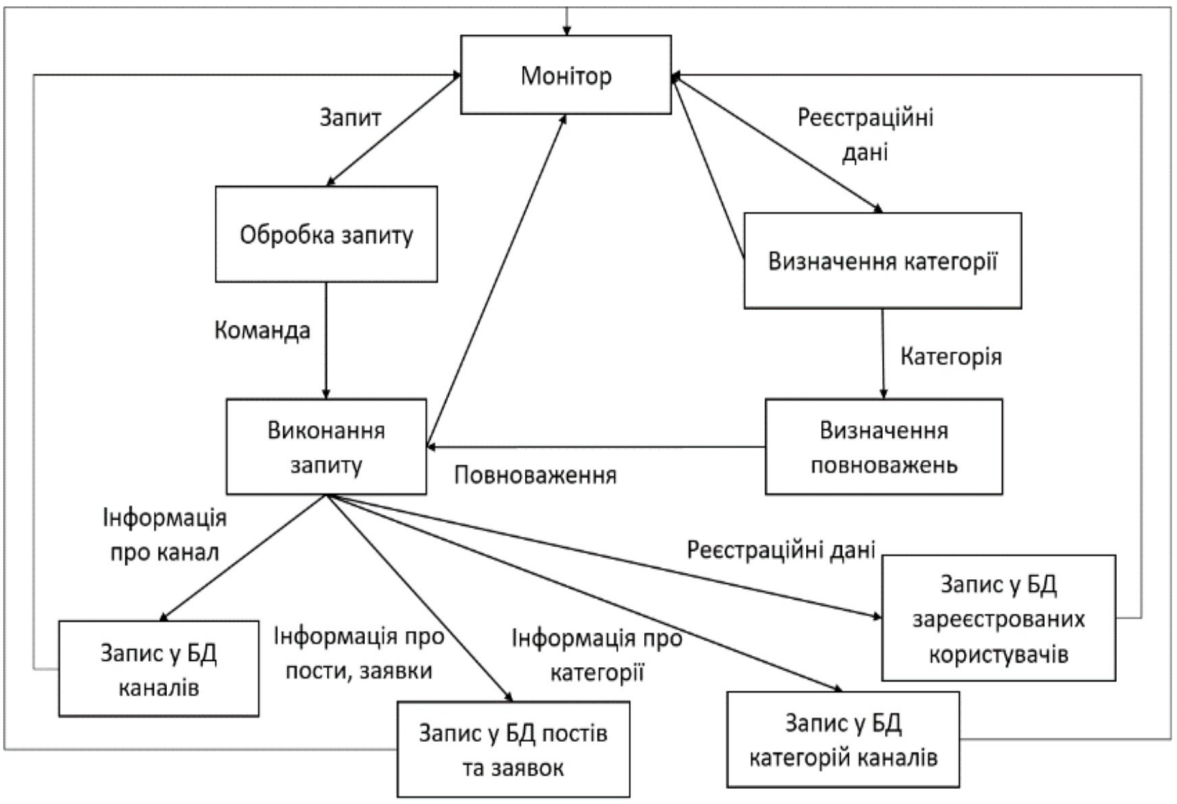

Рисунок 2.1.3.1 Модульна структура підсистеми обробки запитів та визначення категорій користувачів

Визначення категорії – модуль, що визначає категорію користувача. Визначення повноважень – модуль, що визначає повноваження користувача. Обробка запиту - модуль, призначений для обробки запитів користувача. Виконання запиту – модуль, призначений для виконання запитів користувача. Запис у БД зареєстрованих користувачів – модуль, призначений для роботи з БД зареєстрованих користувачів. Запис у БД каналів – модуль, призначений для роботи із таблицею каналів. Запис у БД постів та заявок – модуль, призначений для роботи із таблицями постів та заявок. Запис у БД категорій каналів – модуль, призначений для роботи з категоріями каналів.

2.1.4. Підсистема модерації БД

Ця підсистема призначена для внесення змін в таблиці постів, заявок, каналів. Вона подає інформацію про сутності, а також дозволяє вносити корективи. Звичайний користувач може за її допомогою вносити зміни до власних записів, а модератор - до будь-яких. Модульна структура підсистеми показана на рисунку 2.1.4.1.

Рисунок 2.1.4.1 Підсистема модерації БД

Визначення статусу клієнта.Клієнт входить у підсистему з ідентифікаційним номером. За номером привласнюється рівень доступу. Підсистема видає привілегії, що відповідають рівню доступу.

Модерація БД. *Модератор:* надсилає запит на надання інформації про пости, заявки та канали. При знаходженні невідповідності сутностей правилам публікування в сервісі, моє змогу їх редагувати та видаляти при необхідності.

Користувач: надсилає запит на надання інформації про власні пости, заявки та канали. При необхідності має змогу вносити в них зміни, які в подальшому будуть проходити перевірку модератора.

Згортка: модуль, що одержує з тимчасової БД змінені сутності, обробляє їх і передає оброблені дані в загальну БД.

2.2. Методологія розробки ПЗ

Було вирішено розробляти програмний продукт з підходом DevOps, який пов’язує людські ресурси, процеси та технології для отримання найкращих

результатів. Він поділяється на такі етапи, як: планування і відстеження, розробка, збірка та тестування, доставка, моніторинг, експлуатація. Основна перевага в тому, що розробка, IT-операції, забезпечення якості і безпеки працюють разом для ефективності виконання завдань, пов’язаних в свою чергу із запуском продукта, підготовкою випусків та оновлень. DevOps фокусується на максимальному підвищенні якості, ефективності і розширеному застосуванні автоматизації. Саме ці якості було визначено необхідними для проектування та розробки інформаційної системи з просування реклами для досягнення найкращих результатів та виконання всіх поставлених задач.

Реалізація DevOps полягає в безперервній інтеграції, представленні та розгортанні.

В якості засобу DevOps було взято сервіс Azure DevOps, що дозволяє планувати, відслідковувати та обговорювати роботу у випадку декількох учасників команди. Крім цього за допомогою даного сервісу можна персоналізувати роботу відовідно до особливостей робочих процесів.

На рисунку 2.2.1 зображено дошку завдань у сервічі Azure DevOps для відстеження прогресу виконання задач та якісного планування процесів розробки.

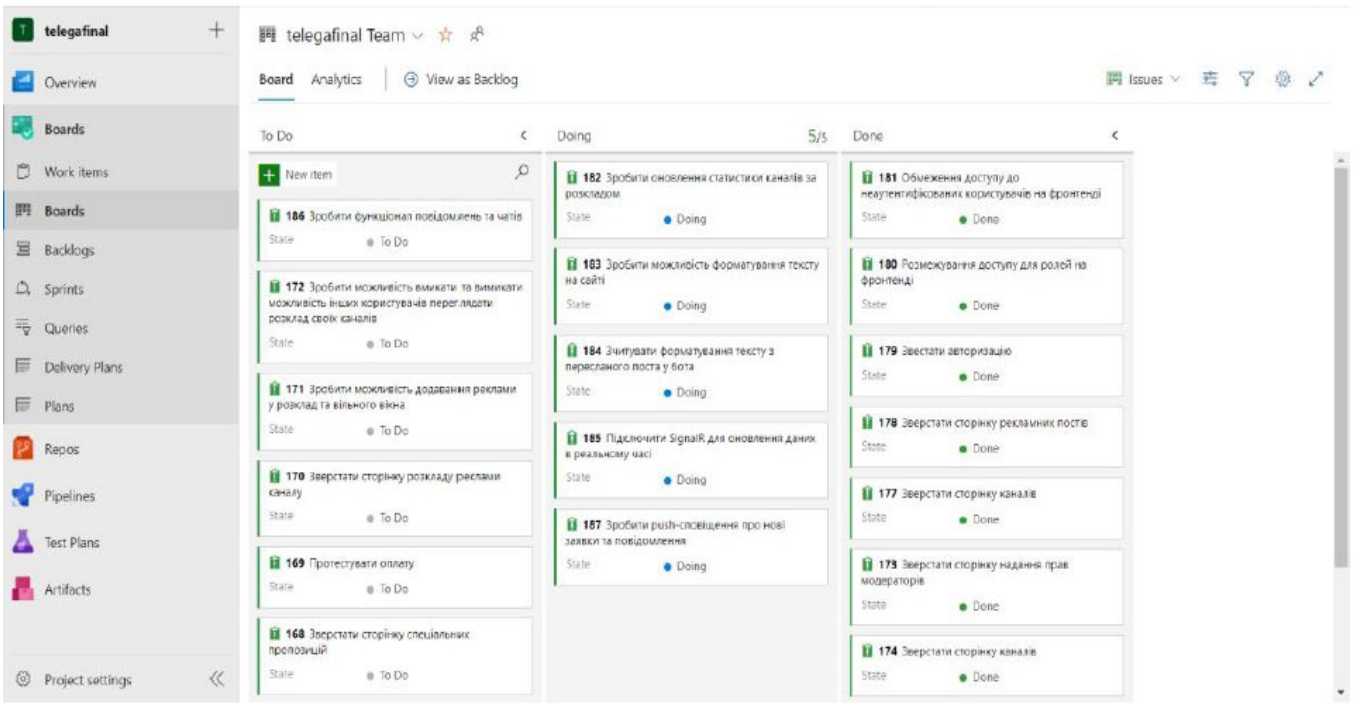

Рисунок 2.2.1 Дошка завдань в Azure DevOps

2.3. User Stories

Одним з найбільш зручних засобів проектування функціональних можливостей є User Stories.

User Stories – це коротке формулювання намірів, що описує щось, що система повинна робити для користувача.

Нижче представлено User Stories для **рекламодавця**:

- Як рекламодавець, я можу переглянути всі канали і зв'язатися з адміністратором відповідного, щоб домовитися про купівлю реклами.
- Як рекламодавець, я можу подивитися оголошення про вільні рекламні місці на певному каналі і подати свою заявку, щоб розмістити свою рекламу.
- Як рекламодавець, я можу створити запис про купівлю реклами і вибрати серед заявок найбільш прийнятну, щоб розмістити там свою рекламу.

До User Stories зі сторони **продавця реклами** відносяться наступні:

- Як продавець реклами, я можу додати свій канал в базу, щоб рекламодавці змогли знайти мене і домовитися про купівлю реклами.
- Як продавець реклами, я можу створити повідомлення про вільномі рекламні місця в моєму каналі, щоб знайти рекламодавця і продати рекламу.
- Як продавець реклами, я можу переглянути повідомлення про купівлю реклами та подати свою заявку, щоб домовитися про продаж реклами.

User Stories **адміністратора**:

- Як адміністратор, я можу управляти модераторами, щоб організувати сумлінну експлуатацію сайту користувачами
- Як адміністратор, я можу банити користувачів за порушення правил, щоб домогтися сумлінної експлуатації сайту користувачами

Модератор має наступні User Stories:

- Як модератор, я можу банити канали, пости і заявки, які порушують правила, щоб домогтися сумлінної експлуатації сайту користувачами.

2.4. Проектування бази даних

Предметна область бази даних – **платформа реклам**. Основними об’єктами бази даних є наступні:

1. **Користувач**: унікальний ідентифікатор, пароль, токен, унікальний ідентифікатор підписки, унікальний ідентифікатор банківських даних, ролі, кількість грошей на рахунку, статус блокування.
2. **Канал**: унікальний ідентифікатор, назва, посилання на аватар, кількість підписників, охоплень за день, охоплень за тиждень, збільшення охоплень за тиждень, за день, відсоток ER, ціна за рекламу, ціна для продажу каналу, посилання на канал, опис, контакт менеджера, статус підтвердження, дата та час останнього оновлення інформації, код для підтвердження, унікальний ідентифікатор адміністратора.
3. **Категорія каналу**: унікальний ідентифікатор, назва категорії.
4. **Чат**: унікальний ідентифікатор, повідомлення користувачів.
5. **Повідомлення**: унікальний ідентифікатор, нікнейм відправника, дата та час.
6. **Пост**: унікальний ідентифікатор, ціна, текст поста, матеріали (фото та відео), мінімальні дата та час, максимальні дата та час, унікальний ідентифікатор користувача, що створив, дата та час створення поста, унікальний ідентифікатор каналу, статус.
7. **Заявка**: унікальний ідентифікатор, унікальний ідентифікатор користувача, унікальний ідентифікатор каналу, ціна, дата та час створення, дата та час публікації, коментар, статус.
8. **Профіль**: унікальний ідентифікатор, код відновлення паролю, нікнейм, посилання на аватар, унікальний ідентифікатор користувача.
9. **Підписка**: унікальний ідентифікатор, назва, доступ до аналітики, кількість доступних постів на день/місяць, кількість доступних заявок на день/місяць.
10. **Угода**: унікальний ідентифікатор, сума, користувач-отримувач, користувач-відправник.
11. **Банківські дані**: унікальний ідентифікатор, ім’я власника картки, номер картки, унікальний ідентифікатор користувача, CVV, строк користування.

В межах предметної області відносяться такі зовнішні вхідні потоки:

1. Інформація, що надходить безпосередньо з Telegram до біржі (кількість підписників, переглядів на публікаціях, текст публікацій, ER) – збір даних через Telegram API та Telegram-бота.
2. Інформація від користувачів про канал, рекламні пости.

Зовнішні вихідні потоки:

1. Публікація постів у Telegram-каналах після здійснення угоди між рекламодавцем та рекламістом.
2. Аналітика зареєстрованих у біржі Telegram-каналів для користувачів.
3. Результати угод користувачів.

Внутрішні потоки:

1. Вхідний у відділ адміністрації (вихідний з відділу модерації) - фінансова інформація, звіти про порушення правил.
2. Вхідний у відділ модерації (вихідний з відділу адміністрації) – новий склад модераторів.
3. Вхідний у відділ модерації (вихідний з відділу купівлі та продажу реклами та каналів) – актуальні списки рекламних постів та заявок.

До користувачів БД відносяться наступні:

1. Адміністратор. До його інформаційних потреб відносяться: список модераторів та користувачів, їх дані.
2. Модератор. Потреби: список даних користувачів, список даних рекламних постів, заявок на рекламу.
3. Користувач. Потреби: список рекламних постів, заявок на рекламу, каналів, доступних для продажу та актуальних цін на рекламу в них.

Схема спроектованої бази даних наведена на рисунку 2.4.1.

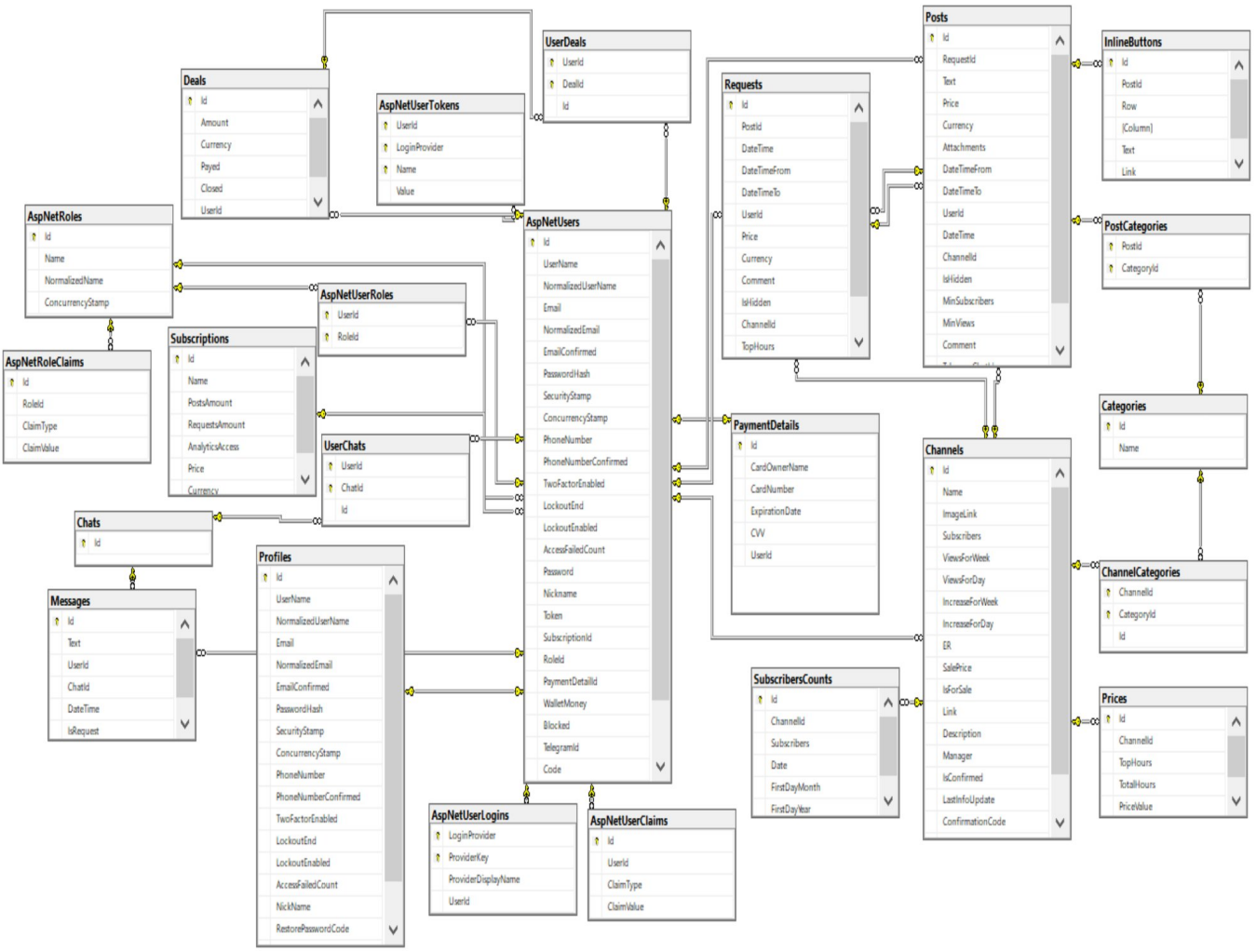

Рисунок 2.4.1 Схема БД до програмного продукту

2.5. Архітектура вебзастосунку

Для даного проекту було використано одну з найбільш використовуваних архітектур, класична трирівнева система, яка передбачає поділ додатка на три рівні.

Класична трирівнева система складається з наступних рівнів: Presentation Layer, Business Layer, Data Access Layer (рис. 2.5).

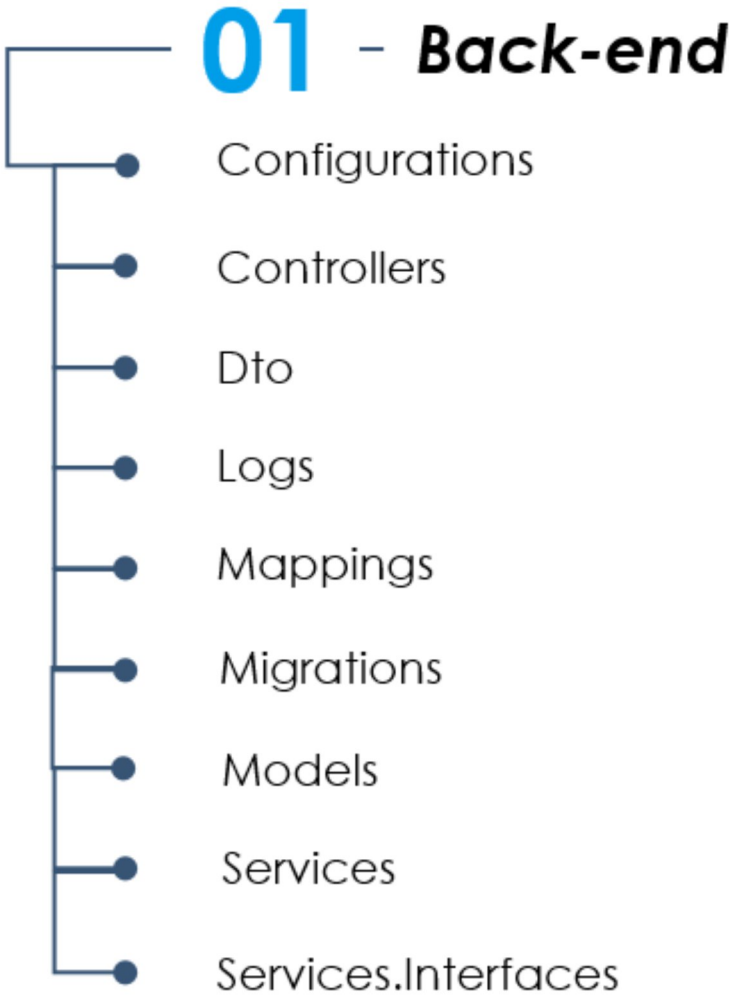

Рисунок 2.5 Архітектура API

Presentation layer (рівень представлення): це той рівень, з яким безпосередньо взаємодіє користувач. Цей рівень включає компоненти для користувацького інтерфейсу, механізм отримання введення від користувача. Стосовно до asp.net mvc на даному рівні розташовані уявлення і всі ті компоненти, які складають призначений для користувача інтерфейс (стилі, статичні сторінки html, javascript), а також моделі уявлень, контролери, об'єкти контексту запиту.

Business layer (рівень бізнес-логіки): містить набір компонентів, які відповідають за обробку отриманих від рівня уявлень даних, реалізує всю необхідну логіку додатка, все обчислення, взаємодіє з базою даних і передає рівнем подання результат обробки.

Data Access layer (рівень доступу до даних): зберігає моделі, що описують використовувані суті, також тут розміщуються специфічні класи для роботи з різними технологіями доступу до даних, наприклад, клас контексту даних Entity Framework.

Тут також зберігаються репозиторії, через які рівень бізнес-логіки взаємодіє з базою даних.

2.6. Стек технологій

При розробці даного програмного продукту було використано декілька стектехнологій, однією з яких є мова **C#**. Вибір мови для даного проекту зупинився саме на цій, оскільки є безліч переваг роботи на мові C#. До прикладу, в ній немає ніяких обмежень у наслідуванні, модифікації та універсалізації створеного програмного продукту, код, написаний на C# є простим, зручним та надійним. До того ж, великою перевагою цієї мови є підтримка ООП, шаблонів проектування та можливістю розробляти програми на C# для практично всіх існуючих найбільш популярних операційних систем.

Крім цього для розробки серверної частини було використано технологію **ASP.NET** — технологія створення веб-застосунків і веб-сервісів від компанії Майкрософт. Вона є складовою частиною платформи Microsoft.NET і розвитком старішої технології Microsoft ASP. На цей час останньою версією цієї технології є ASP.NET Core 2.0

ASP.NET зовні багато в чому зберігає схожість із старішою технологією ASP, що дозволяє розробникам відносно легко перейти на ASP.NET. У той же час внутрішній устрій ASP.NET істотно відрізняється від ASP, оскільки вона заснована на платформі. NET і, отже, використовує всі нові можливості, що надаються цією платформою.

Для розробки бази даних було виристано **Entity Framework Core Code First**, що являє собою об'єктно-орієнтовану, легковажну і розширюючу технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти). Тобто EF Core дозволяє працювати базами даних, але є більш високий рівень абстракції: EF Core дозволяє абстрагуватися від самої бази даних і її таблиць і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами,

первинними і зовнішніми ключами, то на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами.

Entity Framework Core підтримує безліч різних систем баз даних. Таким чином, ми можемо через EF Core працювати з будь-якої СУБД, якщо для неї є потрібний провайдер.

В якості системи контролю версій було використано **Git**, яка є незамінною для використання і надає нам такі корисні функції, як зберігання історії змін з централізованому сховищі, копіювання об’єктів з розгалуженням історії та ін.

Для тестування API було використано специфікацію OpenAPI, початково відому як **Swagger**. Swagger – це специфікація машиночитабельних файлів з інтерфейсами, для опису, створення, використання і візуалізації REST веб сервісів. Існують різноманітні інструменти що можуть генерувати код, документацію і тести за файлом з описом інтерфейсу. За розробкою специфікації OpenAPI (OAS) наглядає Open API Initiative, проект Linux Foundation.

Для розробки користувацького інтерфейсу було використано **AngularJS**, що є JavaScript-фреймворк з відкритим програмним кодом, який розробляє Google. Призначений для розробки односторінкових додатків, що складаються з одної HTML сторінки з CSS і JavaScript. Його мета — розширення браузерних застосунків на основі шаблону Модель-вид-контролер (MVC), а також спрощення їх тестування та розробки.

Фреймворк працює зі сторінкою HTML, що містить додаткові атрибути і пов'язує області вводу або виводу сторінки з моделлю, яка є звичайними змінними JavaScript. Значення цих змінних задаються вручну або отримуються зі статичних або динамічних JSON-даних.

Для розробки Telegram-бота обрано **Python**, мову програмування з динамічною типізацією, що була використана для проектування та розробки Telegram-бота, що виконує функції реєстрації, авторизації, збору даних про канал, формування статистики, роботи з відкладеними постами та системою сповіщень. Перевагою розробки бота на даній мові програмування є безмежність можливостей розробки, економія часу та гнучкість спроектованого проекту. В рамках Python було

використано такі основні бібліотеки для роботи з Telegram API, як: telethon та pyTelegramBotApi.

2.7 Бінарні дерева

Бінарне дерево – це структура даних, кожен елемент якої окрім самих даних містить покажчики на два наступних елементи структури. Один з цих наступних елементів умовно називається лівим, а інший правим. Кожен елемент дерева називається вузлом або листом дерева. Перший вузол дерева (з якого дерево власне починається) називається коренем. Фрагмент дерева разом з вузлом, від якого він починається, називається піддеревом або віткою. Множина всіх вузлів, рівновіддалених від кореня, називається рівнем. Вузол, з якого не починається жодна вітка, називається кінцевим або термінальним вузлом. Оскільки дерево бінарне, кожен вузол може породжувати два вузли наступного рівня. Породжені вузли є дочірніми по відношенню до вузла, що їх породив. Породжуючий вузол є батьківським по відношенню до своїх дочірніх вузлів. Батьківський вузол разом із своїми дочірніми складає ланку. Сумарна кількість рівнів дерева називається висотою дерева.

Основними операціями при роботі з деревами є:

- додавання елемента до дерева;
- пошук елемента дерева, що відповідає заданому критерію пошуку;
- сортування елементів дерева;
- вилучення елемента дерева.

Процес доступу до елементів дерева називається проходженням дерева. Існує три способи проходження дерев: послідовний, низхідний та висхідний.

При послідовному методі доступу до елементів дерева порядок проходження дерева наступний: спочатку розглядається самий лівий відносно кореня елемент, потім його батьківський вузол, потім правий елемент даної ланки, потім переходять до елемента попереднього рівня, що є батьківським по відношенню до батьківського вузла даної ланки і т. д. Вверх до кореня, а потім від кореня вниз до самого правого елемента.

При низхідному проходженні дерева порядок обходу елементів зверху- вниз та зліва-направо. Тобто спочатку проходиться корінь, потім його лівий елемент, а потім правий і т.д.

При висхідному проходженні порядок проходження зліва-направо та знизу-вверх. Тобто спочатку проходиться лівий вузол найнижчої ланки, потім правий вузол цієї ланки, а потім їх батьківський вузол і т.д.

Як бачимо бінарне дерево – досить складна структура даних. Фактично це ускладнений варіант списку. Деякі операції реалізуються значно складніше ніж у списку. Однак операції пошуку у відсортованому бінарному дереві виконуються значно ефективніше ніж у списку, тому бінарні дерева використовуються на практиці досить часто.

2.8. Бінарні дерева пошуку

Бінарне дерево пошуку — це бінарне дерево, яке володіє додатковими властивостями: значення лівого нащадка менше значення батька, а значення правого нащадка більше значення батька для кожного вузла дерева. Тобто, дані в бінарному дереві пошуку зберігаються у відсортованому вигляді. При кожній операції вставки нового або видалення існуючого вузла відсортований порядок дерева зберігається. При пошуку елемента порівнюється шукане значення з коренем. Якщо шукане більше кореня, то пошук продовжується в правому нащадка кореня, якщо менше, то в лівому, якщо одно, то значення знайдено і пошук припиняється.

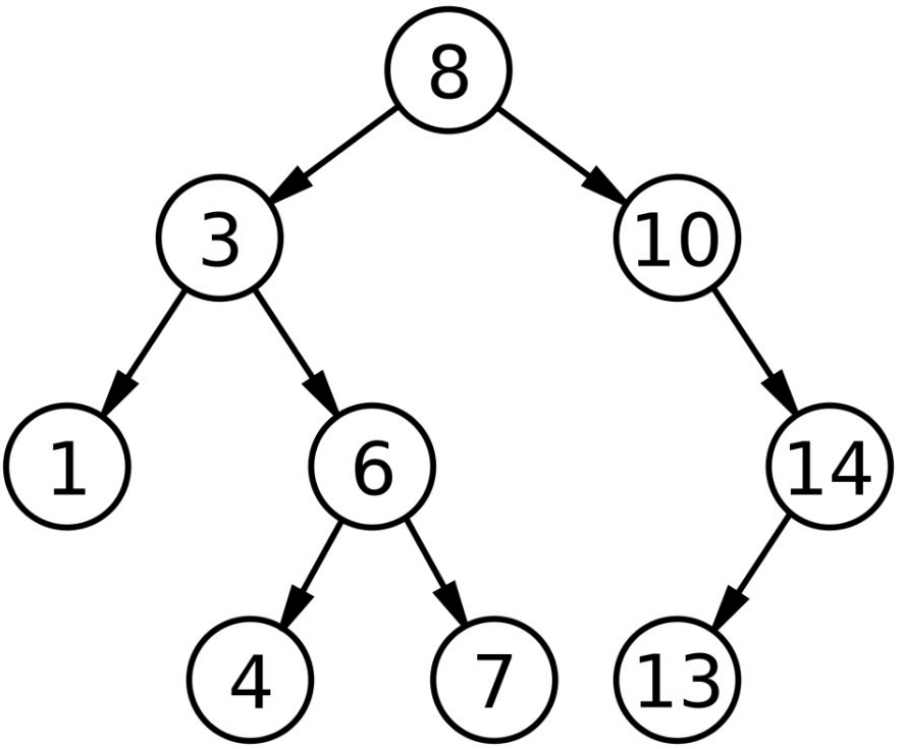

Рисунок 2.8.1. Бінарне дерево пошуку

Збалансоване бінарне дерево пошуку — це бінарне дерево пошуку з логарифмічною заввишки. Дане визначення швидше ідейне, ніж суворе. Суворе визначення оперує різницею глибини самого глибокого і самого неглибокого аркуша (AVL-дерева) чи відношенням глибини самого глибокого і самого неглибокого листа (у червоно-чорних деревах). У збалансованому бінарному дереві пошуку операції пошуку, вставки і видалення виконуються за логарифмічний час (так як шлях до будь-якого листа від кореня не більше логарифма). В виродженому випадку незбалансованого бінарного дерева пошуку, наприклад, коли в порожнє дерево вставлялася відсортована послідовність, дерево перетвориться в лінійний список, і операції пошуку, вставки і видалення будуть виконуватися за лінійний час. Тому балансування дерева вкрай важлива. Технічно балансування здійснюється поворотами частин дерева при вставці нового елемента, якщо вставка елемента порушила умова збалансованості.

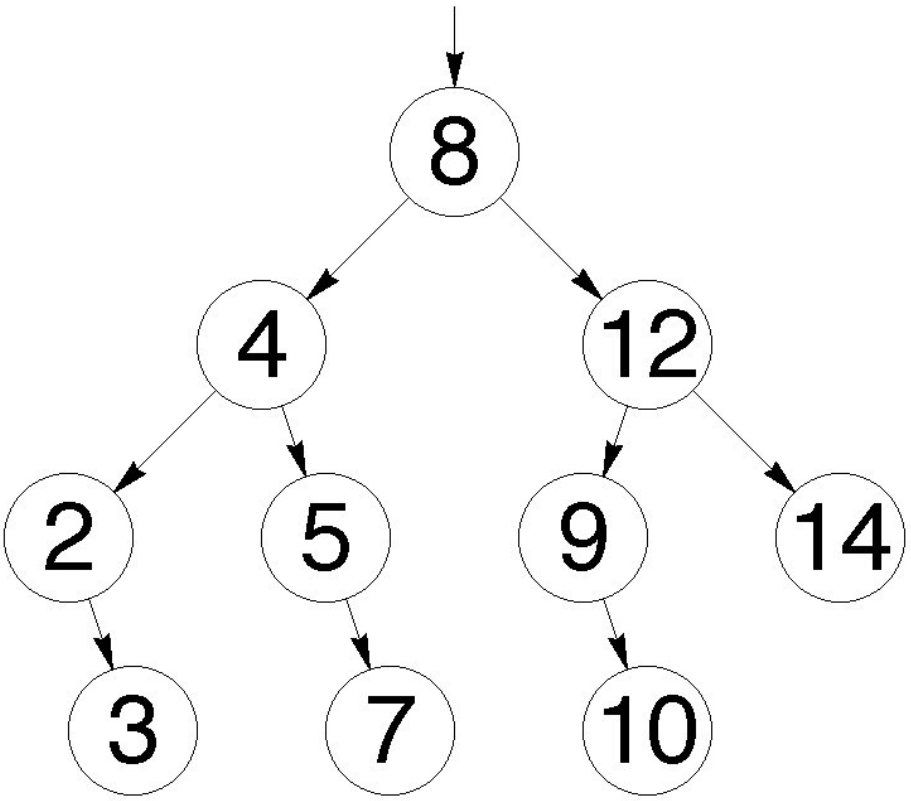

Рисунок 2.8.2. Збалансоване бінарне дерево пошуку

Збалансоване бінарне дерево пошуку застосовується, коли необхідно здійснювати швидкий пошук елементів, що чергується зі вставками нових елементів і вилученнями існуючих. У разі, якщо набір елементів, що зберігається в структурі даних фіксований і немає нових вставок і вилучень, то масив краще. Тому що пошук можна здійснювати алгоритм бінарного пошуку за логарифмічний час, але відсутні додаткові витрати по зберіганню і використанню покажчиків.

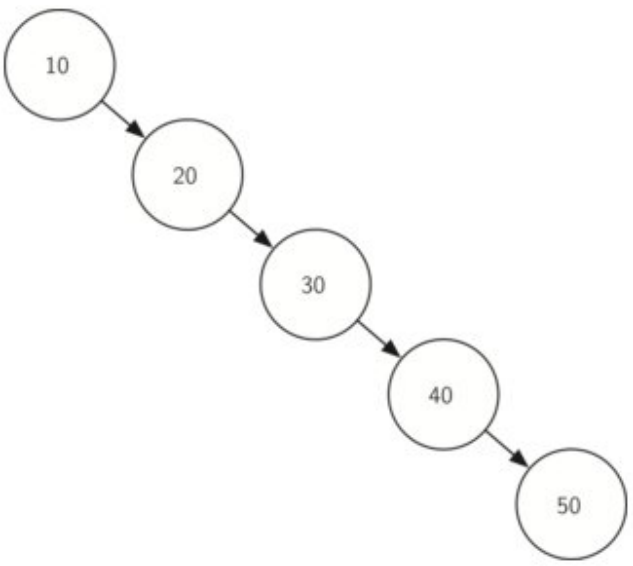

Рисунок 2.8.3. Екстремально збалансоване бінарне дерево пошуку

Отже, бінарне дерево пошуку – це структура даних, яка призначена для того, щоб реалізувати швидкий пошук елементів, які зберігаються в структурі.

Розділ 3 Програмне та технічне забезпечення

3.1. Алгоритми обробки та аналізу даних

3.1.1. Алгоритми додавання нового каналу в базу даних

Процес додавання нового каналу в базу даних починається з того, що користувач, який є власником Telegram-каналу заповнює форму, що складається з посилання на канал, прайс-листа з різними таймінгами, опису та вартості каналу в разі продажу. Після цього виконується запит із додаванням всіх введених даних до БД.

Наступним кроком є додавання до опису каналу згенерованого і доданого в БД коду для підтвердження права власності на канал. Враховуючи те, що в додатку присутні грошові перекази, це принципово важливий крок задля того, щоб запобігти шахрайству.

Далі сервіс пропонує користувачу надіслати Telegram-боту спеціальну команду, а бот, в свою чергу, просить надіслати будь-який пост із доданого каналу.

Коли користувач надає пост, бот надсилає спеціальному Telegram-клієнту повідомлення із посиланням на вступ до каналу, а клієнт автоматично підписується на канал за допомогою посилання, завантажує аватарку каналу і збирає всю необхідну інформацію для формування статистики каналу і заносить її у БД, а саме: кількість переглядів кожного з постів за останній тиждень (групує медіа-контент та текстове повідомлення з одного поста в одне число охоплень, а потім по дням і формує число охоплень за день та за останні 7 днів), кількість підписників, назву.

3.1.2. Алгоритм пошуку постів за фільтром

Було створено ріре для фільтрації постів, що відбувається за такими параметрами, як: дата з, дата по, час з, час по, мінімум підписників, мінімальна ціна, категорії.

Крім цього відбувається фільтрування з вибіркою лише тих постів, які підходять для розміщення хоча б в одному каналі поточному користувача, у відповідь на які користувач може надіслати запит, також є поле для виведення лише постів, що підходять для розміщення конкретного каналу, що присутній в базі даних.

3.1.3. Алгоритм подачі заявки на пост

В таких процесах, як подача заявки та створення нового поста на серверній стороні проводиться перевірка поточної підписки користувача і відмова у разі вичерпання кількості доступних сутностей в рамках підписки.

Кодова імплементація:

```
public Request CreateRequest(Request request)
{
var userId = Guid.Parse(_httpContextAccessor.HttpContext.User.Claims.First(x => x.Type == "UserID").Value);
request.UserId = userId;
var subscriptionId = _requestcontext.Users.FirstOrDefault(x => x.Id == userId).SubscriptionId;
var subscription = _requestcontext.Subscriptions.First(x => x.Id == subscriptionId);
if (GetAmountByUserId(userId) < subscription.RequestsAmount)
{
_requestcontext.Requests.Add(request);
_requestcontext.SaveChanges();
}
return request;
}
```

3.2. Розробка бази даних

Робота створеного програмного продукту грунтується на базі даних (БД), створеній на основі підходу Entity Framework Code-First.

База даних містить 21 таблицю. Їхній перелік, а також властивості стовпців наведені у таблиці 3.2.1.

Таблиця 3.2.1 – Таблиці БД та їх властивості

Назва таблиці	Назва поля	Тип даних	NULL	Primary Key	Foreign Key
_EFMigrationsHistory	MigrationId	nvarchar(150)		+	
	ProductVersion	nvarchar(32)			
AspNetRoleClaims	Id	int		+	
	RoleId	uniqueidentifier			+
	ClaimType	nvarchar(MAX)	+		
	ClaimValue	nvarchar(MAX)	+		
AspNetRoles	Id	uniqueidentifier		+	
	Name	nvarchar(256)	+		
	NormalizedName	nvarchar(256)	+		
	ConcurrencyStamp	nvarchar(MAX)	+		
AspNetUserClaims	Id	int		+	
	UserId	uniqueidentifier			+
	ClaimType	nvarchar(MAX)	+		
	ClaimValue	nvarchar(MAX)	+		
AspNetUserLogins	LoginProvider	nvarchar(450)		+	
	ProviderKey	nvarchar(450)			
	ProviderDisplayName	nvarchar(MAX)	+		
	UserId	uniqueidentifier			+
AspNetUserRoles	UserId	uniqueidentifier		+	
	RoleId	uniqueidentifier			+
AspNetUsers	Id	uniqueidentifier		+	
	UserName	nvarchar(256)	+		
	NormalizedUserName	nvarchar(256)	+		
	Email	nvarchar(256)	+		
	NormalizedEmail	nvarchar(256)	+		
	EmailConfirmed	bit			
	PasswordHash	nvarchar(MAX)	+		
	SequrityStamp	nvarchar(MAX)	+		
	ConcurrencyStamp	nvarchar(MAX)	+		
	PhoneNumber	nvarchar(MAX)	+		
	PhoneNumberConfirmed	Bit			
	TwoFactorEnabled	Bit			
	LockoutEnd	datetimeoffset(7)	+		
	LockoutEnabled	bit			
	AccessFailedCount	int			

	Password	nvarchar(MAX)	+		
	Nickname	nvarchar(MAX)	+		
	Token	nvarchar(MAX)	+		
	SubscriptionId	Uniqueidentifier	+		+
	RoleId	Uniqueidentifier	+		+
	PaymentDetailId	Uniqueidentifier	+		+
	WalletMoney	float			
	Blocked	bit			
	TelegramId	uniqueidentifier			
	Code	nvarchar(MAX)	+		
	CodeGenerationDateTime	datetime2(7)			
AspNetUserTokens	UserId	uniqueidentifier		+	
	LoginProvider	nvarchar(450)			
	Name	nvarchar(450)			
	Value	nvarchar(MAX)	+		
Categories	Id	uniqueidentifier		+	
	Name	nvarchar(30)			
ChannelCategories	Id	Uniqueidentifier		+	
	CategoryId	Uniqueidentifier			+
	ChannelId	Uniqueidentifier			+
Channels	Id	Uniqueidentifier		+	
	Name	nvarchar(40)			
	ImageLink	nvarchar(MAX)	+		
	Subscribers	Int			
	ViewsForWeek	Int			
	ViewsForDay	Int			
	IncreaseForWeek	Int			
	IncreaseForDay	Int			
	Percent	Real			
	AdPrice	Float			
	SalePrice	Float			

	IsForSale	Bit			
	Link	nvarchar(MAX)			
	Description	nvarchar(200)	+		
	Manager	nvarchar(MAX)			
	IsConfirmed	bit			
	LastInfoUpdate	datetime2(7)			
	ConfirmationCode	nvarchar(MAX)			
	UserId	uniqueidentifier			+

Chat	Id	uniqueidentifier		+	
Deals	Id	uniqueidentifier		+	
	Amount	float			
	Currency	Int			
	Payed	Bit			
	Closed	Bit			
	UserId	uniqueidentifier			+
Messages	Id	Uniqueidentifier		+	
	Text	nvarchar(500)			
	UserId	Uniqueidentifier			+
	ChatId	Uniqueidentifier			+
	DateTime	datetime2(7)			
	IsRequest	bit			
PaymentDetails	Id	Uniqueidentifier		+	
	CardOwnerName	nvarchar(100)			
	CardNumber	varchar(16)			
	ExpirationDate	varchar(5)			
	CVV	varchar(3)			
	UserId	uniqueidentifier			+
Posts	Id	Uniqueidentifier		+	
	RequestId	Uniqueidentifier	+		+
	Text	nvarchar(500)			
	Price	Float			
	Currency	Int			
	Attachments	nvarchar(300)			
	DateTimeFrom	datetime2(7)			
	DateTimeTo	datetime2(7)			
	UserId	Uniqueidentifier	+		+
	DateTime	datetime2(7)			
	ChannelId	Uniqueidentifier	+		+
	IsHidden	Bit			

Profiles	Id	uniqueidentifier		+	
	UserName	nvarchar(MAX)	+		
	NormalizedUserName	nvarchar(MAX)	+		
	Email	nvarchar(MAX)	+		
	NormalizedEmail	nvarchar(MAX)	+		
	EmailConfirmed	Bit			
	PasswordHash	nvarchar(MAX)	+		
	SecurityStamp	nvarchar(MAX)	+		
	ConcurrencyStamp	nvarchar(MAX)	+		
	PhoneNumber	nvarchar(MAX)	+		
	PhoneNumberConfirmed	Bit			
	TwoFactorEnabled	Bit			
	LockoutEnd	datetimeoffset(7)	+		
	LockoutEnabled	Bit			
	AccessFailedCount	Int			
	NickName	nvarchar(30)	+		
	RestorePasswordCode	nvarchar(MAX)	+		
	PictureUrl	nvarchar(MAX)	+		
	UserId	uniqueidentifier			+
Requests	Id	uniqueidentifier		+	
	PostId	Uniqueidentifier	+		+
	DateTime	datetime2(7)			
	DateTimeFrom	datetime2(7)			
	DateTimeTo	datetime2(7)			
	UserId	Uniqueidentifier	+		+
	Price	Float			
	Currency	Int			
	Comment	nvarchar(500)	+		
	IsHidden	bit			
	ChannelId	uniqueidentifier			+

Subscriptions	Id	uniqueidentifier		+	
	Name	nvarchar(30)			
	PostsAmount	Int			
	RequestsAmount	Int			
	AnalyticsAccess	Bit			
	Price	float			
	Currency	int			
UserChats	Id	uniqueidentifier		+	
	UserId	Uniqueidentifier			+
	ChatId	Uniqueidentifier			+
UserDeals	Id	Uniqueidentifier		+	
	UserId	Uniqueidentifier			+
	DealId	uniqueidentifier			+

3.3. Розробка інтерфейсу користувача

Інтерфейс користувача було розроблено на Angular з використанням стилів з Material Angular та шаблону Fuse.

Інтерейс адаптований під різні типи пристроїв, а також передбачена тема (рис. 3.3.1).

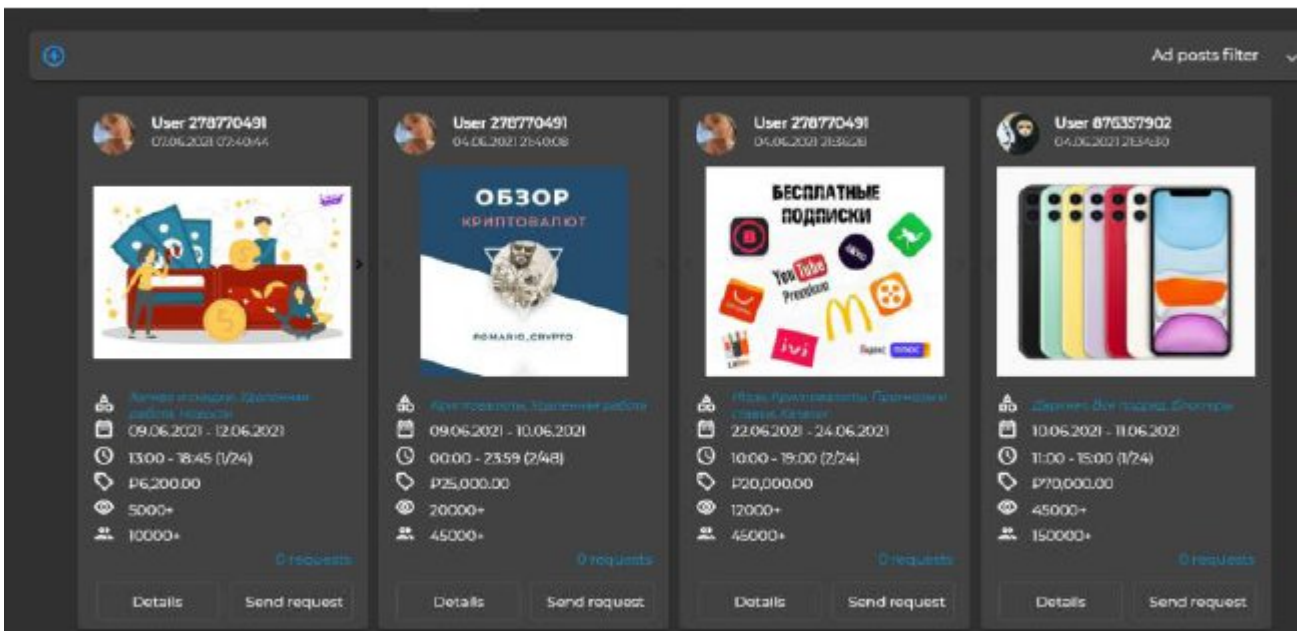

Рисунок 3.3.1 Тема додатку

3.3.1. Меню навігації

Проектуючи користувацький інтерфейс було розроблено меню для зручної навігації, яке користувач зможе згорнути (рис. 3.3.1.1) або розгорнути (рис. 3.3.1.2) при натисканні на кнопку.

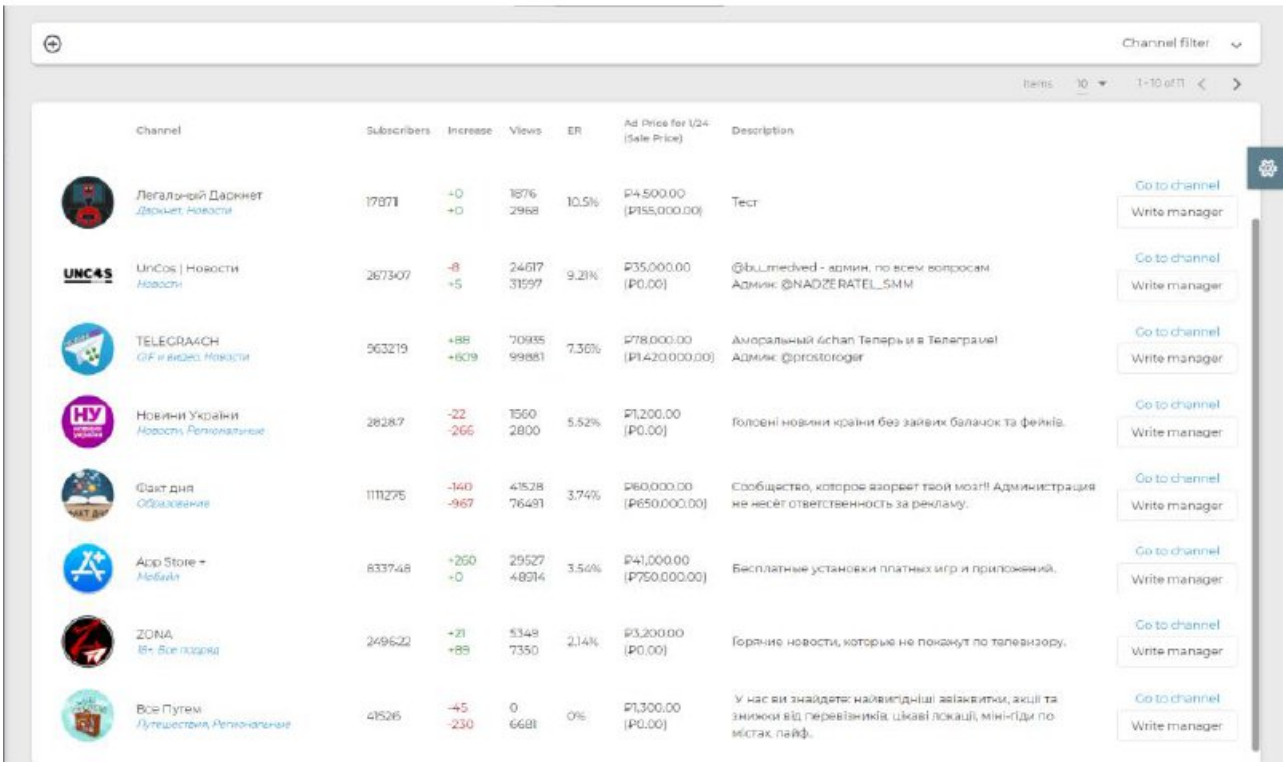

Рисунок 3.3.1.1 Згорнуте меню навігації

Меню містить 2 основні категорії: головне меню (пункти: Профіль, Канали, Рекламні пости) та Налаштування (пункти: Кастомізація, Налаштування профілю та Вихід з акаунту).

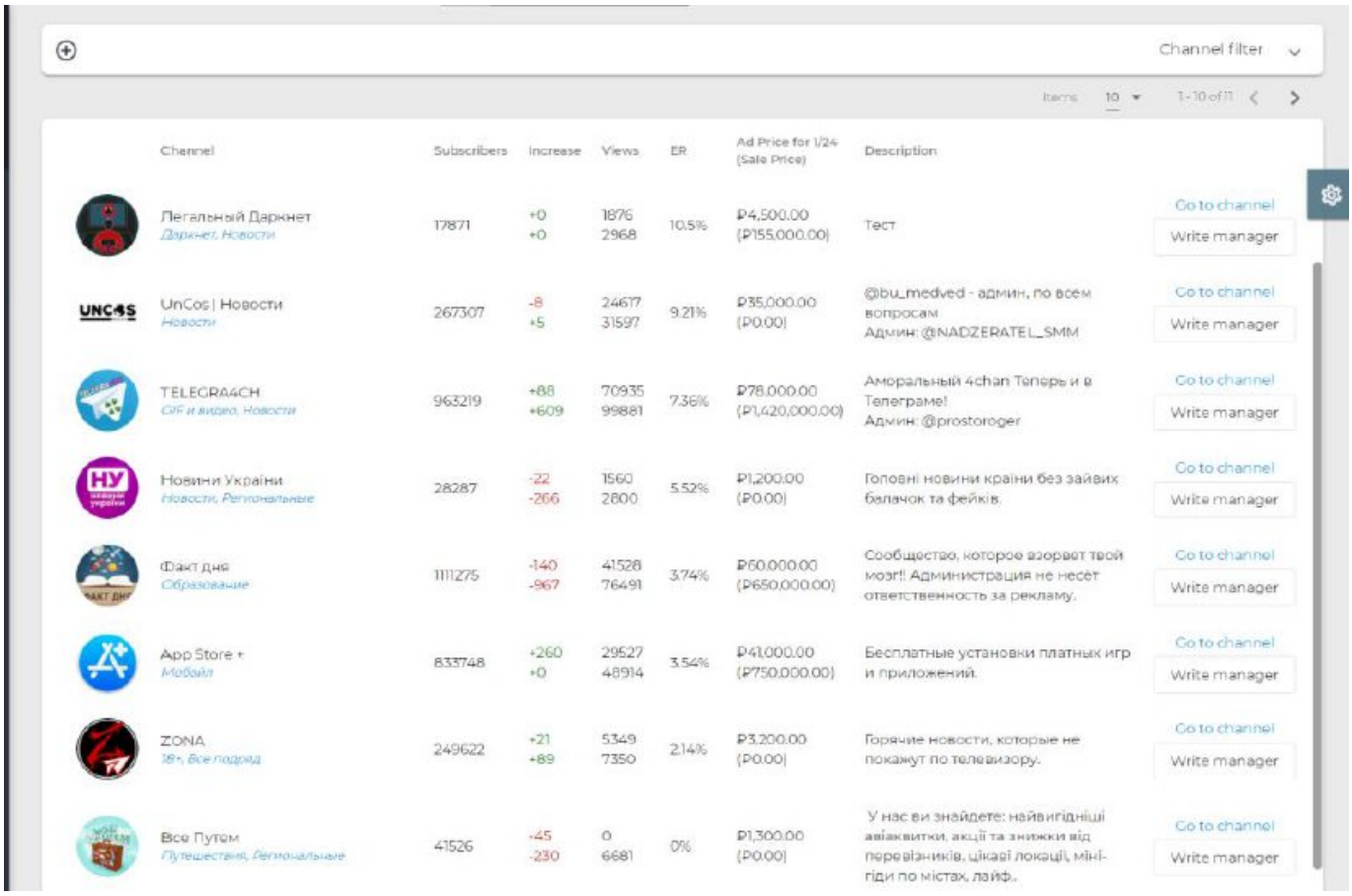

Рисунок 3.3.1.2 Сторінка каналів із розгорнутим меню навігації

3.3.2. Сторінка каналів

Однією з основних сторінок додатку є сторінка каналів (рис. 3.3.2.1). На ній розташований список каналів з головною інформацією про них: аватарка, назва, категорії, кількість підписників, приріст підписників за день та тиждень, середня кількість переглядів на публікаціях за день та тиждень, відношення кількості підписників до кількості переглядів (у відсотках), ціна на рекламну публукацію та на самий канал, опис каналу та кнопки для переходу на канал та для особистих повідомлень менеджеру каналу.

При натисканні на канал розгортається список із цінами на рекламу з відповідними значеннями таймінгу (рис. 3.3.2.2).

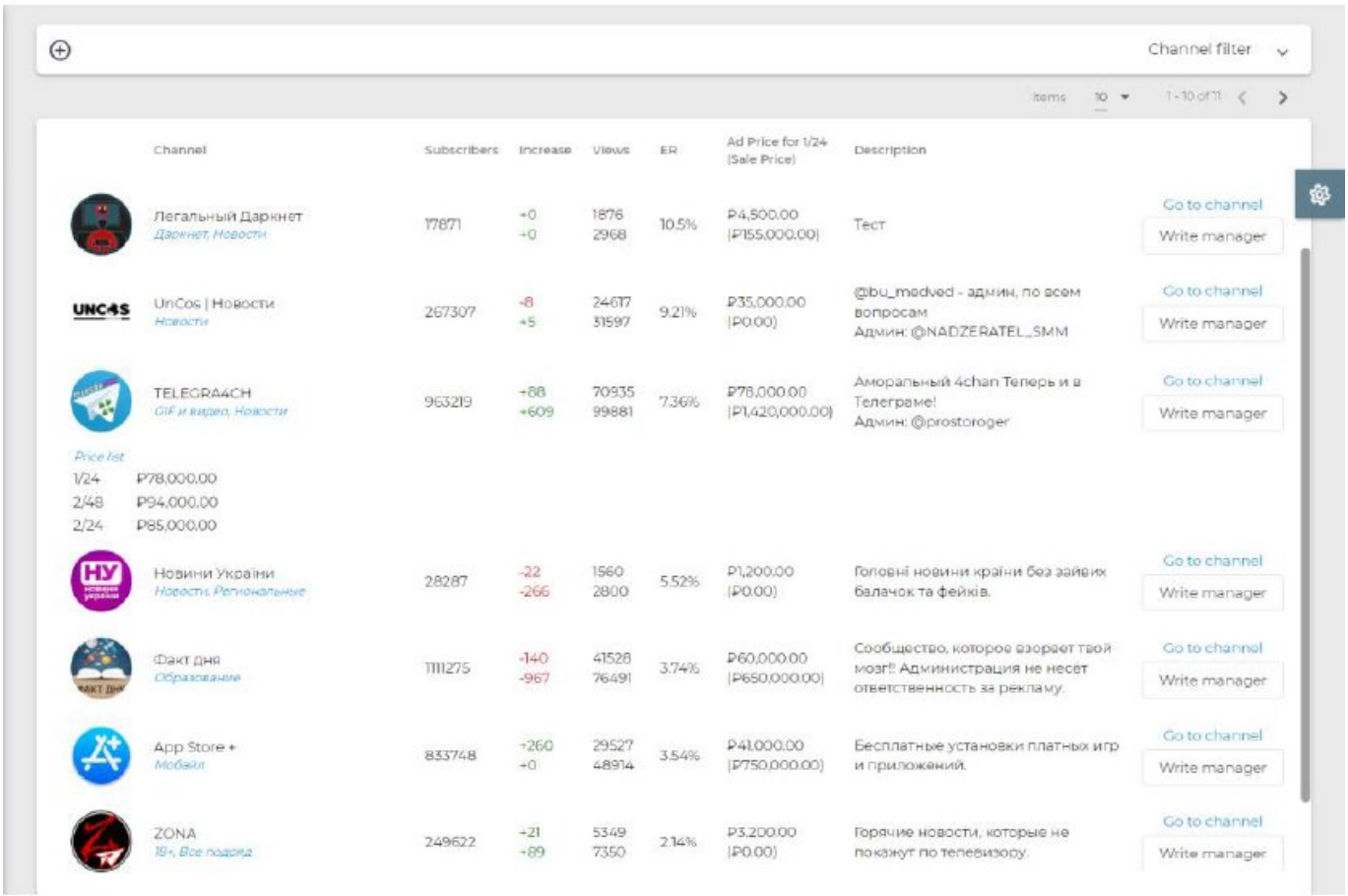

Рисунок 3.3.2.1 Розгорнутий прайс-ліст каналу

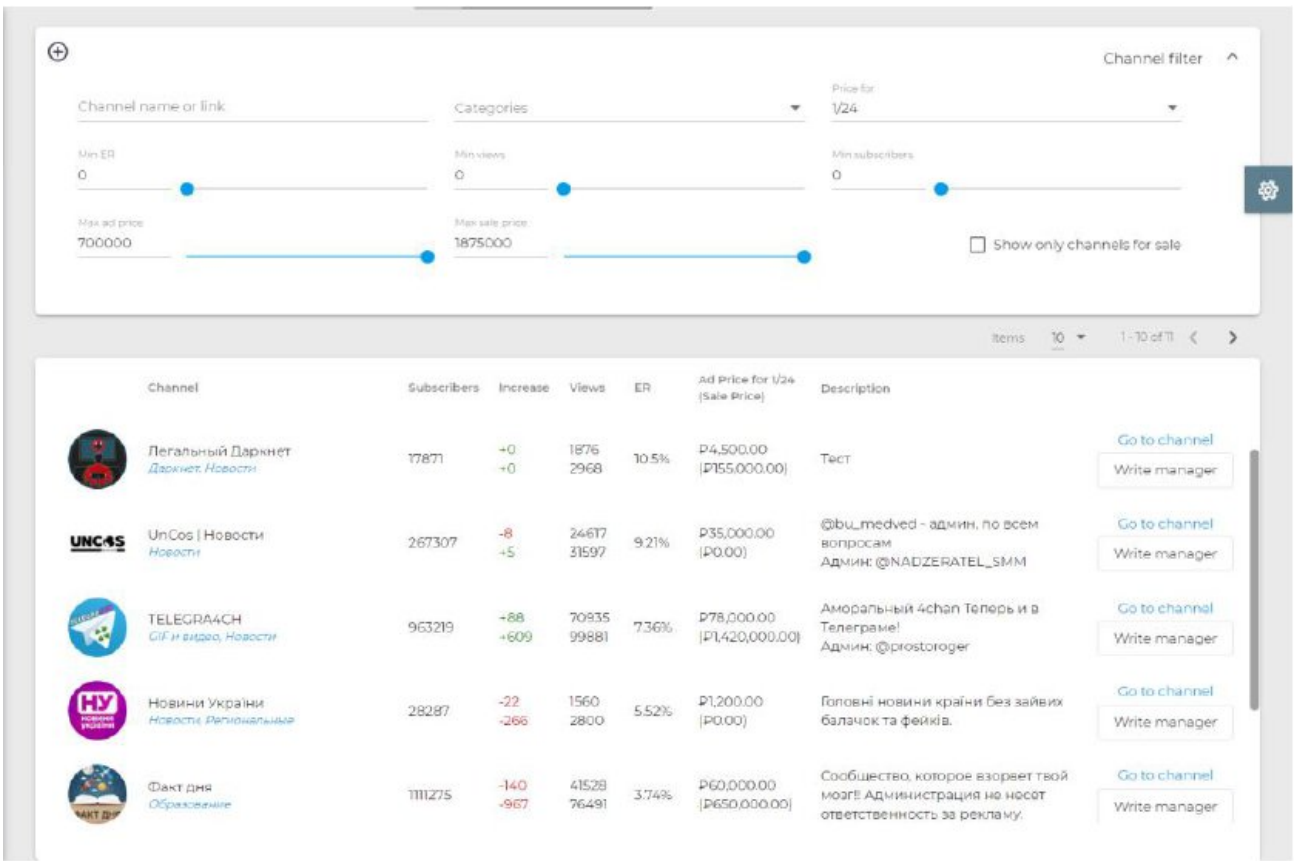

Рисунок 3.3.2.2 Розгорнутий фільтр каналів

При цьому на сторінці є зручний фільтр, що згортається та розгортається і оновлює список каналів в реальному часі (рис. 3.3.2.2). Основні критерії для фільтру: назва або посилання на канал, категорія, тип реклами, мінімальний відсоток відношення кількості підписників до кількості переглядів на публікаціях, мінімальна кількість переглядів на публікаціях, мінімальна кількість підписників, максимальна ціна на рекламу та максимальна ціна на продаж каналу.

3.3.3. Сторінка рекламних постів

Ще однією сторінкою на сайті є сторінка із рекламними постами (рис. 3.3.3.1). Там користувач може переглянути рекламні публікації, які пропонують рекламодавці. Кожна рекламна публікація містить наступну інформацію: медіа-контент (зображення/анімація/відео), категорії каналів, що підходять для розміщення, дату та час публікації, максимальну ціна за рекламу, мінімальну кількість переглядів та підписників.

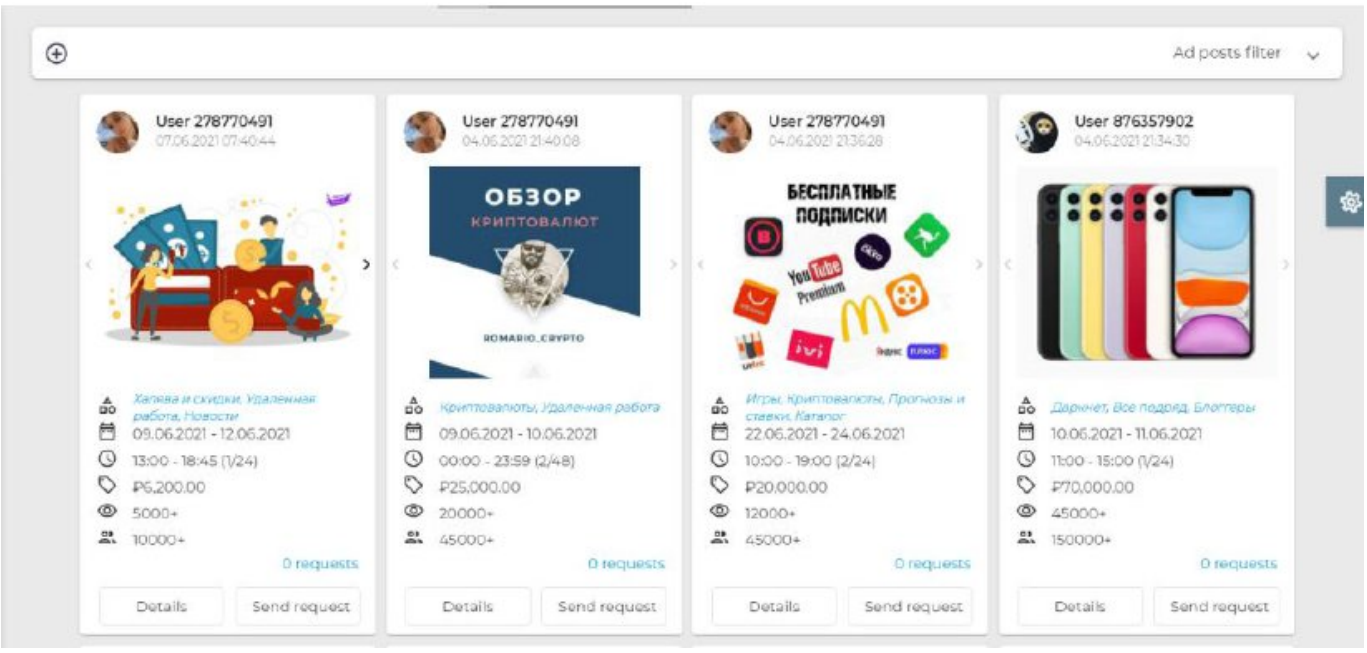

Рисунок 3.3.3.1 Сторінка рекламних постів

Натиснувши на кнопку «Надіслати заявку», відображається вікно для введення наступної інформації: канал (один з випадаючого списку із каналами поточного користувача, що відповідають вимогам), коментар, дата та час публікації, тип реклами та ціна (рис. 3.3.3.2).

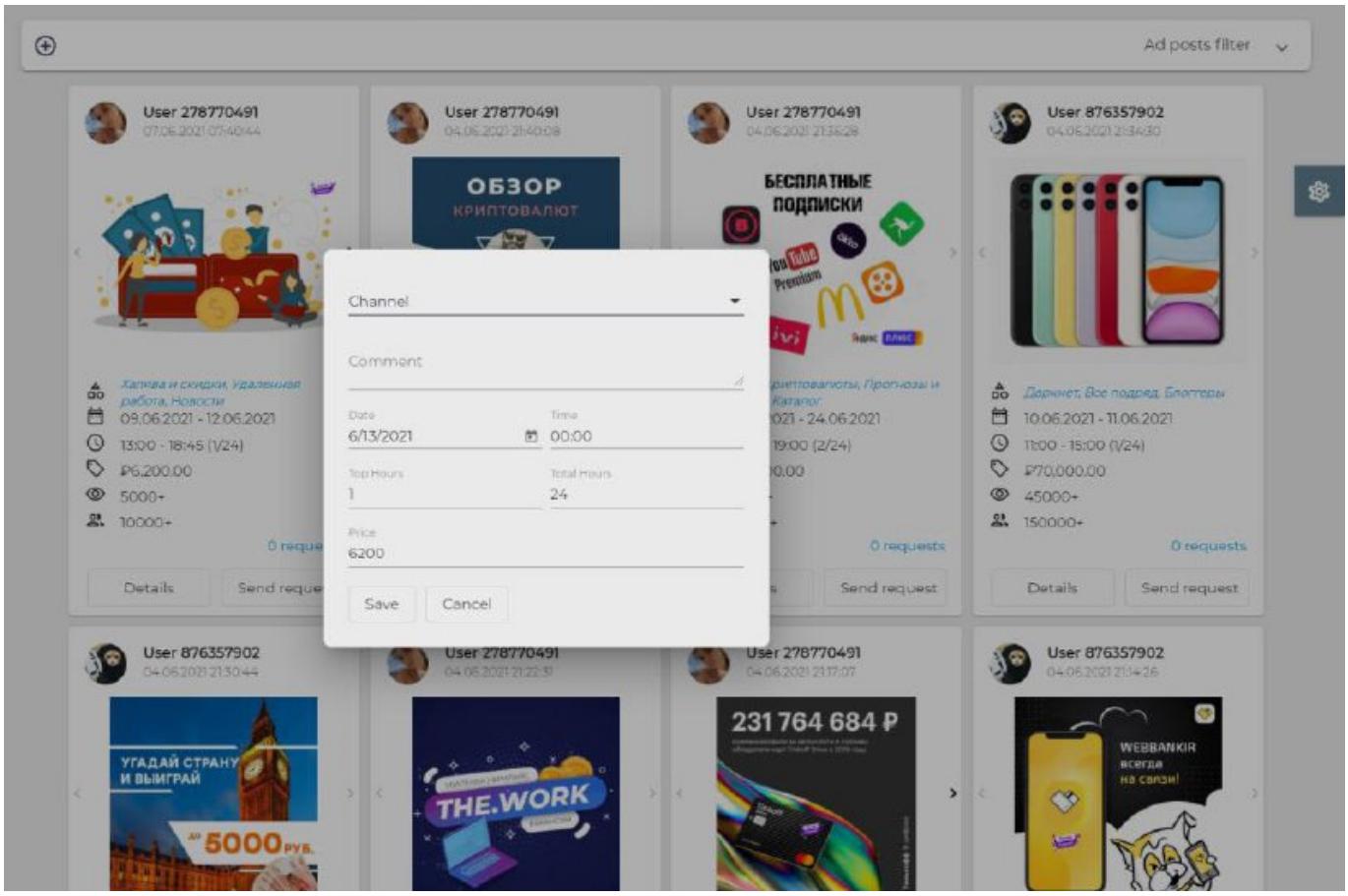

Рисунок 3.3.3.2 Форма створення нової заявки на рекламний пост

3.3.4. Сторінка профілю

Також була створена сторінка профілю, яка містить 4 основні вкладки: канали поточного користувача (рис. 3.3.4.1), рекламні пости, створені поточним користувачем (рис. 3.3.4.2), заявки на рекламу, подані поточним користувачем (рис. 3.3.4.3) та прийняті поточним користувачем заявки на розміщення реклами зі статусом оплати (рис. 3.3.4.4).

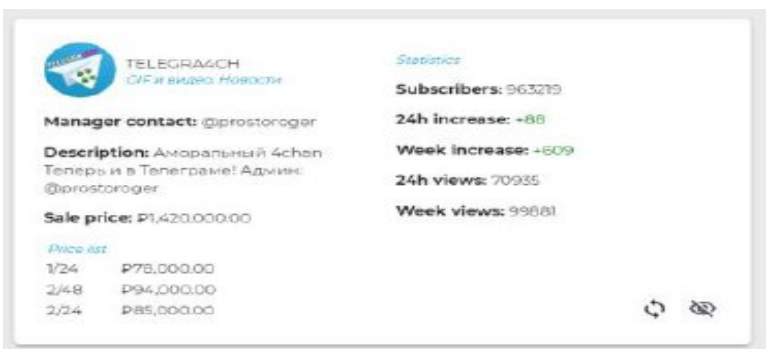

Рисунок 3.3.4.1 Вкладка власних каналів

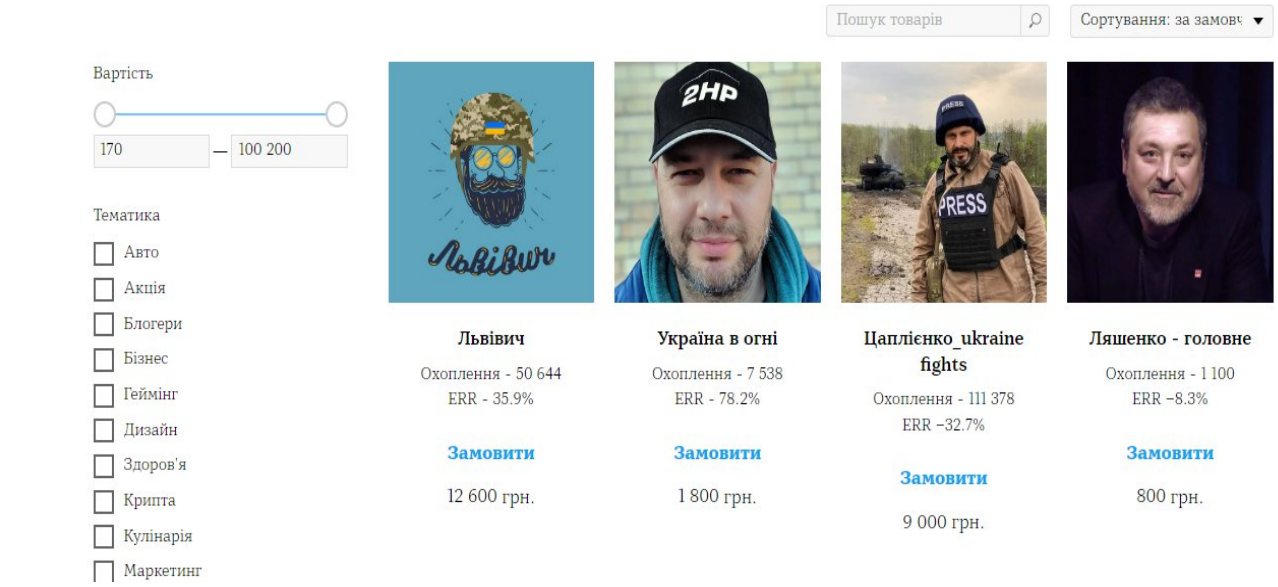

Рисунок 3.3.4.2 Вкладка власних рекламних постів

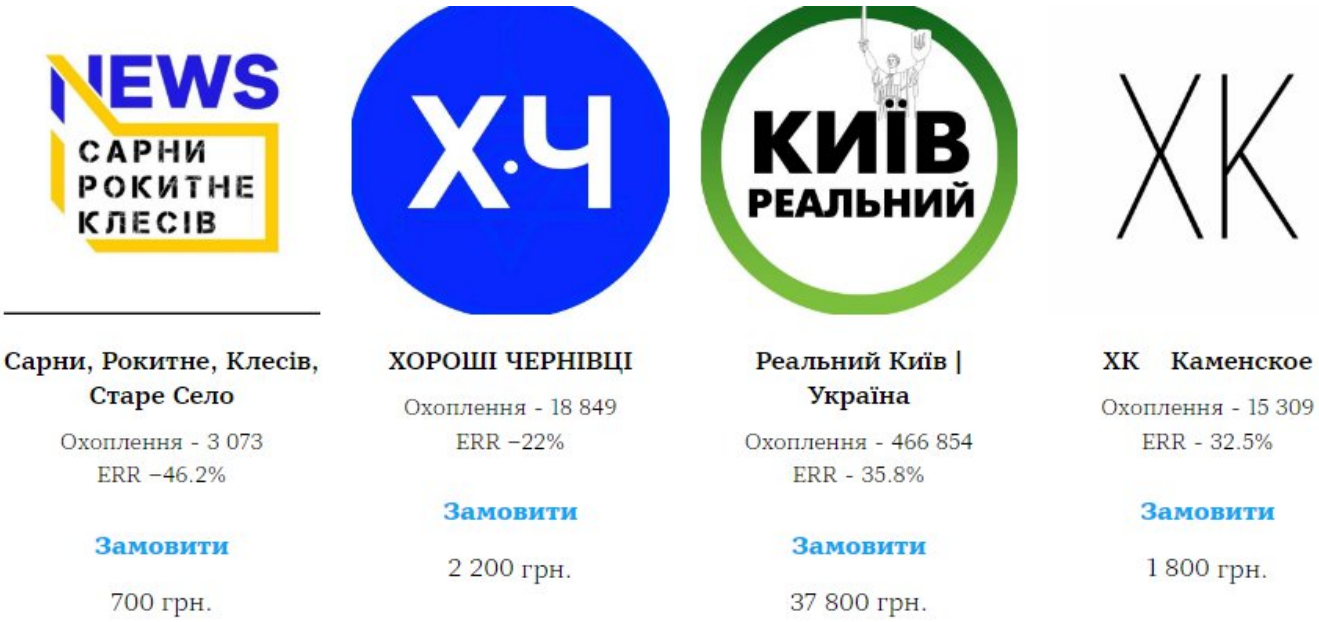

Рисунок 3.3.4.3 Вкладка власних заявок розміщення реклами

Сарни, Рокитне, Клесів, Старе Село

Охоплення - 3 073
ERR −46.2%

Замовити

700 грн.

Рисунок 3.3.4.4 Вкладка прийнятих заявок на власні рекламні пости

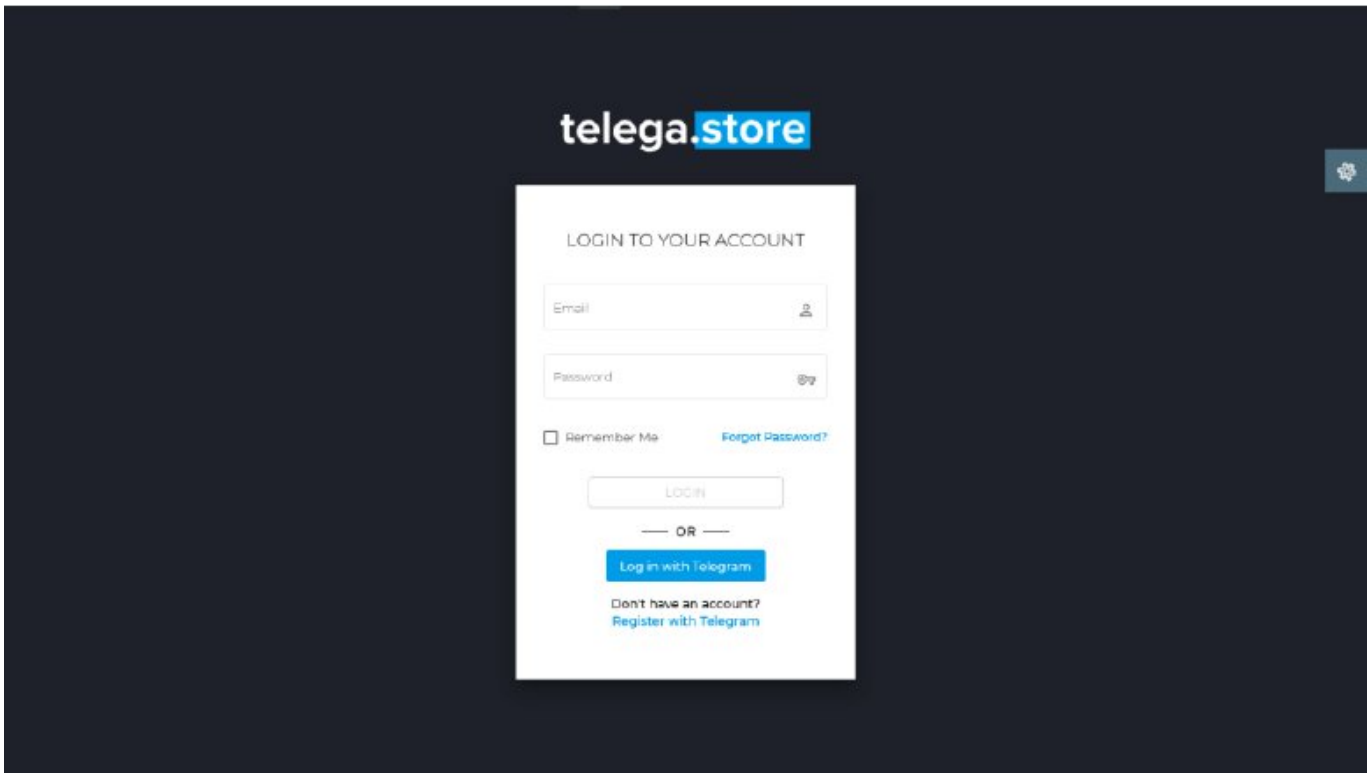

Рисунок 3.3.4.5 Сторінка реєстрації

Для того, щоб використовувати більшість функцій веб-додатку, потрібно зареєструватися на сайті. Для цього була розроблена сторінка реєстрації з формою, що складається з поля електронної пошти (нікнейму), паролю (рис. 3.3.4.5).

3.4. CI/CD

Continuous Integration (рис. 3.4.1) і Continuous Deployment (рис. 3.4.2 і рис. 3.4.3) було впроваджено для оперативності виводу нового функціоналу веб-сервісу.

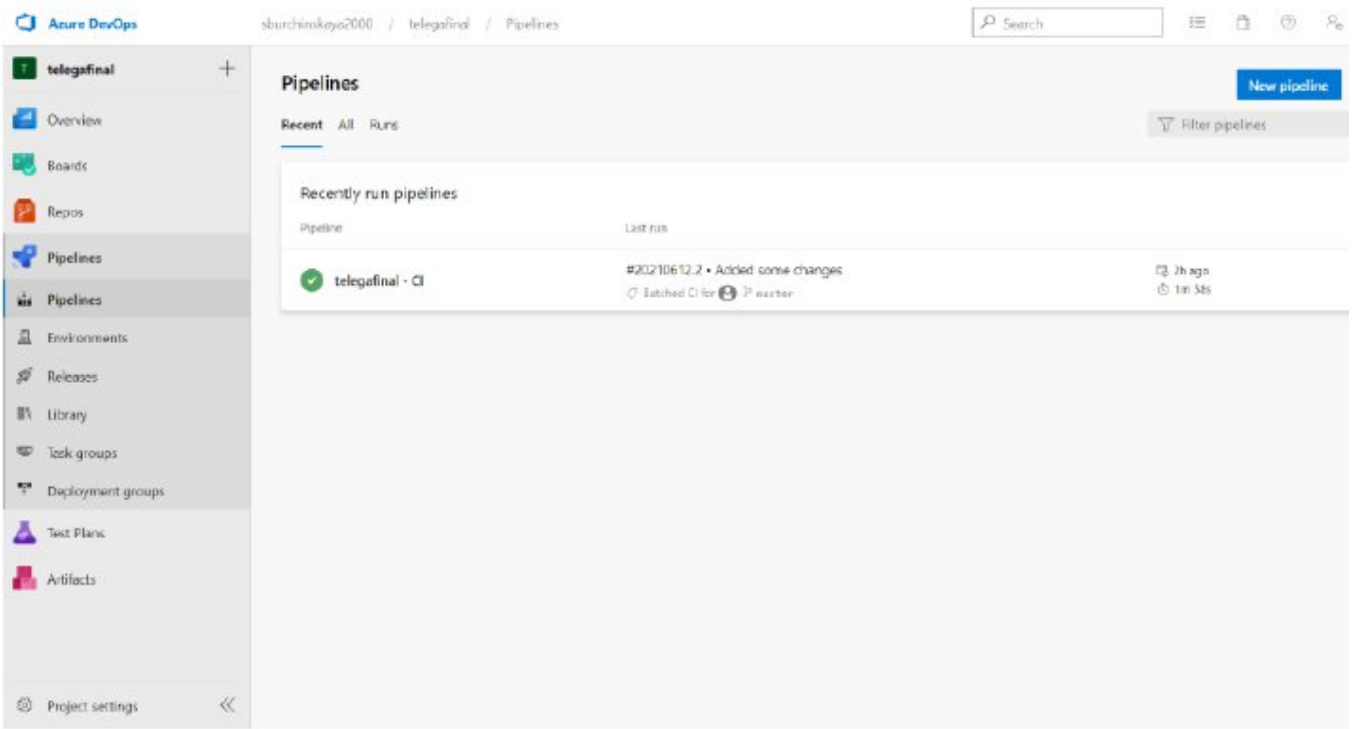

Рисунок 3.4.1 Continuous Integration додатку

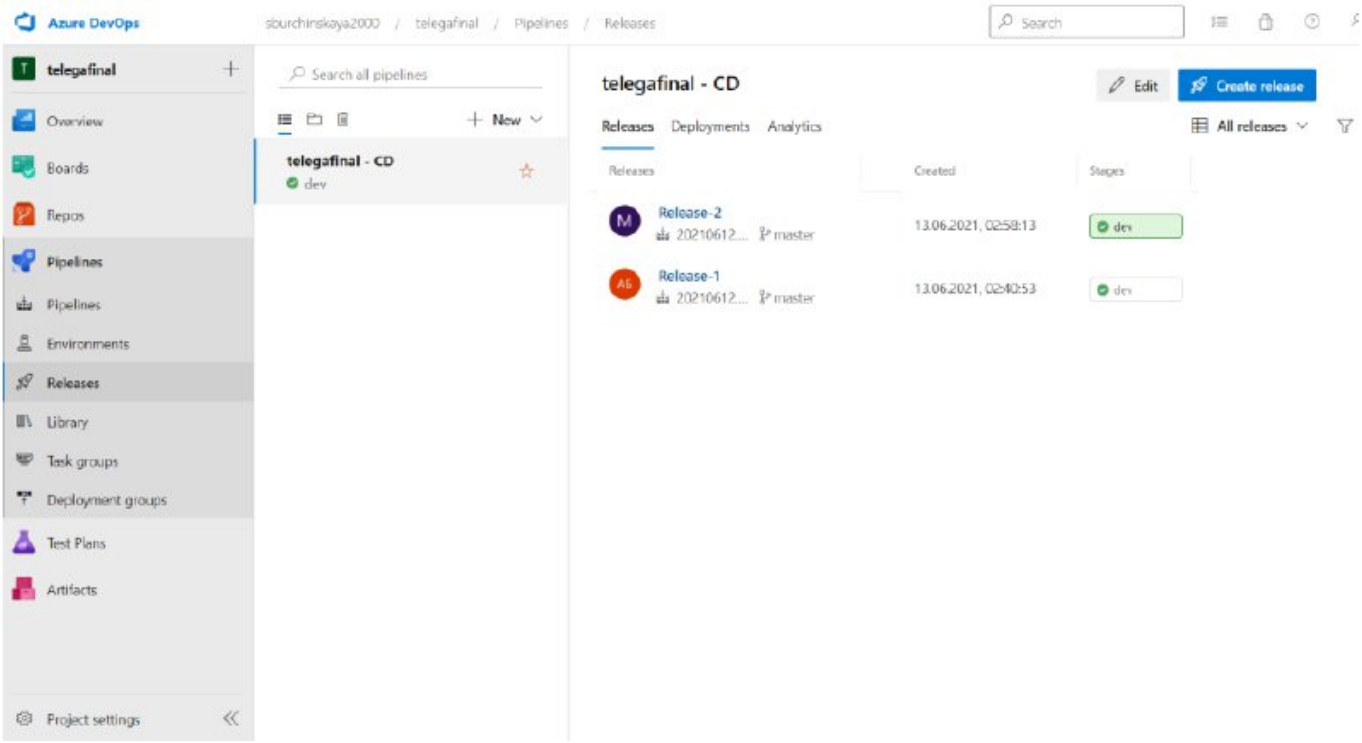

Рисунок 3.4.2 Continuous Deployment

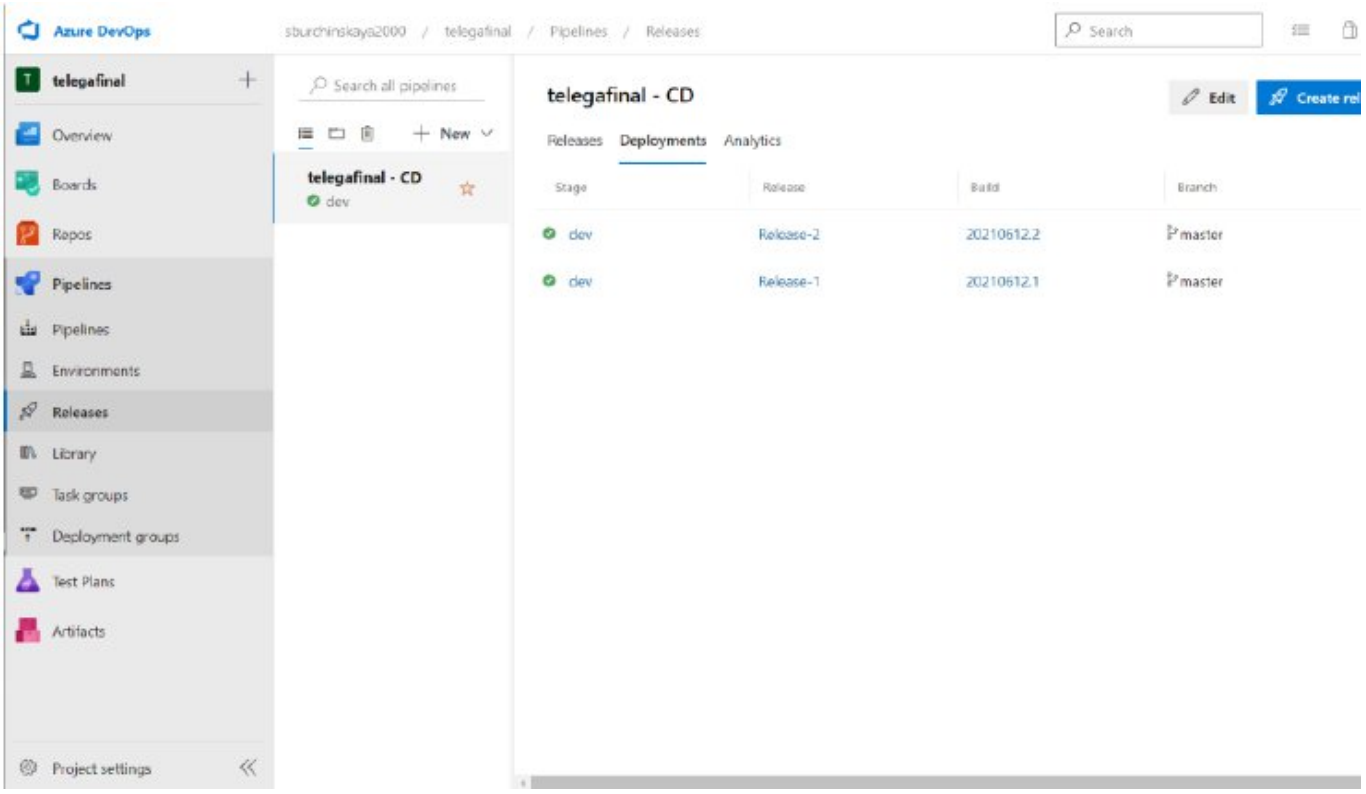

Рисунок 3.4.3 Continuous Deployment

Якість сайту значно підвищується завдячуючи паралельному тестуванню функціональних блоків. Вузькі місця та критичні моменти можна зафіксувати та відробити ще на ранніх стадіях циклу. За допомогою цих методів було знижено ризики при проходженні веб-додатку через стадії життєвого циклу, оскільки забезпечено контроль цілісності бізнес-логіки, користувацького досвіду, оптимізація зберігання та обробки даних, міграції.

3.5. Unit – тестування

Однією з найбільш важливих складових розроблюваного проекту є тестування. Для запобігання неочікуваних помилок та безпечного розширення функціоналу інформаційно-біржевого сервісу було написано ряд Unit-тестів.

Вони забезпечують чітку перевірку окремих модулів вихідного коду проєкту і дозволять іншим розробникам в перспективі розбиратися у коді, використовуючи тести в якості документації.

При написанні Unit-тестів було використано патерн AAA (Arrange, Act, Assert). В блоці Arrange відбувається налаштування тестового оточення тестованого юніта, в

Act – виконання або виклик сценарію, в Assert – перевірка того, що протестований виклик поводить себе відповідним шляхом.

3.6. Огляд функціональних можливостей розробленої інформаційної системи

До функціональних можливостей розробленої інформаціно-біржевої системи з просування реклами можемо віднести наступні пункти:

1. Реєстрація користувача (рис. 3.6.1) – відбувається у два кліки за допомогою Telegram-бота, який збирає необхідну інформацію в автоматичному режимі.

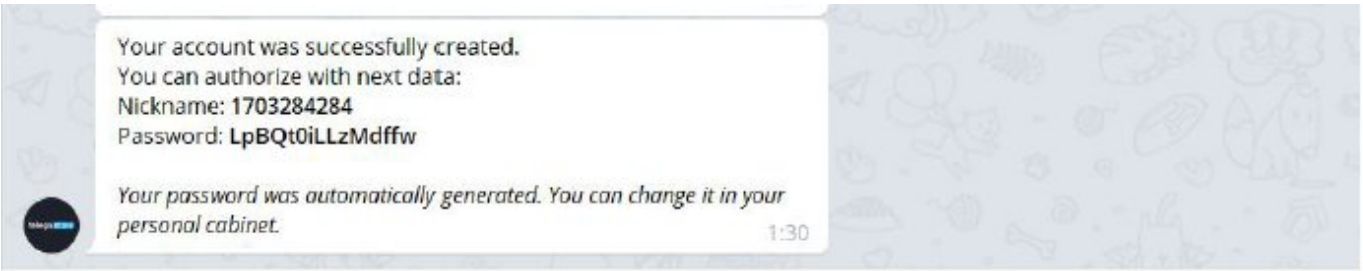

Рисунок 3.6.1 Реєстрація користувача

2. Аутентифікація користувача (рис. 3.6.2) – можлива як через веб-застосунок, так і через один клік в Telegram-боті. Для забезпечення якості введених даних була створена валідація полей форми.

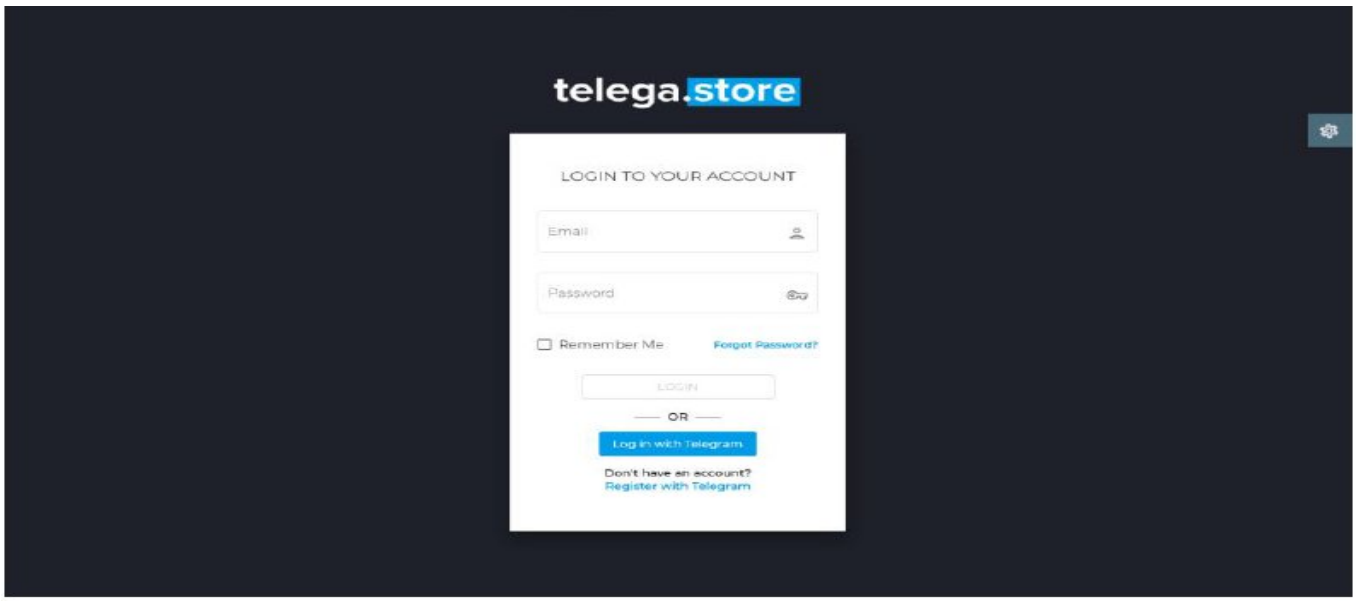

Рисунок 3.6.2 Аутентифікація користувача

3. Перегляд списку каналів зі статистикою із застосуванням зручного фільтру (рис. 3.6.3) – при натисканні на певний канал бачимо прайс-лист.

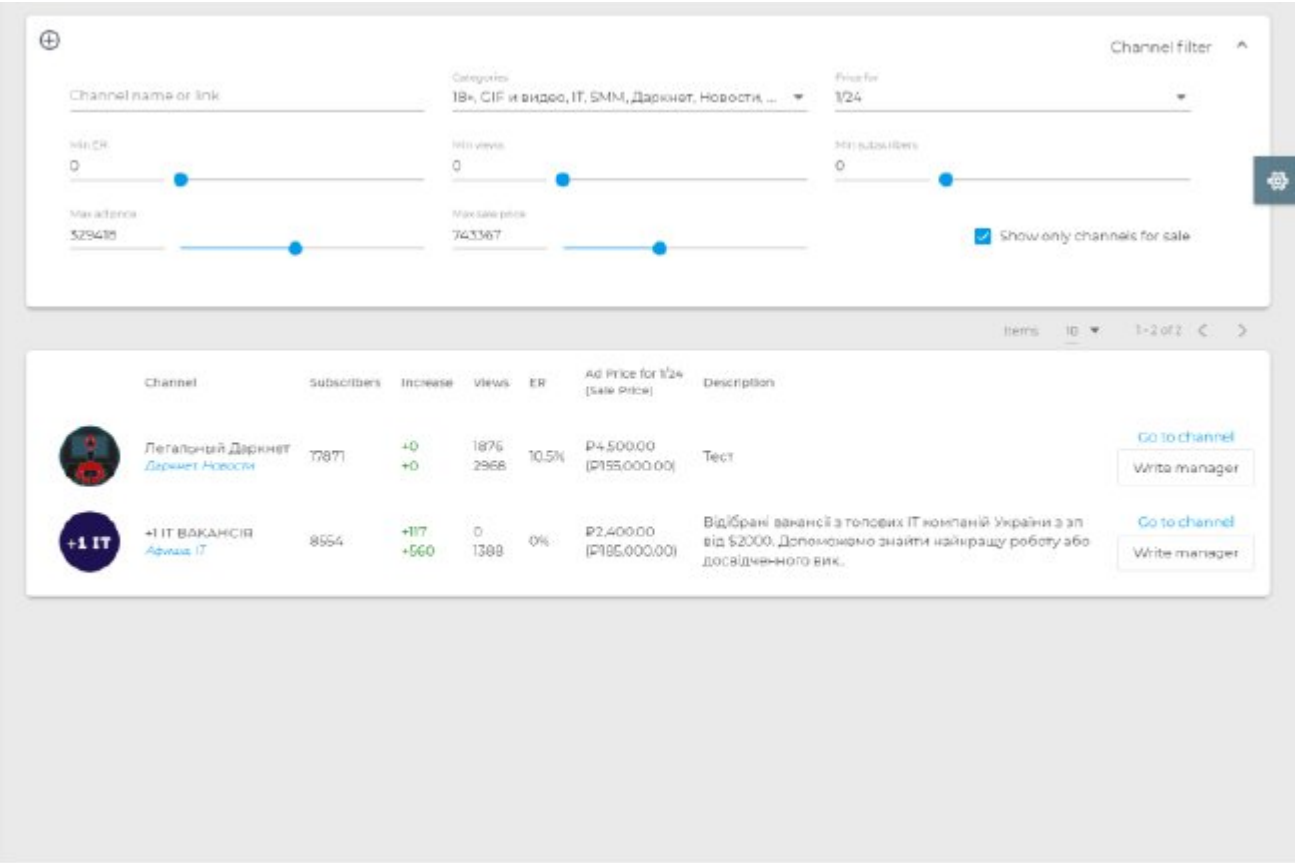

Рисунок 3.6.3 Перегляд списку каналів зі статистикою із застосуванням фільтру

4. Додавання в базу нового каналу (рис. 3.6.4) – три кроки: заповнення форми додавання із посиланням на канал, додавання до опису каналу згенерований додатком код для верифікаіції власника, надсилання боту поста з каналу для збору інформації та формування статистики, необхідної для рекламодавців при виборі прощадки для розміщення реклами.

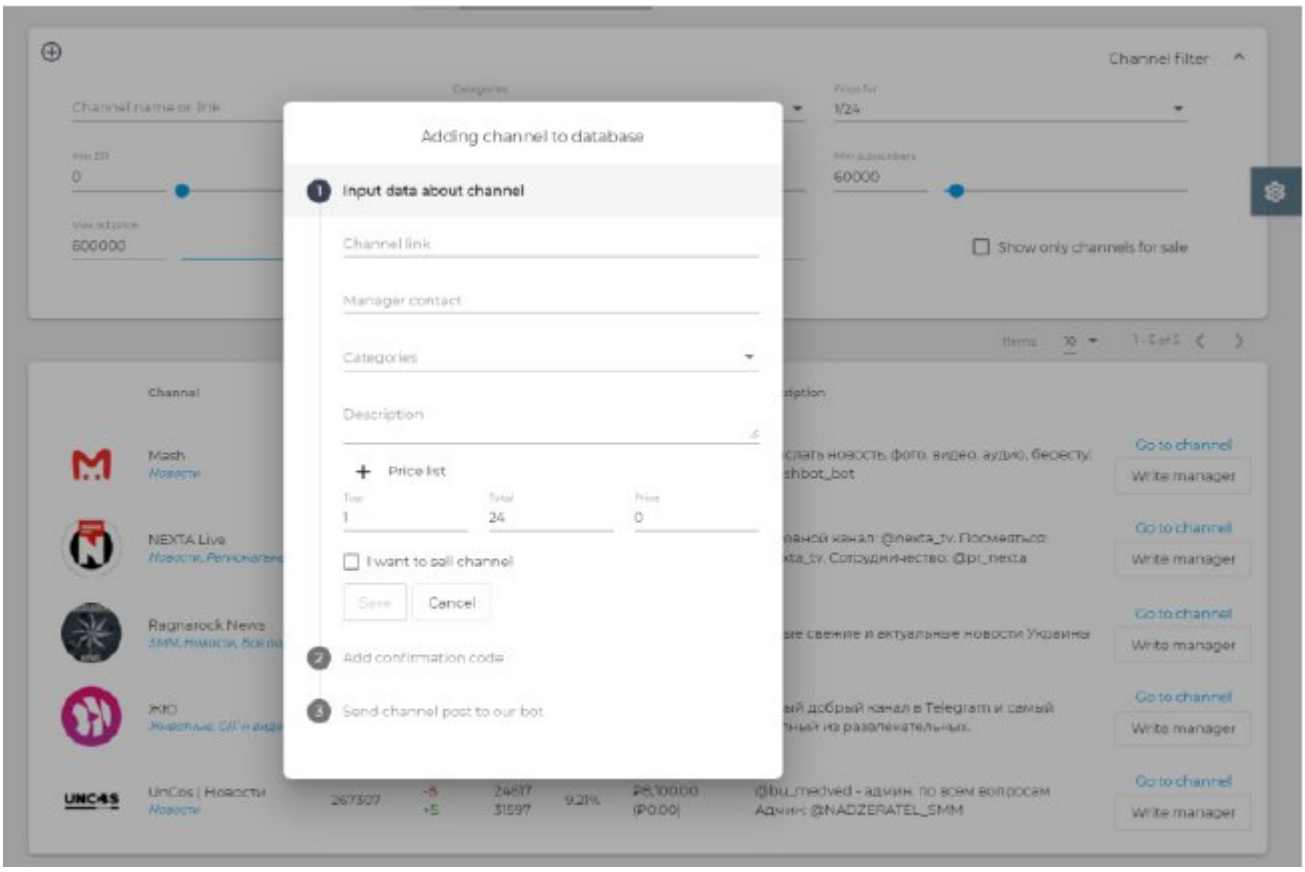

Рисунок 3.6.4 Додавання в базу нового каналу

5. Перегляд списку рекламних постів із застосуванням зручного фільтру (рис. 3.6.5).

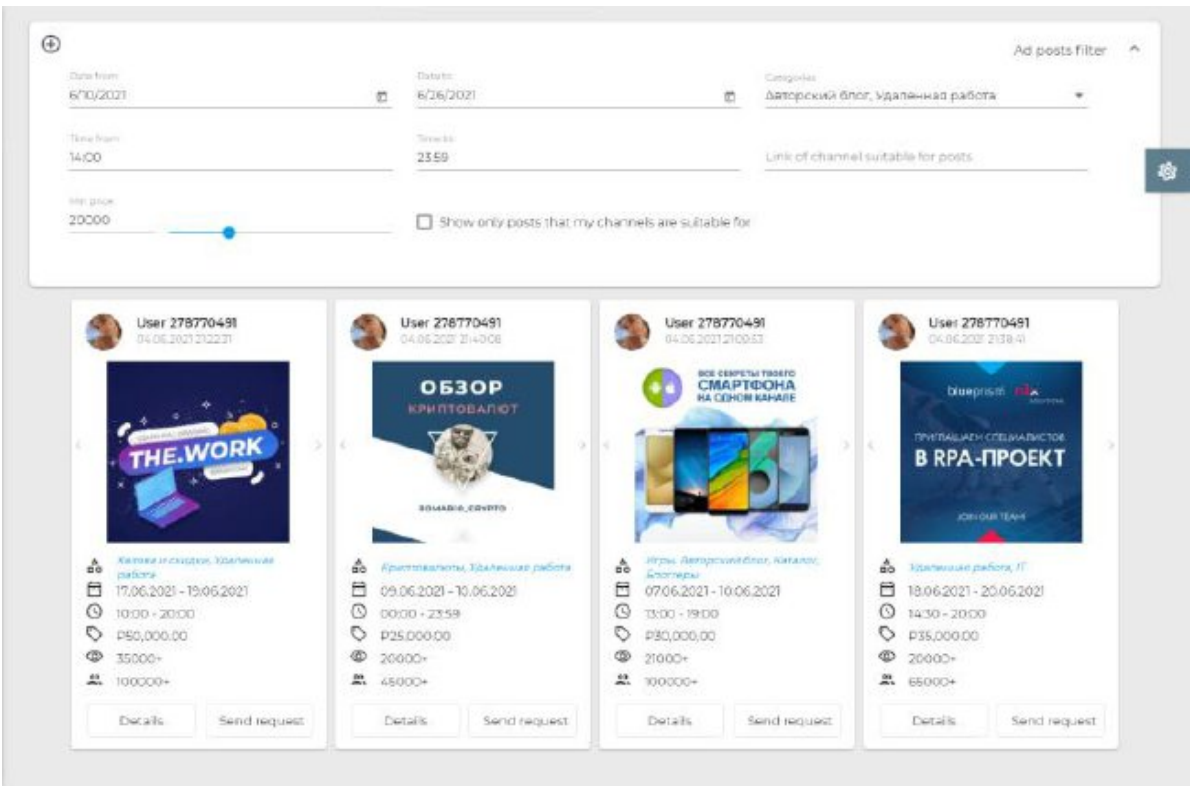

Рисунок 3.6.5 Перегляд списку рекламних постів із застосуванням фільтру

6. Створення нового рекламного поста (рис. 3.6.6) – три кроки, заповнення форми з основними даними, вибір способу створення самого поста: через конструктор, вбудований у веб-застосунок чи пересиланням поста боту, створення самого поста.

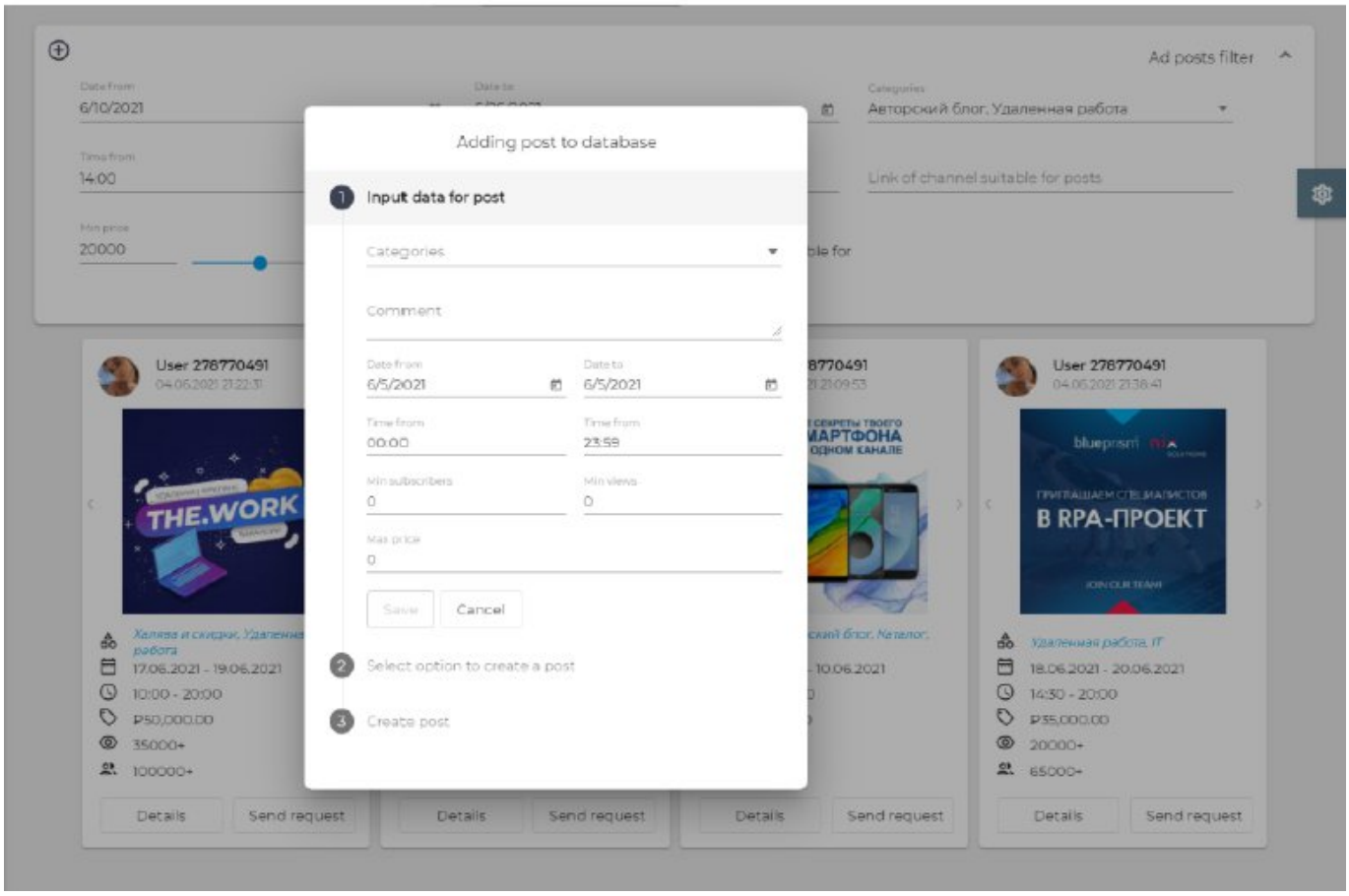

Рисунок 3.6.6 Створення нового рекламного поста

7. Подача заявки на рекламний пост (рис. 3.6.7) – вкаазання даних про розміщення поста, що відповідають вимогам рекламного поста.

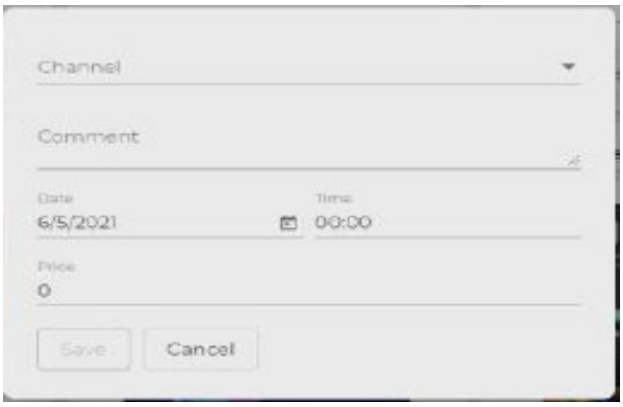

Рисунок 3.6.7 Подача заявки на рекламний пост

8. Перегляд та підтвердження заявки на пост (рис. 3.6.8) – можна відхиляти заявки, що не підходять.

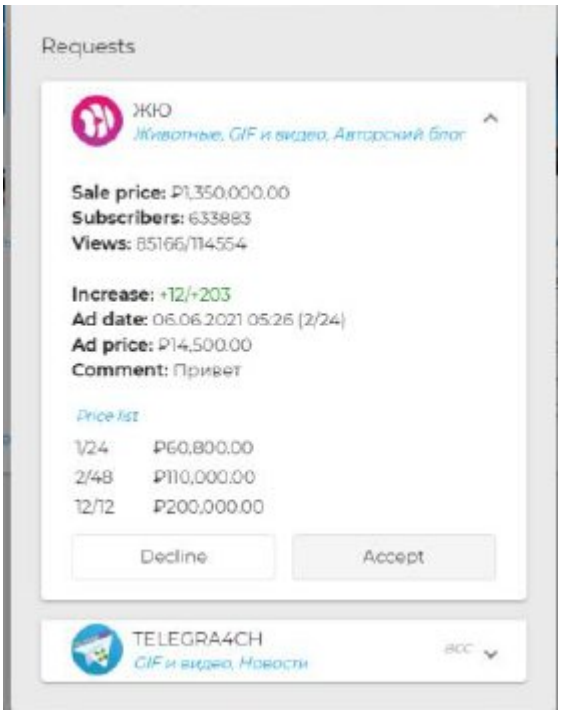

Рисунок 3.6.8 Перегляд та підтвердження заявки на пост

9. Оплата підтвердженої заявки (рис. 3.6.9) – за допомогою сервісу WayForPay. При підтвердженні заявки автоматично формується рахунок і QR-код додається на сторінку із підтвердженими заявками.

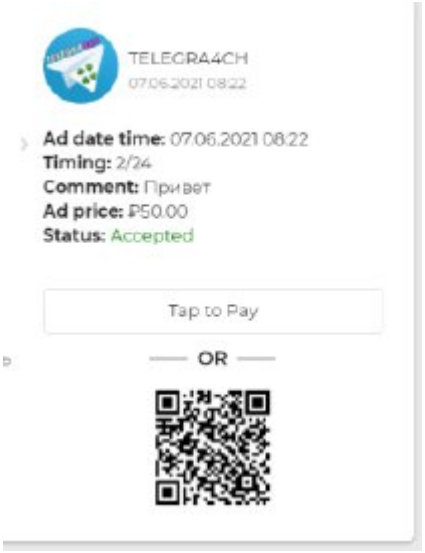

Рисунок 3.6.9 Оплата підтвердженої заявки

10. Редагування та приховання/відображення каналу (рис. 3.6.10)

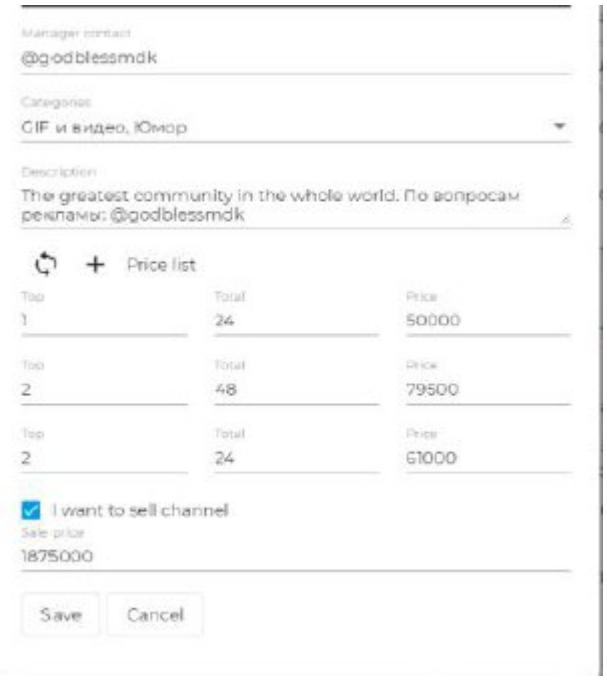

Рисунок 3.6.10 Редагування та приховання/відображення каналу

11.Видалення рекламного поста (рис. 3.6.11)

Рисунок 3.6.11 Видалення рекламного поста

Висновки

В ході виконання даної роботи було проведено аналіз та оцінку аналогів інформаційної системи, розроблено вебзастосунок з широким спектром функцій, що забезпечують зручні та надійні купівлю-продаж реклами по найвигіднішим цінам, а також моніторинг актуальної статистики каналів. Для надання найкращого користувацького досвіду було розроблено гнучкий та простий інтерфейс з мінімальною кількістю необхідних кліків (2-5) для проведення основних операцій.

Для створення програмного продукту було застосовано наступний стек технологій: серверна частина була розроблена на ASP.NET Core, користувацький інтерфейс - на Angular, Telegram-бот і клієнт - на Python з використанням Telegram API. Невід'ємною частиною роботи стало впровадження Continuous Integration та Continuous Deployment та розгортання додатку з використанням популярного сервісу Azure DevOps.

Завдяки розробленому сервісу користувачі можуть легко знаходити рекламодавців та площадки для розміщення власної реклами, отримувати лише актуальні дані про канали, що постійно оновлюються ботом, який самостійно збирає їх та обраховує. Порівнюючи додаток з аналогами, є фінансова перевага, оскільки запропоновані конкурентами 8% з продажу реклами telega.store скорочує мінімум до 5%.

Звичайний процес торгівлі рекламними послугами було вдосконалено, перетворено на більш надійний, зручний та швидкий спосіб купівлі-продажу шляхом мінімізації частки втручання користувачів у процеси, автоматизації значної кількості рутинних задач, таких як: перевірка оплати, створення рахунку, відкладений постинг.

В подальшому планується розробити функціонал розкладу каналів для того, щоб адміністратори та менеджери каналів мали під рукою вільні та зайняті місця на рекламу і могли надати можливість рекламодавцям переглядати їх самостійно, зекономивши час на консультуванні.

Список використаних джерел

1. Джулія Лерман: Programming Entity Framework: Code First: Creating and Configuring Data Models from Your Classes, 2015. – 194с.
2. Ерік Фрімен, Е.Робсон: Head First. Паттерни проектування / Діалектика-Вільямс, 2019. – 656 с.
3. Angular - Accessibility in Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/accessibility.
4. Angular - Lazy-loading feature modules [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/lazy-loading-ngmodules.
5. Banks A., Porcello E. Learning React: Functional Web Development with React and Redux. — O'Reilly Media, 2017. — 350 p.
6. Carlos R. React Cookbook: Create dynamic web apps with React using Redux, Webpack, Node.js, and GraphQL. — Packt Publishing, 2018. — 580 p.
7. Continuous integration vs. continuous delivery vs. continuous deployment [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.
8. DevSecOps: Injecting Security into CD Pipelines | Atlassian [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/continuous-delivery/principles/devsecops.
9. DevSecOps: Injecting Security into CD Pipelines | Atlassian [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/continuous-delivery/principles/devsecops.
10. Gorgon Z. React Explained: Your Step-by-Step Guide to React. – Amazon Digital Services LLC, 2019. — 305 p.
11. Introduction to SignalR | Microsoft Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr.
12. Pereira C. Node.js Building APIs with Node.js. — Apress, 2016. — 136 p.
13. Telethon's Documentation — Telethon 1.21.1 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.telethon.dev/en/latest/.

14. Telegram Bot API [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api.

Додатки

Додаток А

Збір та підрахунок статистичних даних:

```
async def get_messages(channel: Channel, days_num = 1):
grouped_ids = []
fmt = "%Y-%m-%d %H:%M:%S"
messages = await client.get_messages(
entity=channel,
limit=40*days_num)
tzinfo_utc = pytz.timezone("UTC")
today_ = datetime.datetime.now(tzinfo)
yesterday_ = today_ - timedelta(days=days_num)
today = datetime.datetime(today_.year, today_.month, today_.day, 0, 0, 0)
yesterday = datetime.datetime(yesterday_.year, yesterday_.month, yesterday_.day, 0, 0, 0)
today_utc = convert_datetime_timezone(today, "Europe", "UTC")
yesterday_utc = convert_datetime_timezone(yesterday, "Europe", "UTC")
total_views = 0
num_posts = 0
for msg in (messages):
if (days_num == 1):
print(msg.date)
if (msg.date <= today_utc):
if (msg.date >= yesterday_utc):
print(msg.views)
if (not(msg.grouped_id in grouped_ids)):
if (msg.grouped_id != None):
grouped_ids.append(msg.grouped_id)
total_views += msg.views
num_posts = num_posts + 1
else:
break
if (num_posts != 0):
```

```
return (total_views / num_posts)
return 0
channel_id = re.search('\/.* -100([0-9]*)', event.raw_text).group(1)
user_id = re.search('\/.* -100[0-9]* (.*)', event.raw_text).group(1)
res = await client(GetFullChannelRequest(int(channel_id)))
description = res.full_chat.about
cursor.execute("SELECT Id, ConfirmationCode FROM [TelegaStoreDBFinal].[dbo].[Channels] WHERE IsConfirmed = 'False' AND UserId = '"+user_id+"';")
channels = cursor.fetchall()
channel_id_sql = ""
for ch in channels:
if (ch[1] in description):
channel_id_sql = ch[0]
break
#channel_id_sql = channels[0][0]
if channel_id_sql == "":
return
channel = await client.get_entity(int(channel_id))
await client.download_profile_photo(channel, avatar_path + "channels/" + str(channel_id) + ".jpg")
views_for_day = await get_messages(channel, 1)
views_for_week = await get_messages(channel, 7)
er = round((views_for_day / res.full_chat.participants_count) * 100, 2)
query = f"UPDATE [TelegaStoreDBFinal].[dbo].[Channels] SET Name = N'"+res.chats[0].title+"', IsConfirmed = 'True', TelegramId = '"+str(channel_id)+"', Subscribers = "+str(res.full_chat.participants_count)+", ViewsForDay = "+str(views_for_day)+", ViewsForWeek = "+str(views_for_week)+", ImageLink = '"+channel_id +"', ER = "+str(er)+" WHERE Id = '"+channel_id_sql+"';"
print(query)
cursor.execute(query)    cursor.commit
```

Додаток Б

Кодова імплементація фільтру пошуку рекламних постів:

```
@Pipe({
name: 'filterPosts'
})
export class FilterPostsPipe implements PipeTransform {
transform(posts: Post[], searchPrice: number, searchDateFrom: Date,
searchDateTo: Date, searchTimeFrom: string, searchTimeTo: string,
searchCategories: string[], searchLink: string, searchOnlySuitable: boolean,
channels: Channel[], my_channels: Channel[]): Post[] {
var pipe = new DatePipe('en-US');
if (searchLink && posts != undefined) {
var channel = channels.find(x=>x.Link == searchLink);
console.log(channel)
if (channel) {
posts = posts.filter(x=>x.MinSubscribers <= channel.Subscribers
&& x.MinViews <= channel.ViewsForDay &&
x.Categories.some(r=> channel.Categories.map(e=>e.Id).includes(r.Id)));
}
else {
return [];
}
}
if (searchOnlySuitable === true) {
if (posts == undefined || my_channels == undefined) {
return [];
}
var new_posts = [];
posts.forEach(post=>{
if (my_channels.find(x=>x.Subscribers >= post.MinSubscribers &&
```

```
x.ViewsForDay >= post.MinViews && x.Categories.some(r=>
post.Categories.map(e=>e.Id).includes(r.Id))) {
new_posts.push(post);
}
});
posts = new_posts;
}
if (searchPrice && posts != undefined) {
posts = posts.filter(x=>x.Price > searchPrice);
}
if (searchDateFrom && posts != undefined) {
var date = pipe.transform(searchDateFrom, 'MM/dd/yyyy');
posts = posts.filter(x=>moment(pipe.transform(x.DateTimeTo, 'MM/dd/yyyy'),
'MM/DD/YYYY').isSameOrAfter(moment(date, 'MM/DD/YYYY')));
}
if (searchDateTo && posts != undefined) {
var date = pipe.transform(searchDateTo, 'MM/dd/yyyy');
posts = posts.filter(x=>moment(pipe.transform(x.DateTimeFrom, 'MM/dd/yyyy'),
'MM/DD/YYYY').isSameOrBefore(moment(date, 'MM/DD/YYYY')));
}
if (searchCategories.length > 0 && posts != undefined) {
posts = posts.filter(x=>searchCategories.some(r=> x.Categories.map(e=>e.Id).includes(r)));
}
if (searchTimeFrom == "") {
searchTimeFrom = searchTimeTo;
}
if (searchTimeTo == "") {
searchTimeTo = searchTimeFrom;
}
if (searchTimeTo != "" && posts != undefined) {
var today = moment(moment.now()).format("MM/DD/YYYY");
```

```
var yesterday = moment(moment.now()).subtract(1, "days").format("MM/DD/YYYY");
var new_posts = [];
var to = moment();
var from = moment();
if (searchTimeTo < searchTimeFrom) {
to = moment((today + " " + searchTimeTo), "MM/DD/YYYY HH:mm:ss");
from = moment((yesterday + " " + searchTimeFrom), "MM/DD/YYYY HH:mm:ss");;
}
else {
to = moment((today + " " + searchTimeTo), "MM/DD/YYYY HH:mm:ss");
from = moment((today + " " + searchTimeFrom), "MM/DD/YYYY HH:mm:ss");
}
posts.forEach(post=>{
var timeTo = pipe.transform(post.DateTimeTo, 'HH:mm');
var timeFrom = pipe.transform(post.DateTimeFrom, 'HH:mm');
var post_to = moment();
var post_from = moment();
if (timeTo < timeFrom) {
post_to = moment((today + " " + timeTo), "MM/DD/YYYY HH:mm:ss");
post_from = moment((yesterday + " " + timeFrom), "MM/DD/YYYY HH:mm:ss");
}
else {
post_to = moment((today + " " + timeTo), "MM/DD/YYYY HH:mm:ss");
post_from = moment((today + " " + timeFrom), "MM/DD/YYYY HH:mm:ss");
}
if (post_to.isBefore(from)) {
from = from.subtract(1, "days");
to = to.subtract(1, "days");
}
if (post_from.isAfter(to)) {
post_to = post_to.subtract(1, "days");
```

```
post_from = post_from.subtract(1, "days");
}
if (post_from.isSameOrBefore(to) && post_to.isSameOrAfter(from)) {
new_posts.push(post);
}
})
posts = new_posts;
}
return posts;
}
```

Додаток В

Лістинг коду розробки бази даних:

Моделі даних:

User:

```
public class User : IdentityUser<Guid>
{
public string Password { get; set; }
public string Nickname { get; set; }
public string Token { get; set; }
public Profile Profile { get; set; }
public Guid? SubscriptionId { get; set; }
public Subscription Subscription { get; set; }
public Guid? RoleId { get; set; }
public Role Role { get; set; }
public Guid? PaymentDetailId { get; set; }
public PaymentDetail PaymentDetail { get; set; }
public double WalletMoney { get; set; }
public bool Blocked { get; set; }
public long TelegramId { get; set; }
public string Code { get; set; }
public DateTime CodeGenerationDateTime { get; set; }
public List<Channel> Channels { get; set; }
public List<Post> Posts { get; set; }
public List<Request> Requests { get; set; }
public List<Deal> Deals { get; set; }
public List<Message> Messages { get; set; }
public List<UserChat> UserChats { get; set; }
public List<UserDeal> UserDeals { get; set; }
public User()
{
Channels = new List<Channel>();
```

```
Posts = new List<Post>();
Requests = new List<Request>();
Deals = new List<Deal>();
Messages = new List<Message>();
UserChats = new List<UserChat>();
UserDeals = new List<UserDeal>();
}
}
```

Category:

```
public class Category
{
public Guid Id { get; set; }
public string Name { get; set; }
public List<ChannelCategory> ChannelCategories { get; set; }
public Category()
{
ChannelCategories = new List<ChannelCategory>();
}
}
```

Channel:

```
public class Channel {
public Guid Id { get; set; }
public string Name { get; set; }
public string ImageLink { get; set; }
public int Subscribers { get; set; }
public int ViewsForWeek { get; set; }
public int ViewsForDay { get; set; }
public int IncreaseForWeek { get; set; }
public int IncreaseForDay { get; set; }
public float Percent { get; set; }
public float AdPrice { get; set; }
```

```
public float SalePrice { get; set; }
public bool IsForSale { get; set; }
public string Link { get; set; }
public string Description { get; set; }
public string Manager { get; set; }
public bool IsConfirmed { get; set; }
public DateTime LastInfoUpdate { get; set; }
public string ConfirmationCode { get; set; }
public Guid? UserId { get; set; }
public User User { get; set; }
public List<Post> Posts { get; set; }
public List<Request> Requests { get; set; }
public List<ChannelCategory> ChannelCategories {get; set;}
```

```
public Channel()
{
Posts = new List<Post>();
Requests = new List<Request>();
ChannelCategories = new List<ChannelCategory>();
}
}
```

ChannelCategory:

```
public class ChannelCategory
{
public Guid Id { get; set; }
public Guid ChannelId { get; set; }
public Channel Channel { get; set; }
public Guid CategoryId { get; set; }
public Category Category { get; set; }
}
```

Chat:

```
public class Chat
{
public Guid Id { get; set; }
public List<Message> Messages { get; set; }
public List<UserChat> UserChats { get; set; }
public Chat()
{
Messages = new List<Message>();
UserChats = new List<UserChat>();
}
}
```

Deal:

```
public class Deal
{
```

```
public Guid Id { get; set; }
public double Amount { get; set; }
public int Currency { get; set; }
public List<UserDeal> UserDeals { get; set; }
public bool Payed { get; set; }
public bool Closed { get; set; }
```

```
public Deal()
{
UserDeals = new List<UserDeal>();
}
}
```

Message:

```
public class Message
{
public Guid Id { get; set; }
public string Text { get; set; }
public Guid UserId { get; set; }
public User User { get; set; }
public Guid ChatId { get; set; }
public Chat Chat { get; set; }
public DateTime DateTime { get; set; }
public bool IsRequest { get; set; }
}
```

PaymentDetail:

```
public class PaymentDetail
{
public Guid Id { get; set; }
public string CardOwnerName { get; set; }
public string CardNumber { get; set; }
public string ExpirationDate { get; set; }
public string CVV { get; set; }
public Guid UserId { get; set; }
public User User { get; set; }
}
```

Post:

```
public class Post
{
```

```
public Guid Id { get; set; }
public Guid? RequestId { get; set; }
public Request Request { get; set; }
public string Text { get; set; }
public float Price { get; set; }
public int Currency { get; set; }
public string Attachments { get; set; }
public DateTime DateTimeFrom { get; set; }
public DateTime DateTimeTo { get; set; }
public Guid? UserId { get; set; }
public User User { get; set; }
public DateTime DateTime { get; set; }
public Guid? ChannelId { get; set; }
public Channel Channel { get; set; }
public List<Request> Requests { get; set; }
public bool IsHidden { get; set; }
public Post()
{
Requests = new List<Request>();
}
}
```

Profile:

```
public class Profile : IdentityUser<Guid>
{
public string NickName { get; set; }
public string RestorePasswordCode { get; set; }
public string PictureUrl { get; set; }
public Guid UserId { get; set; }
public User User { get; set; }
}
```

Request:

```
public class Request
{
public Guid Id { get; set; }
public Guid? PostId { get; set; }
public Post Post { get; set; }
public DateTime DateTime { get; set; }
public DateTime DateTimeFrom { get; set; }
public DateTime DateTimeTo { get; set; }
public Guid? UserId { get; set; }
public User User { get; set; }
public float Price { get; set; }
public int Currency { get; set; }
public string Comment { get; set; }
public bool IsHidden { get; set; }
public Guid ChannelId { get; set; }
public Channel Channel { get; set; }
public List<Post> Posts { get; set; }
```

```
public Request()
{
Posts = new List<Post>();
}
}
```

Role:

```
public class Role : IdentityRole<Guid>
{
public const string Admin = "Admin";
public const string User = "User";
public const string Moderator = "Moderator";
}
```

Subscription:

```
public class Subscription
{
public Guid Id { get; set; }
public string Name { get; set; }
public List<User> Users { get; set; }
public int PostsAmount { get; set; }
public int RequestsAmount { get; set; }
public bool AnalyticsAccess { get; set; }
public double Price { get; set; }
public int Currency { get; set; }
public Subscription()
{
Users = new List<User>();
}
}
```

UserChat:

```
public class UserChat
{
```

```
public Guid Id { get; set; }
public Guid UserId { get; set; }
public User User { get; set; }
public Guid ChatId { get; set; }
public Chat Chat { get; set; }
}
```

UserDeal:

```
public class UserDeal
{
public Guid Id { get; set; }
public Guid UserId { get; set; }
public User User { get; set; }
public Guid DealId { get; set; }
public Deal Deal{ get; set; }
}
```

Контекст бази даних:

```
public class TelegaContext : IdentityDbContext<User, Role, Guid>
{
public DbSet<Category> Categories { get; set; }
public DbSet<Channel> Channels { get; set; }
public DbSet<ChannelCategory> ChannelCategories { get; set; }
public DbSet<Deal> Deals { get; set; }
public DbSet<Message> Messages { get; set; }
public DbSet<PaymentDetail> PaymentDetails { get; set; }
public DbSet<Post> Posts { get; set; }
public DbSet<Profile> Profiles { get; set; }
public DbSet<Request> Requests { get; set; }
public DbSet<Subscription> Subscriptions { get; set; }
public TelegaContext()
{ }
public TelegaContext(DbContextOptions<TelegaContext> options)
```

```
: base(options)
{
Database.EnsureCreated();
}
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
modelBuilder.ApplyConfiguration(new CategoryConfiguration());
modelBuilder.ApplyConfiguration(new ChannelConfiguration());
modelBuilder.ApplyConfiguration(new ChannelCategoryConfiguration());
modelBuilder.ApplyConfiguration(new DealConfiguration());
modelBuilder.ApplyConfiguration(new MessageConfiguration());
modelBuilder.ApplyConfiguration(new PaymentDetailConfiguration());
modelBuilder.ApplyConfiguration(new ProfileConfiguration());
modelBuilder.ApplyConfiguration(new PostConfiguration());
modelBuilder.ApplyConfiguration(new SubscriptionConfiguration());
modelBuilder.ApplyConfiguration(new UserConfiguration());
modelBuilder.ApplyConfiguration(new RequestConfiguration());
base.OnModelCreating(modelBuilder);
}
}
```

Додаток Г

Приклад верстки сторінки каналів:

```
<div id="channels" class="page-layout carded fullwidth inner-scroll">
<!-- CENTER -->
<div class="center">
<div style="display: flex;margin-bottom: 10px; margin-top: 20px;">
<button mat-icon-button color="primary" class="add_channel" (click)="openDialog()">
<mat-icon>add_circle_outline</mat-icon>
</button>
<mat-accordion style="margin-left: -48px;">
<mat-expansion-panel (opened)="panelOpenState = true"
(closed)="panelOpenState = false">
<mat-expansion-panel-header>
<mat-panel-title>
<p class="filter_label">Channel filter</p>
</mat-panel-title>
</mat-expansion-panel-header>
<div class="example-container">
<form>
<mat-form-field>
<mat-label>Channel name or link</mat-label>
<input [formControl]="filterLink" matInput placeholder="E.g. Chop-Chop or t.me/chop-chop">
</mat-form-field>
<mat-form-field>
<mat-label>Categories</mat-label>
<mat-select [formControl]="filterCategories" [multiple]="true" #multiSelect>
<mat-option *ngFor="let category of categories" [value]="category.Id">{{category.Name}}</mat-option>
</mat-select>
</mat-form-field>
```

```
<mat-form-field>
<mat-label>Price for</mat-label>
<mat-select [formControl]="filterPriceFor" [multiple]="false">
<mat-option *ngFor="let variant of hoursVariants"
[value]="variant.TopHours+'/'+variant.TotalHours">{{variant.TopHours +
"/" + variant.TotalHours}}</mat-option>
</mat-select>
</mat-form-field>
<div class="filter_div">
<mat-form-field>
<mat-label class="slider-label">Min ER</mat-label>
<input matInput [formControl]="filterER" [(value)]="filteredValues.er">
</mat-form-field>
<mat-slider
(input)="onERChange($event)"
[displayWith]="formatLabel"
tickInterval="1"
step="1"
min="0"
max="100"
aria-label="units"
[(value)]="filteredValues.er"
[formControl]="filterER"></mat-slider>
</div>
<div class="filter_div">
<mat-form-field>
<mat-label class="slider-label">Min views</mat-label>
<input matInput [formControl]="filterViews"
[(value)]="filteredValues.views">
</mat-form-field>
<mat-slider
```

```
(input)="onViewsChange($event)"
[displayWith]="formatLabel"
tickInterval="1"
step="1"
min="0"
max="{{maxViews}}"
value="0"
aria-label="units"
[formControl]="filterViews"
[(value)]="filteredValues.views"></mat-slider>
</div>
<div class="filter_div">
<mat-form-field>
<mat-label class="slider-label">Min subscribers</mat-label>
<input matInput
[formControl]="filterSubscribers"
[(value)]="filteredValues.subscribers">
</mat-form-field>
<mat-slider
(input)="onSubscribersChange($event)"
[displayWith]="formatLabel"
tickInterval="1"
step="1"
min="0"
max="{{maxSubscribers}}"
value="0"
aria-label="units"
[formControl]="filterSubscribers"
[(value)]="filteredValues.subscribers"></mat-slider>
</div>
<div class="filter_div">
```

```
<mat-form-field>
<mat-label class="slider-label">Max ad price</mat-label>
<input matInput [formControl]="filterAdPrice"
[(value)]="filteredValues.adPrice">
</mat-form-field>
<mat-slider
(input)="onAdPriceChange($event)"
[formControl]="filterAdPrice"
[displayWith]="formatLabel"
tickInterval="1"
step="1"
min="0"
max="{{maxPrice}}"
[formControl]="filterAdPrice"
[(value)]="filteredValues.adPrice"
aria-label="units" name="adPrice"></mat-slider>
</div>
<div class="filter_div">
<mat-form-field>
<mat-label class="slider-label">Max sale price</mat-label>
<input matInput [formControl]="filterSalePrice"
[(value)]="filteredValues.salePrice" >
</mat-form-field>
<mat-slider
(input)="onSalePriceChange($event)"
[displayWith]="formatLabel"
tickInterval="1"
step="1"
min="0"
max="{{maxSalePrice}}"
aria-label="units"
```

```
[formControl]="filterSalePrice"
[(value)]="filteredValues.salePrice"></mat-slider>
</div>
<div class="filter_div check_filter">
<mat-checkbox [formControl]="filterOnlyForSale"
(change)="onChannelIsForSale($event.checked)" class="example-margin
sale_check">Show only channels for sale</mat-checkbox>
</div>
</form>
</div>
</mat-expansion-panel>
</mat-accordion>
</div>
<mat-paginator #paginator [pageSize]="pageSize" [pageSizeOptions]="[10, 20, 50]"
[length]="totalSize"
[pageIndex]="currentPage" (page)="pageEvent = handlePage($event)">
</mat-paginator>
<mat-spinner *ngIf="spinner" diameter="40" style="margin:0 auto;"></mat-spinner>
<mat-table *ngIf="!spinner" multiTemplateDataRows [dataSource]="channels"
class="channels-table">
<ng-container matColumnDef="imageLink">
<mat-header-cell *matHeaderCellDef>
</mat-header-cell>
<mat-cell *matCellDef="let channel">
<span class="avatar-container rounded-circle">
<img alt="Image placeholder" *ngIf="channel.ImageLink" [alt]="channel.Name"
class="avatar"
src="https://localhost:5000/api/Image/ChannelAvatar/{{channel.ImageLink}}.jpg"/>
<img *ngIf="!channel.ImageLink"
[src]="'assets/images/ecommerce/product-image-placeholder.png'">
</span>
```

```
</mat-cell>
</ng-container>
<ng-container matColumnDef="name">
<mat-header-cell *matHeaderCellDef> Channel </mat-header-cell>
<mat-cell *matCellDef="let channel">
<p class="text-truncate">{{channel.Name}}
<span class="categories">{{ channel.Categories | join }}</span>
</p>
</mat-cell>
</ng-container>
<ng-container matColumnDef="subscribers">
<mat-header-cell *matHeaderCellDef> Subscribers </mat-header-cell>
<mat-cell *matCellDef="let channel">
<p class="text-truncate">{{channel.Subscribers}}</p>
</mat-cell>
</ng-container>
<ng-container matColumnDef="increase">
<mat-header-cell *matHeaderCellDef> Increase </mat-header-cell>
<mat-cell *matCellDef="let channel">
<p class="text-truncate" [innerHTML]="(channel.IncreaseForDay | colorIncrease: channel.IncreaseForWeek)"></p>
</mat-cell>
</ng-container>
<ng-container matColumnDef="views">
<mat-header-cell *matHeaderCellDef> Views </mat-header-cell>
<mat-cell *matCellDef="let channel">
<p class="text-truncate">{{channel.ViewsForDay}}<br>{{channel.ViewsForWeek}}</p>
</mat-cell>
</ng-container>
<ng-container matColumnDef="er">
<mat-header-cell *matHeaderCellDef> ER </mat-header-cell>
```

```
<mat-cell *matCellDef="let channel">
<p class="text-truncate">{{channel.ER}}%</p>
</mat-cell>
</ng-container>
<ng-container matColumnDef="price">
<mat-header-cell *matHeaderCellDef> Ad Price for {{filterPriceFor.value}}<br>(Sale Price) </mat-header-cell>
<mat-cell *matCellDef="let channel">
<p class="text-truncate">{{channel.AdPrice | currency:'RUB':'symbol-narrow'}}<br>({{channel.SalePrice | currency:'RUB':'symbol-narrow'}})</p>
</mat-cell>
</ng-container>
<ng-container matColumnDef="description">
<mat-header-cell *matHeaderCellDef> Description </mat-header-cell>
<mat-cell *matCellDef="let channel">
<p class="text-truncate" style="white-space: pre-wrap">{{(channel.Description.length > 120) ? (channel.Description | slice:0:120)+'..':(channel.Description)}}</p>
</mat-cell>
</ng-container>
<ng-container matColumnDef="buttons">
<mat-header-cell *matHeaderCellDef></mat-header-cell>
<mat-cell *matCellDef="let channel" style="word-wrap: break-word;">
<table>
<tbody>
<tr>
<td class="text-center">
<a mat-primary class="go-to-channel" href="https://t.me/{{channel.Link}}">Go to channel</a>
</td>
</tr>
<tr>
```

```
<td class="text-center">
<a mat-stroked-button href="https://t.me/{{channel.Manager}}">Write manager</a>
</td>
</tr>
</tbody>
</table>
</mat-cell>
</ng-container>
<ng-container matColumnDef="expandedDetail">
<td mat-cell *matCellDef="let channel" [attr.colspan]="8">
<div class="example-element-detail" style="display: block;"
[@detailExpand]="channel == expandedElement ? 'expanded' : 'collapsed'">
<table>
<tr><em><td class="categories" colspan="2">Price list</td></em></tr>
<ng-container *ngFor="let price of channel.Prices">
<tr>
<td style="width: 60px;">{{price.TopHours}}/{{price.TotalHours}}</td>
<td>{{price.PriceValue | currency:'RUB':'symbol-narrow'}}</td>
</tr>
</ng-container>
</table>
</div>
</td>
</ng-container>
<mat-header-row *matHeaderRowDef="['imageLink', 'name', 'subscribers', 'increase',
'views', 'er', 'price', 'description', 'buttons']; sticky: true">
</mat-header-row>
<tr mat-row
[class.example-expanded-row]="expandedElement === channel"
(click)="expand(channel)" *matRowDef="let channel; columns: ['imageLink', 'name',
'subscribers', 'increase', 'views', 'er', 'price', 'description', 'buttons'];">
```

```
</tr>
<tr mat-row *matRowDef="let channel; columns: ['expandedDetail']" class="example-detail-row" [ngClass]="expandedElement !== channel ? 'example-detail-row collapsed' : 'example-detail-row'"></tr>
</mat-table>
</div>
</div>
```

Додаток Д

Лістинг коду інформаційної системи:

Telegram-бот:

```
from telethon import TelegramClient, sync, utils
from telethon.tl.types import PeerChannel
from telethon.sync import TelegramClient
from telethon import functions, types
import datetime
import pytz
from datetime import date, timedelta
from pytz import timezone
from telethon.tl.functions.channels import GetFullChannelRequest
import random
from telethon import events
import string
import urllib.request
import re
import certifi
import logging
import requests
import telebot
import json
import pyodbc
from telegram import Update, ForceReply
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters, CallbackContext
from telethon.tl.functions.messages import ImportChatInviteRequest
import asyncio
avatar_path= "C:/Users/Admin/source/repos/TelegacomApi/TelegacomApi/images/users/"
BOT_TOKEN="1700323441:AAGJPC8H72HXOyMKFIhKSCK34duhCwOEe24"
bot = telebot.TeleBot(BOT_TOKEN)
```

```
bot.set_webhook()
logger = logging.getLogger(__name__)
api_id = 3189981
api_hash = 'a3ce02cd7dbe6a2cf88fbb6b05e73a19'
server = 'MSSQLLocalDB'
database = 'TelegaStoreDBFinal'
client = TelegramClient('session_name', api_id, api_hash)
@bot.message_handler(commands=['start'])
def send_welcome(message):
print(message)
@bot.message_handler(commands=['register'])
def register(message):
conn = pyodbc.connect(r'DRIVER={SQL Server Native Client 11.0};Server=(localdb)\MSSQLLocalDB;Integrated Security=true; database = TelegaStoreDBFinal', autocommit = True)
cursor = conn.cursor()
bot.delete_message(
message.chat.id,
message.message_id)
subscription_id = ""
cursor.execute(f"SELECT TOP 1 Username FROM [TelegaStoreDBFinal].[dbo].[AspNetUsers] WHERE TelegramId = "+str(message.chat.id)+";");
result = cursor.fetchall();
if (len(result) > 0):
bot.send_message(message.chat.id, "You have been already registered.\nTo authorize use /auth command.")
else:
keyboard = telebot.types.InlineKeyboardMarkup()
keyboard.add(
telebot.types.InlineKeyboardButton(
```

```
'Register in Telega.store', callback_data='$pecial data f0r creat1ng new acc0unt'))
bot.send_message(message.chat.id, "Tap the button to end registration", reply_markup=keyboard)
@bot.message_handler(commands=['add_channel'])
def add_channel(message):
conn = pyodbc.connect(r'DRIVER={SQL Server Native Client 11.0};Server=(localdb)\MSSQLLocalDB;Integrated Security=true; database = TelegaStoreDBFinal', autocommit = True)
cursor = conn.cursor()
cursor.execute("UPDATE [TelegaStoreDBFinal].[dbo].[AspNetUsers] SET LastBotMessage = 'add_channel' WHERE TelegramId = "+str(message.chat.id)+";")
cursor.commit()
hashes = []
bot.delete_message(
message.chat.id,
message.message_id)
bot.send_message(
message.chat.id,
"Please forward any message from channel you are adding")
@bot.message_handler(commands=['add_post'])
def add_channel(message):
conn = pyodbc.connect(r'DRIVER={SQL Server Native Client 11.0};Server=(localdb)\MSSQLLocalDB;Integrated Security=true; database = TelegaStoreDBFinal', autocommit = True)
cursor = conn.cursor()
cursor.execute("UPDATE [TelegaStoreDBFinal].[dbo].[AspNetUsers] SET LastBotMessage = 'add_post' WHERE TelegramId = "+str(message.chat.id)+";")
cursor.commit()
hashes = []
bot.delete_message(
message.chat.id,
```

```
message.message_id)
bot.send_message(
message.chat.id,
"Please send me post")
@bot.message_handler(content_types=["document", "animation", "text", "photo", "video"])
def posts_from_channels(message):
conn = pyodbc.connect(r'DRIVER={SQL Server Native Client 11.0};Server=(localdb)\MSSQLLocalDB;Integrated Security=true; database = TelegaStoreDBFinal', autocommit = True)
cursor = conn.cursor()
print("1111111111")
cursor.execute("SELECT TOP 1 Id FROM [TelegaStoreDBFinal].[dbo].[AspNetUsers] WHERE TelegramId = '"+str(message.chat.id)+"';")
print("22222222222222")
user_id = cursor.fetchone()[0]
print("3333333333333")
cursor.execute("SELECT TOP 1 LastBotMessage FROM [TelegaStoreDBFinal].[dbo].[AspNetUsers] WHERE TelegramId = "+str(message.chat.id)+";")
print("4444444444444")
result = cursor.fetchone()
if (result[0] == "add_channel" and message.forward_from_chat):
cursor.execute("SELECT Link FROM [TelegaStoreDBFinal].[dbo].[Channels] WHERE IsConfirmed = 'False' AND UserId = '"+user_id+"';")
channel_links = cursor.fetchall();
hash = ""
for link in channel_links:
link = link[0]
found1 = re.findall('t\.me\/joinchat\/(.+)', link)
if (len(found1) > 0):
hash = found1[0]
```

```
else:
found2 = re.findall("t\.me\/(.+)", link)
if (len(found2) > 0):
hash = found2[0]
if (hash != ""):
print(hash)
bot.send_message(278770491, link+" "+str(message.forward_from_chat.id)+"
"+str(user_id))
else:
bot.send_message(
message.chat.id, "It seems like the link you entered <b>"+link+"</b> is wrong.",
parse_mode='HTML')
elif (result[0] == "add_post"):
fileID = ""
format = "jpg"
if (message.video != None):
fileID = message.video.file_id
format = message.video.mime_type;
hash = re.search('.+\/(.+)', format)
format = hash.group(1)
if (message.animation != None):
fileID = message.animation.file_id
format = message.animation.mime_type;
hash = re.search('.+\/(.+)', format)
format = hash.group(1)
if (message.photo != None):
fileID = message.photo[len(message.photo)- 1].file_id
if (fileID != ""):
file = bot.get_file(fileID)
```

```
cursor.execute("SELECT TOP 1 Id, TelegramMessageId, Attachments FROM [TelegaStoreDBFinal].[dbo].[Posts] WHERE UserId = '"+user_id+"' ORDER BY DateTime DESC;")
post = cursor.fetchone();
name = create_random_password();
if (post != None):
attachments = post[2]
if (post[1] != None):
if (attachments == ""):
attachments = name + "." + format
else:
attachments = attachments + ","+ name + "." + format
if (message.photo != None):
downloaded_file = bot.download_file(file.file_path)
src = 'C:\\Users\\Admin\\source\\repos\\TelegacomApi\\TelegacomApi\\images\\posts\\'+name + "." + format
with open(src, 'wb') as new_file:
new_file.write(downloaded_file)
if (message.video != None or message.animation != None):
urllib.request.urlretrieve("https://api.telegram.org/file/bot" + BOT_TOKEN + "/" + file.file_path,
'C:\\Users\\Admin\\source\\repos\\TelegacomApi\\TelegacomApi\\images\\posts\\'+name + "." + format);
text = "";
if (message.caption != None):
text = message.caption;
cursor.execute("UPDATE [TelegaStoreDBFinal].[dbo].[Posts] SET Attachments = '"+attachments+"', Text = '"+text+"' WHERE Id = '"+str(post[0])+"';")
cursor.commit()
@bot.message_handler(commands=['auth'])
```

```
def auth(message):
conn = pyodbc.connect(r'DRIVER={SQL Server Native Client 11.0};Server=(localdb)\MSSQLLocalDB;Integrated Security=true; database = TelegaStoreDBFinal', autocommit = True)
cursor = conn.cursor()
bot.delete_message(
message.chat.id,
message.message_id)
code = create_random_password(20)
chat_id = message.chat.id
cursor.execute(f"SELECT TOP 1 Username FROM [TelegaStoreDBFinal].[dbo].[AspNetUsers] WHERE TelegramId = "+str(message.chat.id)+";");
result = cursor.fetchall();
if (len(result) > 0):
cursor.execute(f"Update [TelegaStoreDBFinal].[dbo].[AspNetUsers] SET Code = '"+code+"' WHERE TelegramId = "+str(message.chat.id)+";");
cursor.commit()
keyboard = telebot.types.InlineKeyboardMarkup()
keyboard.add(
telebot.types.InlineKeyboardButton(
'Authorize in Telega.store', url='https://localhost:5000/api/Account/AuthorizationTG?verificationCode=' + code))
bot.send_message(message.chat.id, "Tap the button to authorize", reply_markup=keyboard)
else:
bot.send_message(message.chat.id, "You aren`t registered. To do this use /register command.")
@bot.callback_query_handler(func=lambda call: True)
def iq_callback(query):
```

```
conn = pyodbc.connect(r'DRIVER={SQL Server Native Client 11.0};Server=(localdb)\MSSQLLocalDB;Integrated Security=true; database = TelegaStoreDBFinal', autocommit = True)
cursor = conn.cursor()
bot.delete_message(
query.from_user.id,
query.message.message_id)
subscription_id = ""
username = query.from_user.id
password = create_random_password(15)
bot.send_message(278770491, 'avatar - ' + str(query.from_user.id))
headers = {'Content-type':'application/json', 'Accept':'application/json'}
result = requests.post('https://localhost:5000/api/Account/Registration', headers=headers, params={'Password':password, 'NickName': username}, verify=False)
bot.send_message(query.from_user.id, "Your account was successfully created. " +
"\nYou can authorize with next data:\nNickname: " +
"<b>"+str(username)+"</b> " +
"\nPassword: <b>"+password+"</b>\n\n<i>Your password was " +
"automatically generated. You can change it in your " +
"personal cabinet.</i>", parse_mode='HTML')
@bot.message_handler(commands=['auth'])
def send_welcome(message):
bot.reply_to(message, f'Привет, {message.from_user.first_name}. Для активації акаунта надішли мені свій код авторизації. @mr_grey_admin.')
def create_random_password(length=15):
validChars = "ABCDEFGHJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
# define the condition for random string
result = ''.join((random.choice(validChars)) for x in range(length))
return result
# Define a few command handlers. These usually take the two arguments update and
```

```
# context.
def start(update: Update, _: CallbackContext) -> None:
"""Send a message when the command /start is issued."""
user = update.effective_user
update.message.reply_markdown_v2(
fr'Hi {user.mention_markdown_v2()}\!',
reply_markup=ForceReply(selective=True),
)
def help_command(update: Update, _: CallbackContext) -> None:
"""Send a message when the command /help is issued."""
update.message.reply_text('Help!')
def main() -> None:
updater = Updater("1700323441:AAGJPC8H72HXOyMKFIhKSCK34duhCwOEe24")
#
print(bot.get_chat_members_count("-1001177521692"));
print(bot.get_history(limit = 1))
# Get the dispatcher to register handlers
dispatcher = updater.dispatcher
# on different commands - answer in Telegram
dispatcher.add_handler(CommandHandler("start", start))
dispatcher.add_handler(CommandHandler("help", help_command))
# on non command i.e message - echo the message on Telegram
dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, echo))
# Start the Bot
updater.start_polling()
updater.idle()
def convert_datetime_timezone(dt, tz1, tz2):
tz1 = pytz.timezone(tz1)
tz2 = pytz.timezone(tz2)
dt = tz1.localize(dt)
dt = dt.astimezone(tz2)
```

```
return dt
def get_messages(id: int):
grouped_ids = []
peer = PeerChannel(id)
fmt = "%Y-%m-%d %H:%M:%S"
messages = client.iter_messages(peer, 20)
tzinfo = pytz.timezone("Europe")
tzinfo_utc = pytz.timezone("UTC")
today_ = datetime.datetime.now(tzinfo)
yesterday_ = today_ - timedelta(days=1)
today = datetime.datetime(today_.year, today_.month, today_.day, 0, 0, 0)
yesterday = datetime.datetime(yesterday_.year, yesterday_.month, yesterday_.day, 0, 0, 0)
today_utc = convert_datetime_timezone(today, "Europe", "UTC")
yesterday_utc = convert_datetime_timezone(yesterday, "Europe", "UTC")
total_views = 0
num_posts = 0
for msg in reversed(list(messages)):
if (msg.date >= yesterday_utc):
if (msg.date <= today_utc):
if (not(msg.grouped_id in grouped_ids)):
if (msg.grouped_id != None):
grouped_ids.append(msg.grouped_id)
total_views += msg.views
num_posts = num_posts + 1
else:
break
if (num_posts != 0):
print(total_views / num_posts)
if __name__ == '__main__':
bot.polling(none_stop=True)
```

Telegram-клієнт:

```
#!/usr/bin/env python
# pylint: disable=C0116
# This program is dedicated to the public domain under the CC0 license.
from telethon import TelegramClient, sync, utils
from telethon.tl.types import PeerChannel
from telethon.sync import TelegramClient
from telethon import functions, types
from datetime import date, timedelta
from telethon.tl.functions.channels import GetFullChannelRequest
import random
from telethon import events
import string
import datetime
import pytz
from datetime import date, timedelta
from pytz import timezone
import re
import requests
import telebot
import pyodbc
from telegram import Update, ForceReply
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters, CallbackContext
from telethon.tl.functions.messages import ImportChatInviteRequest, CheckChatInviteRequest, GetHistoryRequest
from telethon.tl.functions.contacts import ResolveUsernameRequest
from telethon.tl.functions.channels import JoinChannelRequest
from telethon.tl.types import InputChannel, InputPeerChannel, Channel
import asyncio
from aiostream import stream
```

```
conn = pyodbc.connect(r'DRIVER={SQL Server Native Client 11.0};Server=(localdb)\MSSQLLocalDB;Integrated Security=true; database = TelegaStoreDBFinal; autocommit = True')
cursor = conn.cursor()
api_id = 3189981
avatar_path= "C:/Users/Admin/source/repos/TelegacomApi/TelegacomApi/images/"
api_hash = 'a3ce02cd7dbe6a2cf88fbb6b05e73a19'
client = TelegramClient('session_name', api_id, api_hash)
client.start()
def convert_datetime_timezone(dt, tz1, tz2):
tz1 = pytz.timezone(tz1)
tz2 = pytz.timezone(tz2)
dt = tz1.localize(dt)
dt = dt.astimezone(tz2)
return dt
async def get_messages(channel: Channel, days_num = 1):
grouped_ids = []
fmt = "%Y-%m-%d %H:%M:%S"
messages = await client.get_messages(
entity=channel,
limit=40*days_num)
tzinfo = pytz.timezone("Europe”)
tzinfo_utc = pytz.timezone("UTC")
today_ = datetime.datetime.now(tzinfo)
yesterday_ = today_ - timedelta(days=days_num)
today = datetime.datetime(today_.year, today_.month, today_.day, 0, 0, 0)
yesterday = datetime.datetime(yesterday_.year, yesterday_.month, yesterday_.day, 0, 0, 0)
today_utc = convert_datetime_timezone(today, "Europe", "UTC")
yesterday_utc = convert_datetime_timezone(yesterday, "Europe", "UTC")
total_views = 0
num_posts = 0
```

```
for msg in (messages):
if (days_num == 1):
print(msg.date)
if (msg.date <= today_utc):
if (msg.date >= yesterday_utc):
print(msg.views)
if (not(msg.grouped_id in grouped_ids)):
if (msg.grouped_id != None):
grouped_ids.append(msg.grouped_id)
total_views += msg.views
num_posts = num_posts + 1
else:
break
if (num_posts != 0):
return (total_views / num_posts)
return 0
@client.on(events.NewMessage(outgoing=False, pattern=r'avatar -'))
async def my_event_handler2(event):
hash = re.search('avatar - ([0-9]+)', event.raw_text)
if (hash != None):
entity = await client.get_entity(int(hash.group(1)))
async for photo in client.iter_profile_photos(entity):
await client.download_media(photo, avatar_path + "users/"+ hash.group(1)+".jpg")
break
@client.on(events.NewMessage(outgoing=False, pattern=r'(?i).*t\.me/'))
async def my_event_handler(event):
# extracts the hash of that link
hash = re.search('(?<=joinchat\/)(\w+[-]?\S\w+)', event.raw_text)
if (hash == None):
hash = re.search('(?<=t\.me\/)(\w+[-]?\S\w+)', event.raw_text)
if (hash == None):
```

```
print("ERROR")
else:
try:
result = await client(ResolveUsernameRequest(hash.group(0)))
channel = await InputChannel(result.peer.channel_id, result.chats[0].access_hash)
await client(JoinChannelRequest(channel))
except Exception:
print("ERROR")
else:
try:
updates = await client(ImportChatInviteRequest(hash.group(0)))
except Exception:
print("ERROR")
channel_id = re.search('\/.* -100([0-9]*)', event.raw_text).group(1)
user_id = re.search('\/.* -100[0-9]* (.*)', event.raw_text).group(1)
res = await client(GetFullChannelRequest(int(channel_id)))
description = res.full_chat.about
cursor.execute("SELECT Id, ConfirmationCode FROM [TelegaStoreDBFinal].[dbo].[Channels] WHERE IsConfirmed = 'False' AND UserId = '"+user_id+"';")
channels = cursor.fetchall()
channel_id_sql = ""
for ch in channels:
if (ch[1] in description):
channel_id_sql = ch[0]
break
#channel_id_sql = channels[0][0]
if channel_id_sql == "":
return
channel = await client.get_entity(int(channel_id))
```

```
await client.download_profile_photo(channel, avatar_path + "channels/" + str(channel_id) + ".jpg")
views_for_day = await get_messages(channel, 1)
views_for_week = await get_messages(channel, 7)
er = round((views_for_day / res.full_chat.participants_count) * 100, 2)
query = f"UPDATE [TelegaStoreDBFinal].[dbo].[Channels] SET Name = N'"+res.chats[0].title+"', IsConfirmed = 'True', TelegramId = '"+str(channel_id)+"', Subscribers = "+str(res.full_chat.participants_count)+", ViewsForDay = "+str(views_for_day)+", ViewsForWeek = "+str(views_for_week)+", ImageLink = '"+channel_id +"', ER = "+str(er)+" WHERE Id = '"+channel_id_sql+"';"
print(query)
cursor.execute(query)
cursor.commit()
client.start()
client.run_until_disconnected()
```

ChannelController:

```
[Route("api/[controller]")]
[ApiController]
public class ChannelController : ControllerBase
{
private readonly IChannelService _channels;
private readonly IMapper _mapper;
public ChannelController(IChannelService channels, IMapper mapper)
{
_channels = channels;
_mapper = mapper;
}
[HttpGet("Read")]
public ActionResult<IEnumerable<ChannelDto>> Get()
{
var channels = _channels.GetChannels();
```

```
return Ok(channels);
}
[HttpGet("ReadById/{id}")]
public ActionResult<ChannelDto> Get(Guid id)
{
var channel = _channels.GetChannelById(id);
if (channel == null)
{
return NotFound();
}
return Ok(_mapper.Map<ChannelDto>(channel));
}
[HttpGet("ReadByUserId/{userId}")]
public ActionResult<IEnumerable<ChannelDto>> GetByUser(Guid userId)
{
var channels = _channels.GetChannelsByUserId(userId);
return Ok(channels);
}
[HttpPost("Create")]
public ActionResult Post([FromBody] Newtonsoft.Json.Linq.JObject data)
{
var channel = data["channel"].ToObject<Channel>();
var ids = data["ids"].ToObject<string[]>();
var prices = data["prices"].ToObject<Price[]>();
if (ModelState.IsValid)
{
var item = _channels.CreateChannel(channel, ids, prices);
return Ok(item);
}
return BadRequest(ModelState);
}
```

```
[HttpPost("Update")]
public IActionResult Put([FromBody] Newtonsoft.Json.Linq.JObject data)
{
var channel = data["channel"].ToObject<ChannelDto>();
var ids = data["ids"].ToObject<string[]>();
var prices = data["prices"].ToObject<Price[]>();
if (ModelState.IsValid)
{
var _channel = _mapper.Map<Channel>(channel);
var res = _channels.UpdateChannel(_channel, ids, prices);
if (res != null) {
return StatusCode((int)HttpStatusCode.NoContent);
}
else
{
return BadRequest("You can modify only your channels");
}
}
return BadRequest(ModelState);
}
[HttpPost("UpdateStatus")]
public IActionResult PutStatus(Channel channel)
{
if (ModelState.IsValid)
{     _hchchannel = _mapper.Map<Channel>(channel);
      _channels.UpdateStatus(_channel);
      return StatusCode((int)HttpStatusCode.NoContent);
      }
      return BadRequest(ModelState);
      }
```

```
[HttpDelete("Delete/{id}")]
public IActionResult Delete(Guid id)
{
_channels.Delete(id);
return StatusCode((int)HttpStatusCode.NoContent);
}
[HttpGet("GetMaxPrice")]
public int GetMaxPrice()
{
var maxPrice = _channels.GetMaxPrice();
return maxPrice;
}
[HttpGet("GetVariantsHours")]
public IEnumerable<object> GetVariantsHours()
{
var variants = _channels.GetVariantsHours();
return variants;
}
[HttpGet("GetMaxSalePrice")]
public int GetMaxSalePrice()
{
var maxPrice = _channels.GetMaxSalePrice();
return maxPrice;
}
[HttpGet("GetMaxViews")]
public int GetMaxViews()
{
var maxViews = _channels.GetMaxViews();
return maxViews;
}
[HttpGet("GetMaxSubscribers")]
```

```
public int GetMaxSubscribers()
{
var maxSubscribers = _channels.GetMaxSubscribers();
return maxSubscribers;
}
```

PostController:

```
[Route("api/[controller]")]
[ApiController]
public class PostController : ControllerBase
{
private readonly IPostService _post;
private readonly IMapper _mapper;
public PostController(IPostService purchasepost, IMapper mapper)
{
_post = purchasepost;
_mapper = mapper;
}
[Authorize]
[HttpPost("Create")]
public IActionResult CreatePost([FromBody] Newtonsoft.Json.Linq.JObject data)
{
var post = data["post"].ToObject<PostDto>();
var post_ = _mapper.Map<Post>(post);
var ids = data["ids"].ToObject<string[]>();
return Created("", _post.CreatePost(post_, ids));
}
[HttpGet("Read")]
public ActionResult<IEnumerable<PostDto>> Get()
{
var posts = _post.GetPosts();
return Ok(posts);
```

```
}
[HttpGet("ReadById/{id}")]
public IActionResult GetById(Guid id)
{
var post = _post.GetPostById(id);
return Ok(_mapper.Map<IEnumerable<PostDto>>(post));
}
[HttpGet("ReadByUserId/{userId}")]
public ActionResult<IEnumerable<PostDto>> GetByUser(Guid userId)
{
var posts = _post.GetPostsByUserId(userId);
return Ok(posts);
}
[HttpPost("UpdateStatus")]
public IActionResult PutStatus(Post post)
{
if (ModelState.IsValid)
{
var post_ = _mapper.Map<Post>(post);
_post.UpdateStatus(post_);
return StatusCode((int)HttpStatusCode.NoContent);
}
return BadRequest(ModelState);
}
[HttpGet("GetMaxPrice")]
public int GetMaxPrice()
{
var maxPrice = _post.GetMaxPrice();
return maxPrice;
}
[Authorize]
```

```
[HttpPost("Update")]
public IActionResult UpdatePost([FromBody] Newtonsoft.Json.Linq.JObject data)
{
var post = data["post"].ToObject<PostDto>();
var post_ = _mapper.Map<Post>(post);
var buttons = data["buttons"].ToObject<IEnumerable<InlineButton[]>>();
return StatusCode((int)HttpStatusCode.NoContent, _post.UpdatePost(post_, buttons));
}
[HttpPost("Delete/{id}")]
public IActionResult DeleteReq(Guid id)
{
_post.Delete(id);
return StatusCode((int)HttpStatusCode.NoContent);
}
}
```

Додаток Е

Приклад Unit – тестів:

```
[Fact]
public void Get_WhenCalled_ReturnsOkResult()
{
// arrange
var service = new Mock<IChannelService>();
var persons = GetFakeData();
service.Setup(x => x.GetChannels()).Returns(persons);
var configuration = new MapperConfiguration(cfg => cfg.AddProfile(new ApplicationProfile()));
var controller = new ChannelController(service.Object, new Mapper(configuration));
// Act
var okResult = controller.Get().Result;
// Assert
Assert.IsType<OkObjectResult>(okResult);
}
[Fact]
public void Get_WhenCalled_ReturnsAllItems()
{ // arrange
var service = new Mock<IChannelService>();
var persons = GetFakeData();
service.Setup(x => x.GetChannels()).Returns(persons);
var configuration = new MapperConfiguration(cfg => cfg.AddProfile(new ApplicationProfile()));
var controller = new ChannelController(service.Object, new Mapper(configuration));
// Act
var okResult = (controller.Get()).Result as OkObjectResult;
// Assert
var items = Assert.IsType<List<ChannelDto>>(okResult.Value);
Assert.Equal(10, items.Count);
```

```
}
[Fact]
public void Get_WhenCalled_BadRequestResult()
{
//Arrange
var service = new Mock<IChannelService>();
var persons = GetFakeData();
service.Setup(x => x.GetChannels()).Returns(persons);
var configuration = new MapperConfiguration(cfg => cfg.AddProfile(new
ApplicationProfile()));
var controller = new ChannelController(service.Object, new Mapper(configuration));
//Act
var data = controller.Get().Result;
data = null;
if (data != null)
//Assert
Assert.IsType<BadRequestResult>(data);
}
[Fact]
public void Task_GetPosts_MatchResult()
{
//Arrange
var service = new Mock<IChannelService>();
var persons = GetFakeData();
service.Setup(x => x.GetChannels()).Returns(persons);
var configuration = new MapperConfiguration(cfg => cfg.AddProfile(new
ApplicationProfile()));
var controller = new ChannelController(service.Object, new Mapper(configuration));
//Act
var okResult = (controller.Get()).Result as OkObjectResult;
//Assert
```

```
var items = Assert.IsType<List<ChannelDto>>(okResult.Value);
Assert.Equal(Guids[0], items[0].Id);
Assert.Equal("Channel 1", items[1].Name);
}
[Fact]
public void GetById_ExistingGuidPassed_ReturnsOkResult()
{
// Arrange
var service = new Mock<IChannelService>();
var persons = GetFakeData();
service.Setup(x => x.GetChannelById(Guids[0])).Returns(persons.First());
var configuration = new MapperConfiguration(cfg => cfg.AddProfile(new
ApplicationProfile()));
var controller = new ChannelController(service.Object, new Mapper(configuration));
// Act
var okResult = (controller.Get(Guids[0])).Result as OkObjectResult;
var item = Assert.IsType<ChannelDto>(okResult.Value);
// Assert
Assert.IsType<O
kObjectResult>(
okResult);
}
```