Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА TELEGRAM MINI APP ДЛЯ ПОШУКУ КВАРТИР

Текстова частина до курсової роботи за спеціальністю “Інженерія програмного забезпечення” - 121

Керівник курсової роботи
с.в. Борозенний С.О.
_____
(підпис)
“____”_____ 2025 р.
Виконав студент
Швачка Д. І.
“____”_____ 2025 р.

Київ 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
к.т.н., доцент Жежерун О.П.
„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Швачка Денису Ігоровичу факультету інформатики 3 курсу

ТЕМА: Розробка Telegram Mini App для пошуку квартир

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план виконання роботи
Анотація
Вступ
Розділ 1. Аналіз предметної області та існуючих рішень
Розділ 2. Технології Telegram та їх застосування
Розділ 3. Розробка додатку
Висновки
Використані джерела

Дата видачі
«____» _____ 2024 р.
Керівник _____
_____
(прізвище та ініціали)
_____
(підпис)
Завдання отримав _____
_____
(прізвище та ініціали)
_____
(підпис)

Тема: Розробка Telegram Mini App для пошуку квартир

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Вибір теми індивідуального завдання.	02.10.2024	
2.	Отримання завдання на курсову роботу.	09.10.2024	
3.	Пошук літератури за темою роботи.	20.10.2024	
4.	Огляд технічної літератури за темою роботи.	29.10.2024	
5.	Проектування структури проєкту.	03.11.2024	
6.	Дослідження літератури з теми парсингу даних та відповідних інструментів.	05.11.2024	
7.	Розробка парсера.	20.11.2024	
8.	Дослідження можливостей технології Telegram Mini App.	03.02.2025	
9.	Розробка парсинг-сервера.	15.02.2025	
10.	Огляд технічної літератури за бібліотеки для бекенд частини.	27.02.2025	
11.	Розробка серверної частини додатку.	13.03.2025	
12	Ознайомлення з технічною літератури Vue, огляд документації Telegram WebApp.	22.03.2025	
13.	Розробка веб частини додатку.	29.03.2025	
14.	Написання текстової частини.	12.04.2025	
15.	Створення презентації.	02.05.2025	

Студент Швачка Д. І.

Керівник_____. “_____”_____

ЗМІСТ

Перелік прийнятих скорочень

API (Application Programming Interface) – програмний інтерфейс додатку;

TG / ТГ (Telegram) - Телеграм;

TMA (Telegram Mini App) – Міні-додаток Telegram;

TWA (Telegram Web App) - синонім до TMA;

БД (База Даних) – Database;

JS (JavaScript) – Мова програмування JavaScript;

UI (User Interface) – Інтерфейс користувача;

UX (User Experience) – Досвід користувача;

HTTP (HyperText Transfer Protocol) – Протокол передачі гіпертексту;

JSON (JavaScript Object Notation) – Текстовий формат обміну даними;

CRUD (Create, Read, Update, Delete) – Базові операції з даними;

Анотація

Метою курсової роботи є дослідження та демонстрація можливостей технології Telegram Mini Apps шляхом розробки прикладного застосунку для пошуку оренди квартир. Застосунок повністю інтегрований у платформу Telegram та надає користувачам можливість зручно шукати актуальні оголошення про оренду нерухомості, використовуючи власні фільтри. Дані для додатку агрегуються з сайту оренди за допомогою розробленого парсера.

У роботі проведено аналіз ринку додатків для пошуку нерухомості, розглянути можливості Telegram Bot Api та Telegram Mini Apps, спроєктовано архітектуру системи та обґрунтовано вибір технологічного стеку. Описано процес розробки серверної частини, клієнтського Mini App, інтеграції з TG, бази даних та модуля парсингу.

Вступ

Месенджер Telegram стрімко розвивається: будучи на початку простим засобом перебування, він перетворився на багатофункціональну платформу, що вже більше за “просто месенджер”.

Одним із ключових напрямків цього розвитку є технологія Telegram Mini Apps (TMA), що дозволяє розробникам створювати повноцінні веб-додатки, які можна запустити безпосередньо всередині месенджера. Це відкриває нові можливості для бізнесу та розробників, дозволяючи створювати інтегровані сервіси, ігри, інструменти та цілі екосистеми, доступні мільйонам користувачів Telegram без необхідності встановлення окремих додатків. Так як люди більшість часу можуть проводити саме в месенджері, то можна сказати, що ці застосунки “завжди під рукою”. Актуальність дослідження можливостей та процесу розробки TMA зумовлена зростаючим попитом на зручні та безшовні рішення, вбудовані в популярні платформи, а також необхідністю вивчення найкращих практик та підходів до створення таких додатків.

Telegram Mini App це сайт, який можно відкрити додатком Telegram у боті. Але завдяки інтеграції з месенджером він перестає бути звичайним сайтом: найголовнішим бонусом є те, що сайт отримує дані про користувача, його ID, ім’я, username, фотографії тощо. Завдяки цьому виконується безшовна автентифікація користувача, що дозволяє оминути процес авторизації та зменшує бар’єр входу для користувача та спрощує розробку. Користувач економить час, а розробникам легше взаємодіяти. з користувачами.

Окрім цього, Telegram Mini App може отримувати інші дані зі застосунку месенджера, наприклад, кольорову тему. Це дозволяє робити дизайн, що буде цілісним з Telegram. Також є можливість повної взаємодії з застосунком через сайт: відкриття та закриття вікна у Telegram, можливість робити callback для операції боту на сервері, взаємодіяти зі

сховищем користувача, ділитись повідомленнями в чати чи історії, робити розрахунки тощо.

Моєю метою є продемонструвати процес розробки та ключові можливості платформи Telegram Mini Apps на прикладі створення застосунку для пошуку оренди квартир та показати переваги інтеграції веб-додатків у середовище месенджера.

Приклад застосунку для пошуку квартир обрано як достатньо комплексний сценарій, що дозволяє показати широкий спектр можливостей Telegram Mini App та Telegram Bot Api. У проєкті присутня робота з базою даних, фільтрація, сповіщення, списки, взаємодія з користувачем. Також був розроблений парсер оголошень, що парсить найновіші оголошення, що дозволить користувачам отримувати сповіщення про нові квартири одними з перших.

Результати дослідження та описаний у роботі процес розробки можуть бути корисними для веб-розробників, які планують створювати додатки для платформи Telegram.

РОЗДІЛ 1: Аналіз предметної області та існуючих рішень

1.1 Постановка задачі для демонстраційного застосунку

Пошук житла для оренди є актуальною задачею для багатьох людей. Традиційно цей процес відбувається через спеціалізовані веб-сайти (дошки оголошень), агенції нерухомості або соціальні мережі. Однак користувачі часто стикаються з певними труднощами: необхідність моніторити декілька ресурсів одночасно, інформаційне перевантаження через велику кількість нерелевантних пропозицій, та потреба у швидкому реагуванні на нові оголошення, особливо на конкурентному ринку.

Існуючі рішення, такі як веб-сайти та окремі мобільні додатки, надають широкий функціонал, але вимагають від користувача активного пошуку, реєстрації на різних платформах та встановлення додаткових програм. Сповіщення про нові варіанти часто надходять із затримкою або губляться серед інших повідомлень.

Саме ця задача – пошук та відстеження актуальних оголошень про оренду квартир – була обрана як приклад для розробки демонстраційного Telegram Mini App у рамках даної курсової роботи. Вибір зумовлений тим, що цей сценарій дозволяє продемонструвати широкий спектр можливостей платформи TMA:

1. **Робота з даними:** Отримання, відображення та фільтрація списку оголошень.
2. **Інтеграція з Telegram:** Використання безшовної авторизації (отримання ID користувача для збереження налаштувань, обраного, підписок), адаптація інтерфейсу до теми Telegram.
3. **Взаємодія з ботом:** Можливість налаштування підписок та отримання миттєвих сповіщень про нові релевантні оголошення безпосередньо у месенджері.

4. **Зручність для користувача:** Надання доступу до функціоналу пошуку житла "під рукою", всередині звичного інтерфейсу Telegram, без необхідності встановлення окремого додатку чи реєстрації на окремих сайтах.

Таким чином, розробка Mini App для пошуку квартир слугує не метою створення конкурентного продукту на ринку нерухомості, а практичним полігоном для дослідження, реалізації та демонстрації ключових переваг та технічних аспектів технології Telegram Mini Apps. Це дозволяє показати, як інтеграція веб-додатку в месенджер може спростити вирішення повсякденних задач користувача.

1.2 Аналіз існуючих підходів та платформних рішень

Для вирішення задачі пошуку оренди нерухомості існує кілька основних підходів та платформних рішень, кожне з яких має свої особливості, переваги та недоліки у порівнянні з моделлю інтегрованого в месенджер Mini App.

1. **Традиційні веб-платформи та мобільні додатки**

Платформи, такі як OLX (розділ Нерухомість), ЛУН, Rieltor.ua, Dom.ria, є основними гравцями на ринку пошуку нерухомості в Україні. Вони пропонують великі бази оголошень, розширені можливості фільтрації, інтерактивні карти та власні мобільні додатки.

Переваги: Широке охоплення ринку, деталізований функціонал, звичний для багатьох користувачів інтерфейс.

Недоліки з точки зору інтеграції:

- **Окремі платформи**

  Вимагають від користувача відвідування зовнішнього веб-сайту або встановлення та запуску окремого мобільного додатку.

- **Реєстрація/Авторизація**

  Часто потребують створення окремого облікового запису, що є додатковим кроком для користувача.
- **Сповіщення**

  Нотифікації про нові оголошення надходять через email або push-сповіщення від окремого додатку, що може бути менш зручним та оперативним, ніж сповіщення безпосередньо у месенджері, де користувач проводить значну частину часу.

2. **Telegram-боти**

Існують також рішення, що використовують Telegram Bot API для створення ботів, за допомогою яких користувачі можуть шукати квартири. Ось приклади, які мені вдалось відшукати:

Flated Bot (TG: @flated_bot): Цей бот моніторить OLX та Dom.ria і надсилає користувачам оголошення за їхніми критеріями. Цікаво, що він використовує Telegram Mini App, але тільки як форму для налаштування фільтрів підписки, тоді як основний перегляд оголошень та взаємодія відбуваються через стандартні повідомлення та кнопки бота за допомогою Telegram Bot API. Це демонструє часткову інтеграцію TMA, але не реалізує потенціал повноцінного веб-інтерфейсу всередині Telegram для перегляду та взаємодії з оголошеннями. Але це вже є гарним прикладом того, як за допомогою TMA можна створювати гарну та зручну форму для отримання даних від користувача.

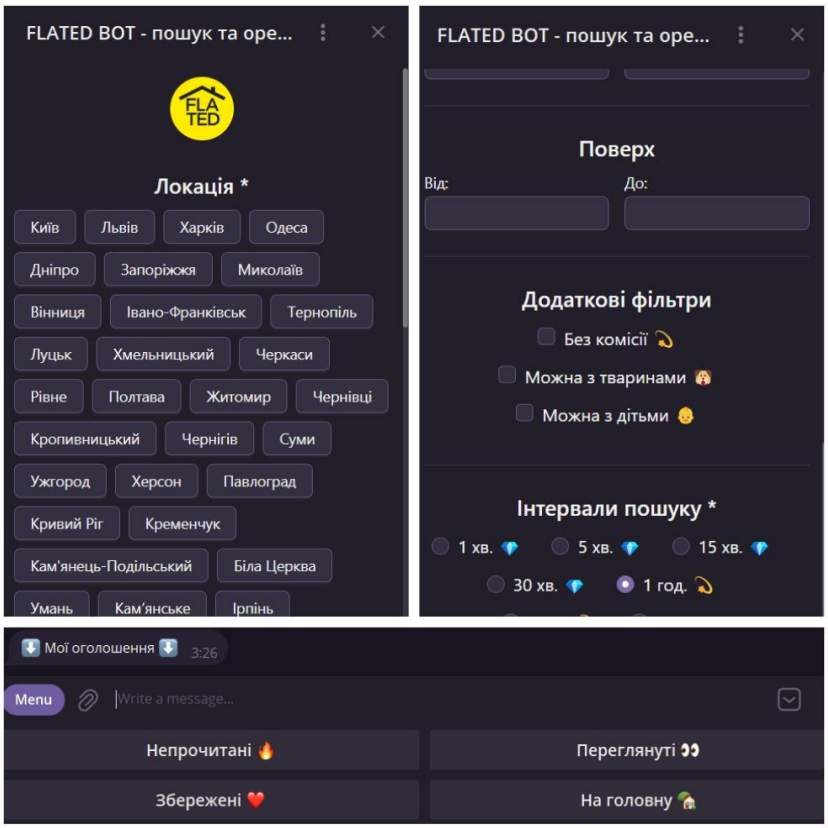


Рисунок 1.1 - Демонстрація Flated.

Findly Chatbot (TG: @findly_chatbot): Цей бот фокусується на пошуку оголошень без комісії та використовує виключно можливості Telegram Bot API (команди, текстові повідомлення, кнопки), не застосовуючи технологію Mini Apps. На мою думку, він шукає оголошення без комісії використовуючи відповідні параметри в оголошеннях (“Без комісії”) та за ключовими словами в описі (“Без комісії”, “Хазяїн”, “Власник”). Це класичний приклад "чистого" бота, інтерфейс та можливості взаємодії якого обмежені стандартними елементами Telegram чату.

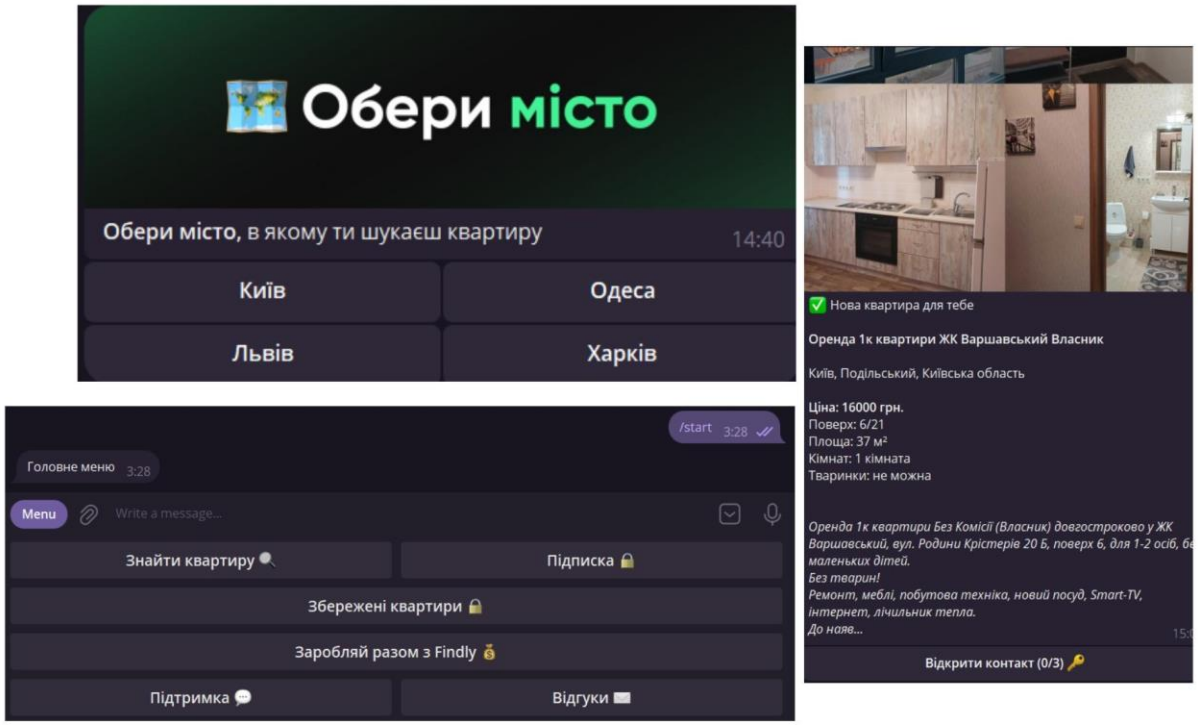


Рисунок 1.2 - Демонстрація Findly.

Традиційні платформи пропонують багатий функціонал, але програють у зручності інтеграції та оперативності сповіщень порівняно з рішеннями всередині месенджера. Існуючі Telegram-боти або використовують TMA обмежено (як Flated Bot), або не використовують зовсім (як Findly), покладаючись на стандартний інтерфейс Bot API.

Це створює нішу для дослідження та демонстрації повноцінного Telegram Mini App, який би поєднував:

1. Багатство та інтерактивність веб-інтерфейсу (зручні списки, фільтри, картки оголошень).
2. Переваги глибокої інтеграції з Telegram (безшовна авторизація, використання UI-компонентів та теми месенджера, миттєві сповіщення через бота).

1.3 Вимоги до демонстраційного застосунку

Застосунок має бути повноцінним та працюючим як справжній продукт, що готовий для споживання користувачами. Метою є не створення конкурентно здатного продукту, а застосунку, що покаже на гарному прикладі можливости Telegram Mini Apps. Можна виділити наступні основні вимоги:

1. **Агрегація даних**

Система повинна мати можливість збирати дані про оголошення оренди квартир з зовнішнього джерела, наприклад сайтів оголошень, та зберігати їх у власній базі даних. Так як ніякої можливості отримати безпосередньо дані від сайтів за допомогою API немає, то залишається тільки парсинг.

2. **Відображення даних у TMA**

Клієнтська частина (Mini App) повинна відображати отримані оголошення у зручному для мобільних пристроїв форматі списку з можливістю перегляду деталей окремого оголошення.

3. **Пошук та фільтрація**

Mini App повинен надавати користувачеві інструменти для пошуку та фільтрації оголошень за ключовими параметрами (наприклад, місто, ціновий діапазон, кількість кімнат, можливість заселитись з тваринкою чи дитиною).

4. **Інтеграція з Telegram - Авторизація**

Застосунок повинен використовувати механізм безшовної авторизації Telegram, отримуючи дані про користувача для його ідентифікації без необхідності окремої реєстрації.

5. **Інтеграція з Telegram - Персоналізація:**

Система повинна зберігати дані, специфічні для користувача (збережені фільтри/підписки, обрані оголошення), прив'язуючи їх до Telegram ID користувача.

6. **Інтеграція з Telegram - UI**

Інтерфейс Mini App повинен адаптуватися до поточної теми оформлення Telegram користувача, забезпечуючи візуальну цілісність. За можливості використовувати нативні елементи Telegram.

7. **Взаємодія з ботом**

Повинна бути реалізована можливість для користувача створювати підписки на пошукові запити через Mini App. Бот, у свою чергу, повинен надсилати користувачеві сповіщення у Telegram при появі нових оголошень, що відповідають його підписці.

8. **Можливість ділитись**

У користувача має бути можливість поширювати певні дані, наприклад, оголошення квартири або рієлтора, безпосередньо завдяки можливостям Telegram, не копіюючи текст.

9. **Безпека**

Так як проєкт розрахований виключно для користувачів Telegram, переважно з мобільних пристроїв, потрібно забезпечити безпеку застосунку, щоб не було можливості допустити несанкціонований доступ ззовні середовища Telegram.

1.4 Функціональні можливості демонстраційного застосунку як об'єкта дослідження

На основі сформульованих вимог, демонстраційний застосунок, розроблений у цій роботі, реалізує наступний основний функціонал, що слугує об'єктом дослідження технології TMA:

1. **Запуск та Авторизація**

Користувач запускає Mini App через кнопку або посилання в Telegram боті. Застосунок автоматично отримує дані користувача для його ідентифікації та завантаження персональних налаштувань. Якщо

користувача не знайдено, то сайт не відповідає користувачу, бо це значить, що користувач не з середовища Telegram.

2. **Перегляд та Фільтрація**

Mini App відображає список оголошень квартир, отриманих з backend API (які, у свою чергу, наповнюються парсером). Користувач може застосовувати фільтри (місто, район, ціна, кімнати тощо) для звуження пошуку. Інтерфейс адаптується до теми Telegram.

3. **Збереження Обраного**

Користувач має можливість додати оголошення, що сподобались, до списку "Обране". Ця інформація зберігається у базі даних, прив'язаній до Telegram ID користувача.

4. **Створення Підписки**

Користувач може зберегти поточні налаштування фільтрів як підписку. Дані про підписку також зберігаються у базі даних.

5. **Сповіщення через Бота**

Серверна частина періодично перевіряє наявність нових оголошень, що відповідають активним підпискам користувачів. У разі знаходження збігів, Telegram бот надсилає користувачеві сповіщення з посиланням на нове оголошення або пропозицією відкрити Mini App. Там користувач може переглянути оголошення та, якщо сподобається, зберегти його в обрані

6. **Додаткові функції**

Додатковий функціонал для демонстрації можливостей TMA, що можуть бути корисні на прикладі цього застосунку. Наприклад, ділитись оголошеннями квартир або договорами оренди у чатах.

Цей набір функцій дозволяє комплексно продемонструвати життєвий цикл взаємодії користувача з Telegram Mini App: від безшовного входу та навігації по веб-інтерфейсу до асинхронної взаємодії через Telegram бота для отримання актуальної інформації. Реалізація цих можливостей слугує

основою для аналізу переваг та особливостей розробки на платформі Telegram Mini Apps у наступних розділах роботи.

РОЗДІЛ 2: Технології Telegram та їх застосування

2.1. Огляд Telegram Bot API

Telegram Bot API це справжня фундаментальна технологія без якої неможливо створити жодного функціонуючого бота. Це HTTP-інтерфейс, що дозволяє розробникам створювати програми (ботів), що можуть отримувати повідомлення від користувачів та відповідати на них, виконувати команди, надсилати різноманітний контент та інтегруватися з іншими сервісами. По суті, Bot API виступає мостом між вашим серверним застосунком (де живе логіка бота) та клієнтами Telegram.

Взаємодія з Bot API відбувається через надсилання HTTPS запитів до сервера Telegram за адресою *https://api.telegram.org/*. Кожен запит містить унікальний токен бота та викликає певний метод API з необхідними параметрами. Відповіді надходять у форматі JSON.

Звідси постає питання: “Як наш бот розуміє, що тільки що був надісланий запит з його токеном?”. Для цього існує два основні способи отримання ботом інформації від користувачів (оновлень - Updates):

1. **Long Polling**

Ваш бот періодично надсилає запит *getUpdates* до сервера Telegram, "зависаючи" на певний час (long polling). Якщо за цей час надходять нові оновлення (повідомлення, натискання кнопок тощо), сервер негайно повертає їх боту. Якщо оновлень немає, сервер повертає порожню відповідь після тайм-ауту, і бот робить новий запит. Це простіший спосіб для початку розробки, але в експлуатації може бути затратним через постійні запити від бота до Telegram.

2. **Webhooks**

Для цього потрібно зареєструвати URL для HTTPS-сервера (вебхук) у Telegram. Коли надходить нове оновлення для вашого бота, сервер Telegram негайно надсилає HTTPS POST-запит з цим оновленням (у форматі JSON) на ваш зареєстрований URL. Цей метод є більш ефективним та рекомендованим для ботів з високим навантаженням, хоч і складнішим у реалізації.

Ключові компоненти та концепції:

1. **Токен Бота (Bot Token)**

Унікальний рядок символів, що слугує для автентифікації вашого бота при зверненні до API. Видається спеціальним ботом @BotFather при створенні нового бота. Токен є конфіденційною інформацією і має зберігатися в безпеці, аналогічно до пароля чи API-ключа. Використовується у кожному запиті до API у наступному форматі: *https://api.telegram.org/bot<token>/<METHOD_NAME>*.

2. **Оновлення (Updates)**

Об'єкти у форматі JSON, які містять інформацію про події, що стосуються бота. Це можуть бути нові повідомлення (текстові, фото, документи), натискання на inline-кнопки (*callback_query*), додавання бота до групи, редагування повідомлень тощо. Кожен тип події має свою структуру даних у межах об'єкта *Update*. Містить у собі також інформацію про користувача та номер чату з його надійшла відповідь.

3. **Методи (Methods)**

Спеціальні команди, які бот може викликати для виконання певних дій. Кожен метод має свою назву (наприклад, *sendMessage, sendPhoto, editMessageText, answerCallbackQuery*. Більшість з назв легко демонструє те, що виконує метод, а для чогось треба розібратись у документації) та набір обов'язкових і необов'язкових параметрів. Наприклад, метод *sendMessage* вимагає вказати *chat_id* (ідентифікатор чату, куди надсилати) та text (текст повідомлення), а також може приймати необов'язкові

параметри, як-от *reply_markup* для додавання клавіатури. І це логічно, бо для того, щоб відправити, обов’язково потрібно місце призначення та те, що надсилати, а вже без додаткових компонентів можно й обійтись.

4. **Ідентифікатори (IDs)**

Унікальні числові значення, що ідентифікують користувачів (*user_id*), чати (*chat_id*) та повідомлення (*message_id*). Вони є ключовими для адресації повідомлень та дій конкретним користувачам чи у конкретних чатах, бо інакше неможливо зрозуміти куди бот має надіслати повідомлення.

5. **Клавіатури (Keyboards)**

Інтерфейсні елементи, що дозволяють користувачам взаємодіяти з ботом не лише за допомогою тексту, а й за допомогою певних кнопок. Існує два основних типи клавіатур:

1. **ReplyKeyboardMarkup**

   Замінює стандартну клавіатуру користувача на набір кнопок (*KeyboardButton*), які можуть надсилати визначений текст, запитувати номер телефону (*request_contact*) або геолокацію (*request_location*). Далі бот вже отримує надіслані дані та обробляє їх за своєю логікою.

2. **InlineKeyboardMarkup**

   Набір кнопок (*InlineKeyboardButton*), що прикріплюються безпосередньо до повідомлення бота. Ці кнопки можуть містити URL-посилання (*url*), дані для зворотного виклику (*callback_data*, які бот отримує як *Update* типу *callback_query* при натисканні), перемикач в inline-режим (*switch_inline_query*) або, що найважливіше для даної роботи, посилання для запуску Mini App (*web_app*).

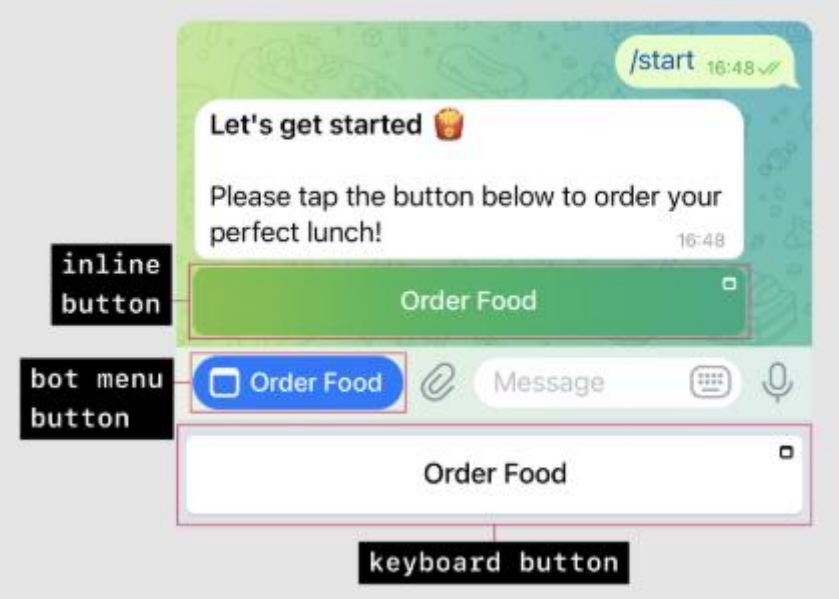


Рисунок 2.1 - Ілюстрація клавіатур

6. **Команди (Commands)**

Повідомлення, що починаються зі слеша / (наприклад, */start, /help*). Боти можуть легко розпізнавати та обробляти такі команди для виконання специфічних дій. Список команд можна налаштувати через *@BotFather* для зручності користувачів.

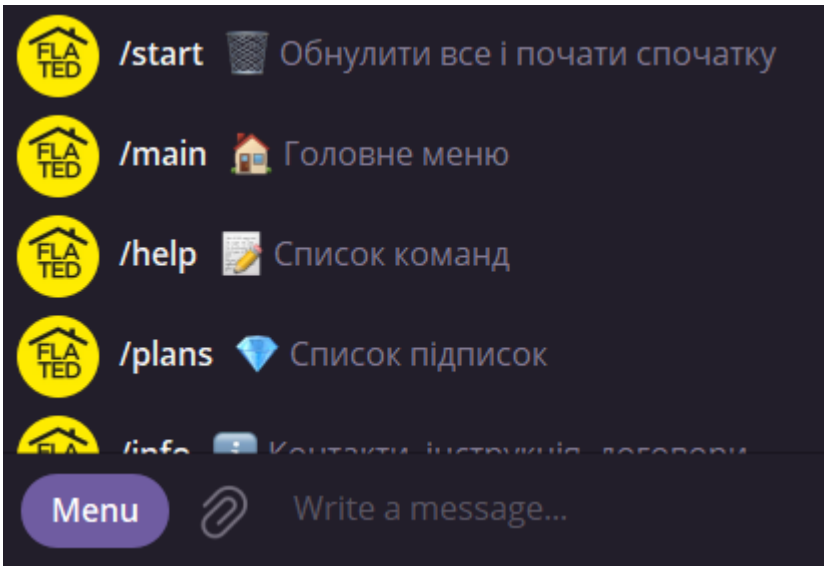


Рисунок 2.2 - Меню команд на прикладі FlatedBot

7. Обробка помилок

У разі невдалого виклику методу API, Telegram повертає JSON-об'єкт з полем *ok*, встановленим у *false*, та полями *error_code* і *description*, що пояснюють причину помилки [5].

Значення Telegram Bot Api для Telegram Mini App

TMA є веб-додатками, що працюють на стороні клієнта, але без Bot Api їх існування під загрозою, бо ця технологія відіграє критичну роль у їхньому життєвому циклі.

Наприклад, для запуску TMA є декілька способів. Але найзручнішим та найгнучкішим є спосіб через натискання на *InlineKeyboardButton* типу *web_app*, яку бот надсилає за допомогою методу (*sendMessage*, *editMessageReplyMarkup* тощо) Bot API. Ми можемо передавати посилання на застосунок у кнопку одразу з необхідними параметрами, щоб, наприклад, TMA запустилась на конкретній сторінці.

Telegram Mini App хоч і працює у браузерному середовищі, він все одно часто потребує взаємодії з серверною backend логікою. Взаємодія може відбуватись через стандартні веб-запити (наприклад, до API вашого backend-сервера), але також Mini App може надсилати дані назад до бота (або напряму на сервер через валідацію *initData*), щоб ініціювати дії на сервері або викликати методи Bot API (наприклад, надіслати сповіщення іншому користувачеві від імені бота).

Таким чином, Telegram Bot API є невід'ємною частиною екосистеми Telegram, забезпечуючи базову інфраструктуру для створення інтерактивних ботів та слугуючи точкою входу і каналом комунікації для більш складних інтерфейсів, таких як Telegram Mini Apps.

2.2. Технологія Telegram Mini Apps (Web Apps)

Telegram Mini Apps (також відомі як Web Apps або TMA) – це веб-додатки, побудовані за допомогою стандартних веб-технологій (HTML, CSS, JavaScript), які запускаються всередині Telegram у спеціальному вбудованому браузерному вікні (WebView). Ця технологія дозволяє розробникам створювати багаті, інтерактивні та гнучкі інтерфейси, які значно перевершують можливості стандартних повідомлень та кнопок бота, при цьому залишаючись тісно інтегрованими з досвідом користування Telegram [3].

Для під'єднання технології до сайту потрібно лише вставити у сторінку наступний JavaScript:

<script src="https://telegram.org/js/telegram-web-app.js"></script>

2.2.1 Принципи роботи та можливості

Архітектура та запуск

Mini App – це, по суті, звичайний веб-сайт, доступний через HTTPS. Коли користувач натискає спеціальну кнопку (*web_app*) у чаті з ботом, клієнт Telegram відкриває цей веб-сайт у вбудованому WebView.

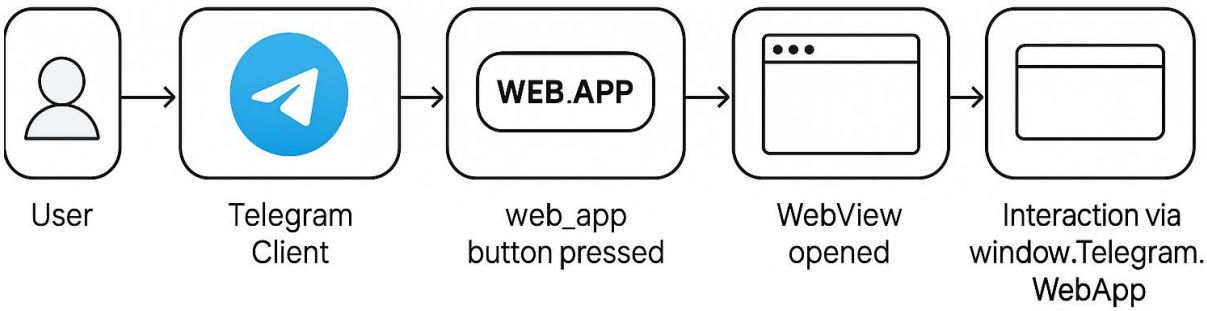


Рисунок 2.3 - Схематичне зображення архітектури TMA

1. Користувач натискає на кнопку запуску TMA. Це може бути як головна кнопка біля клавіатури вводу, як кнопка "Open App", або запуск за посиланням з тегом *start_app* (https://t.me/bot_name?startapp=@param), або за кнопкою

InlineKeyboardButton або KeyboardButton, у якої поле *web_app* містить об'єкт *WebAppInfo* з URL-адресою Mini App [1, 3].

Приклад InlineKeyboardButton для запуску TMA:

```
{
  "text": "Відкрити пошук квартир",
  "web_app": { "url": "https://your-tma-domain.com/app" }
}
```

2. Mini App виконується у ізольованому браузерному середовищі всередині Telegram. Це забезпечує доступ до сучасних веб-API, але також накладає певні обмеження з міркувань безпеки.
3. Ключовим елементом зв’язку з Telegram є JavaScript-об'єкт window.Telegram.WebApp, який надається Telegram клієнтом всередині WebView. Цей об'єкт слугує мостом між веб-додатком та нативним клієнтом Telegram, надаючи доступ до унікальних функцій та даних [3].

Основні можливості

Завдяки об'єкту window.Telegram.WebApp Mini App отримує доступ до ряду потужних функцій:

1. **Безшовна авторизація**

При запуску Mini App отримує спеціальні дані (*initData*), що містять інформацію про поточного користувача (ID, ім'я, username тощо) та чат. Ці дані підписані токеном бота, що дозволяє backend-серверу верифікувати їхню автентичність та ідентифікувати користувача без необхідності логіну/паролю [3]. Детальніше про *initData* розглянуто у підрозділі 2.2.2.

2. **Інформація про клієнт Telegram**

Доступ до версії Telegram, платформи (iOS, Android, Desktop), колірної схеми (*colorScheme*: 'light' або 'dark') та параметрів теми (*themeParams*) для адаптації UI/UX [3].

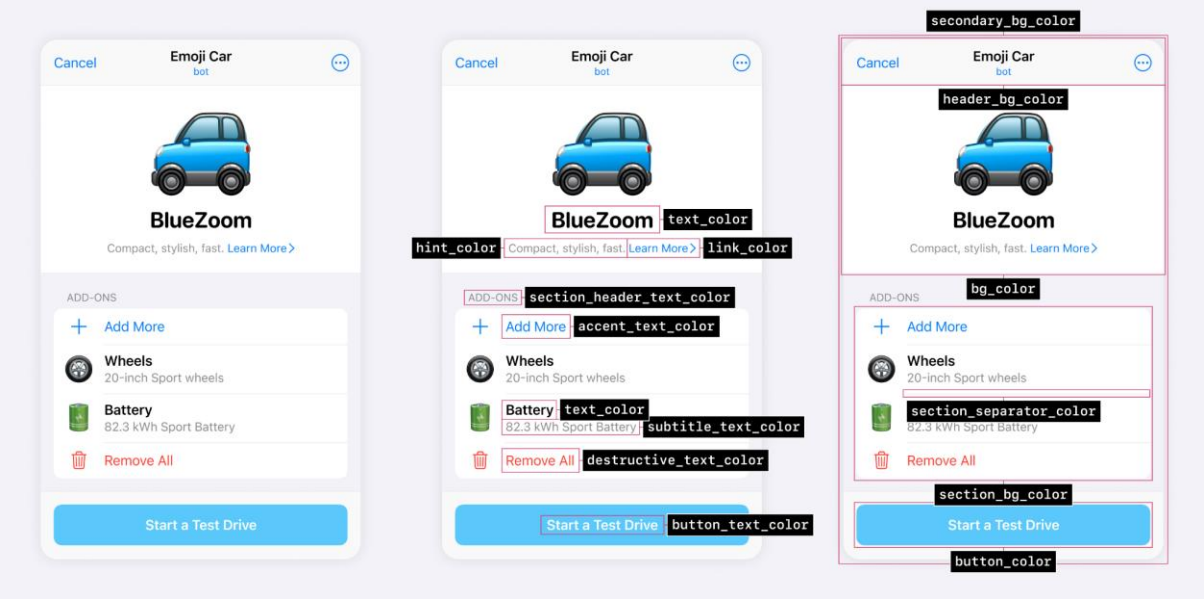


Рисунок 2.4 - Пояснення роботи WebViewColor

3. **Керування вікном TMA**

Можливість програмно розгортати (*expand()*), закривати (*close()*) вікно Mini App, а також керувати поведінкою кнопки "Назад" (*BackButton*) [3].

4. **Взаємодія з користувачем**

TMA надає можливість використовувати нативні стандартні для Telegram сповіщення (*showAlert()*), діалогів підтвердження (*showConfirm()*) та спливаючих повідомлень (*showPopup()*) [3].

Також присутня можливість додати головну кнопку (*MainButton*), що знаходиться знизу екрану, та керувати текстом, кольором, видимістю та станом активності головної кнопки, що відображається внизу екрану Mini App. Натискання на неї генерує подію, яку може обробити JavaScript код TMA [3].

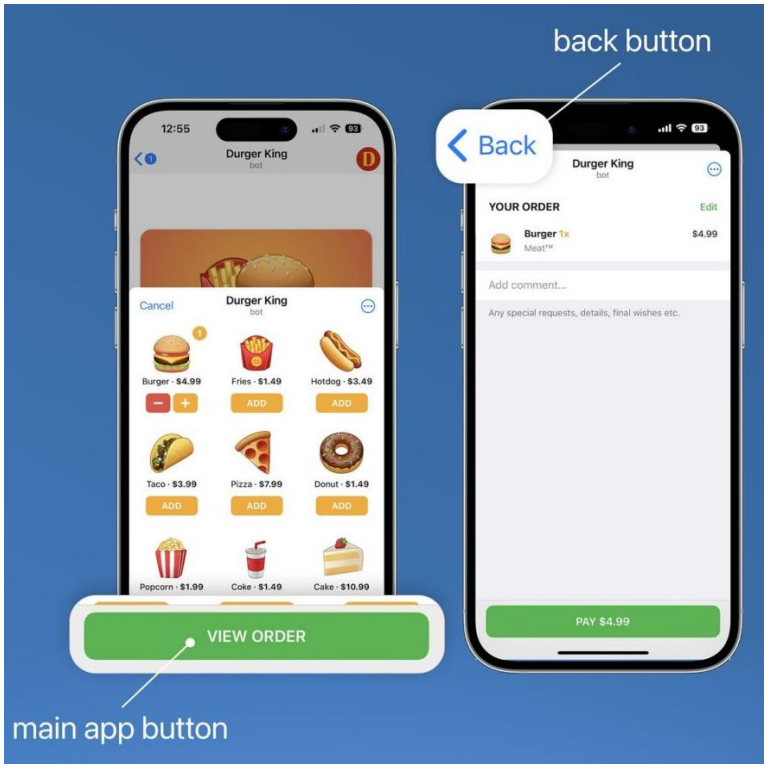


Рисунок 2.5 - Ілюстрація кнопок MainButton та BackButton у TMA

Також можна додати генерацію вібраційних відгуків для позначення важливих подій або успішних дій (*HapticFeedback*) [3].

5. **Обмін даними**

Можливість надсилати довільні дані з Mini App назад до бота або на сервер (*sendData()*). Цей метод викликає подію *web_app_data* у бота [3].

6. **Платежі**

Інтеграція з платіжною системою Telegram для продажу товарів та послуг [2].

7. **Інші можливості**

Можливість зчитувати QR-коди (*showScanQrPopup*), зчитувати вміст буферу обміну користувача (*readTextFromClipboard*), відкриття посилань у зовнішньому браузері (*openLink*) або посилань на профілі/чати всередині Telegram (*openTelegramLink*). TMA має ще безліч можливостей, але про найголовніші було зазначено [3].

Як приклад реалізації багатьох з цих можливостей можна розглянути офіційний демонстраційний бот *@DurgerKingBot* [6].

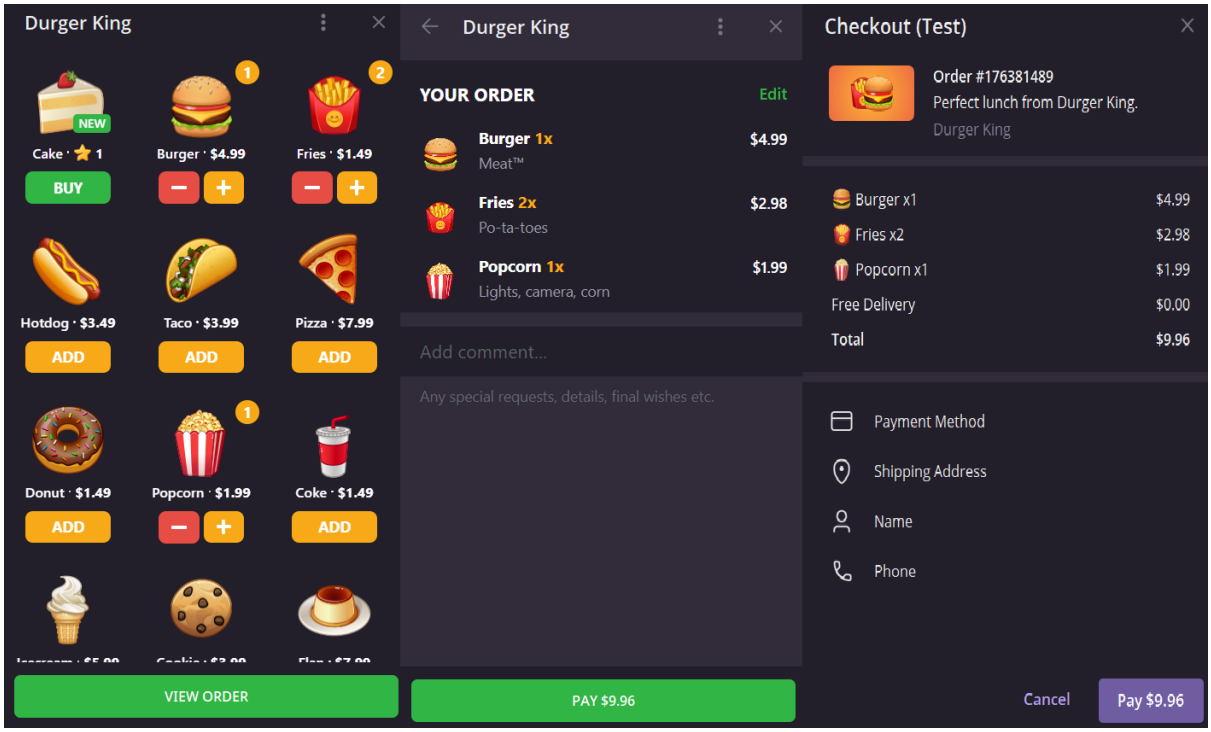


Рисунок 2.6 - Демонстрація TMA бота *@DurgerKingBot*

2.2.2. Інтеграція з ботом та інтерфейсом Telegram

Глибока інтеграція з месенджером є ключовою перевагою Telegram Mini Apps перед звичайними веб-сайтами. Ця інтеграція реалізується на кількох рівнях:

1. **Запуск та авторизація (*initData*)**

Як згадувалося, запуск відбувається через кнопку *web_app*. У момент відкриття URL, Telegram додає до URL-фрагменту *(#)* спеціальний параметр *tgWebAppData*, що містить URL-кодований рядок *initData*. Цей рядок містить дані про користувача, чат та інші параметри запуску, підписані хешем для перевірки цілісності [3].

Структура *initData* виглядає як рядок формату *query_id=...&user=...&auth_date=...&hash=....* Поле user є JSON-рядком з інформацією про користувача (id, first_name, last_name, username, language_code, is_premium тощо). Поле hash є HMAC-SHA256 хешем всіх інших параметрів, підписаним токеном бота [3].

JavaScript об'єкт window.Telegram.WebApp.initData надає доступ до цього рядка. Також можна отримати window.Telegram.WebApp.initDatUnsafe, що повертає ті ж самі значення, але у вигляді об’єкта, а не рядка. У документації дуже наполегливо рекомендують не використовувати цей об’єкт для валідації даних.

Дуже важливо перевіряти initData на сервері перед виконанням будь-яких дій, що вимагають автентифікації на бекенді. Алгоритм валідації включає перевірку хешу (hash) за допомогою секретного ключа, отриманого з токена бота. Це гарантує, що дані не були підроблені і дійсно надійшли від Telegram [3].

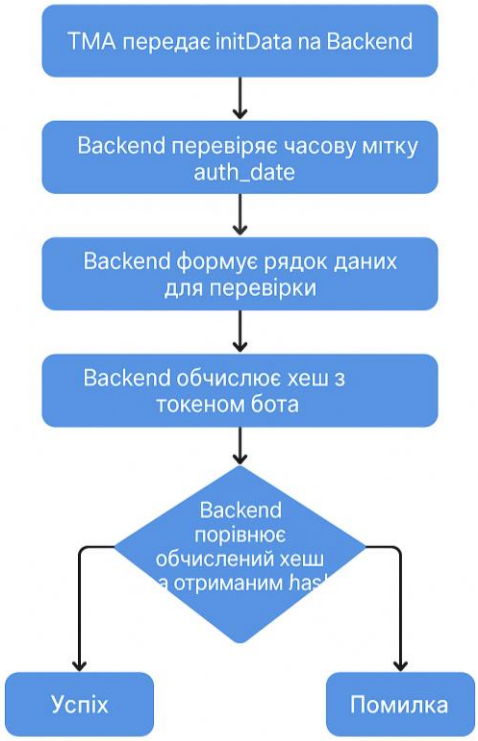


Рисунок 2.7 - Схема валідації initData

2. Інтеграція з інтерфейсом Telegram

TMA може адаптуватися до вигляду та поведінки нативного клієнта Telegram:

- **Колірна схема та параметри теми**

window.Telegram.WebApp.colorScheme ('light' або 'dark') та *window.Telegram.WebApp.themeParams* (об'єкт з кольорами *bg_color*, *text_color*, *button_color* тощо) дозволяють Mini App динамічно змінювати свій CSS для відповідності темі користувача [3]. Це створює відчуття цілісності інтерфейсу.

Можна отримати безпосередньо з об’єкту WebApp: *window.Telegram.WebApp.themeParams.text_color*

Можна використовувати відповідну CSS змінну: *var(--tg-theme-bg-color)*

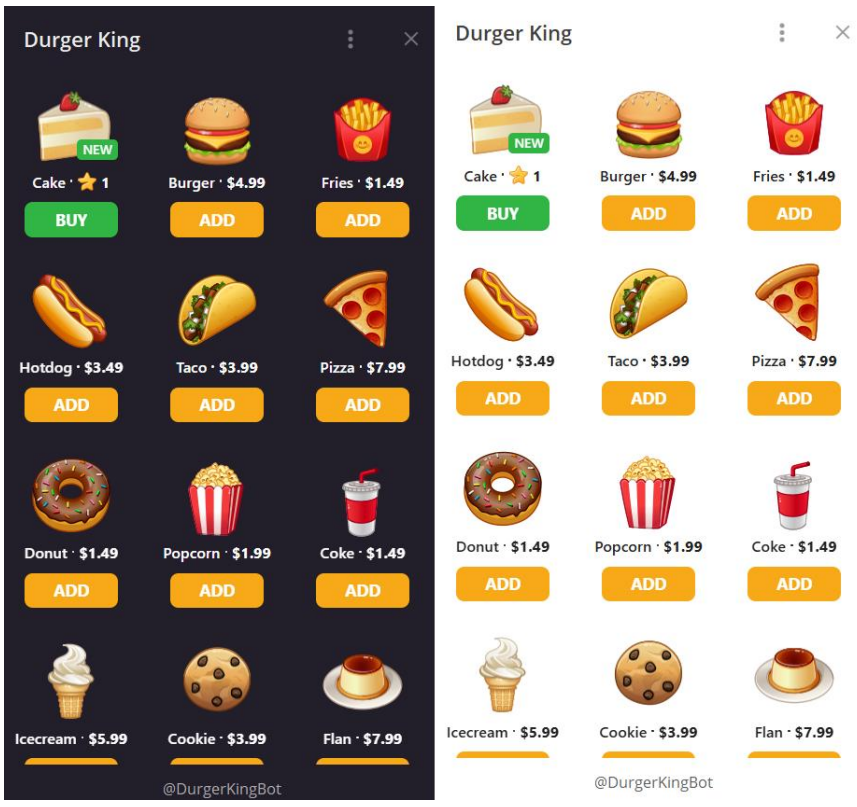


Рисунок 2.8 - Порівняння вигляду одного й того ж TMA у світлій та темній темах Telegram

- **Керування кольорами**

Методи *setHeaderColor()* та *setBackgroundColor()* дозволяють динамічно змінювати колір шапки та фону вікна TMA [3].

- **Нативні UI елементи**

Використання *showAlert(), showConfirm(), showPopup()* замість стандартних браузерних *alert(), confirm()* чи кастомних веб-компонентів робить взаємодію більш звичною для користувача Telegram.

- **Головна кнопка (MainButton)**

Ця кнопка, керована з TMA, дозволяє виконувати основну дію поточного екрану (наприклад, "Зберегти", "Відправити", "Оплатити"). Її зовнішній вигляд (текст, колір) та стан (активна/неактивна, видима/невидима, показ індикатора завантаження) повністю контролюються з JavaScript коду Mini App. Натискання генерує подію *mainButtonClicked* [3].

- **Кнопка "Назад" (BackButton)**

Mini App може керувати видимістю цієї кнопки та призначати обробник на її натискання (подія *backButtonClicked*), що дозволяє реалізувати власну логіку навігації всередині TMA [3].

3. Взаємодія з ботом та сервером

- **Надсилання даних боту (sendData)**

Метод *window.Telegram.WebApp.sendData(data)* надсилає рядок *data* боту. Бот отримує це як *Update* з полем *web_app_data*, що містить цей рядок. Це зручно для простих дій, які має виконати бот у відповідь на дії користувача в TMA (наприклад, підтвердження замовлення) [3].

- **Пряма взаємодія з Backend API**

Для більш складних операцій Mini App може напряму звертатися до власного backend API (розробленого, наприклад, на Node.js/Express, як у даній роботі), використовуючи стандартні *fetch* або *axios*. У цьому випадку *initData* передається в заголовках або тілі запиту для автентифікації та

авторизації користувача на сервері після валідації. Це основний спосіб роботи з даними (CRUD операції) у комплексних TMA.

Ці механізми інтеграції дозволяють створювати потужні та зручні додатки, що органічно вписуються в екосистему Telegram, надаючи користувачам значно кращий досвід, ніж просто перехід на зовнішній веб-сайт. У наступних розділах буде детально розглянуто, як ці можливості були використані при розробці демонстраційного застосунку для пошуку квартир.

2.3 Обґрунтування вибору Telegram Mini App для задачі

Вибір технології Telegram Mini Apps (TMA) для реалізації демонстраційного застосунку пошуку квартир у рамках даної курсової роботи зумовлений низкою переваг цієї платформи, які дозволяють ефективно продемонструвати сучасні підходи до інтеграції веб-сервісів у месенджери.

По-перше, задача пошуку квартир є вдалим демонстраційним прикладом. Вона включає типові для багатьох веб-сервісів елементи: роботу з великими обсягами даних (списки оголошень), необхідність фільтрації та пошуку, збереження персональних налаштувань користувача (фільтри, обране), та асинхронні сповіщення. Така комплексність дозволяє показати широкий спектр можливостей TMA, від відображення динамічних списків до взаємодії з базою даних та Telegram Bot API.

По-друге, TMA забезпечує високий рівень інтеграції з Telegram, що є ключовим аспектом дослідження. На відміну від окремого веб-сайту чи мобільного додатку, Mini App:

- Не потребує встановлення

Доступ здійснюється безпосередньо з месенджера, що спрощує перший контакт користувача з демонстраційним додатком.

- Використовує безшовну авторизацію.

Завдяки *initData* [3], застосунок одразу ідентифікує користувача, демонструючи переваги інтегрованої автентифікації порівняно з традиційними формами реєстрації/логіну.

- Адаптується до інтерфейсу Telegram

Можливість використання параметрів теми (*themeParams*) та нативних елементів (showAlert і т.д.) [3] дозволяє створити додаток, що органічно виглядає та відчувається всередині месенджера, що є важливою характеристикою для демонстрації.

По-третє, TMA надає значно багатші можливості для створення інтерфейсу користувача, ніж класичний Telegram бот. У той час як бот обмежений текстовими повідомленнями та стандартними кнопками, Mini App дозволяє реалізувати повноцінний веб-інтерфейс з використанням HTML, CSS та JavaScript [3]. Це критично важливо для задачі пошуку квартир, яка вимагає зручного відображення списків, фотографій, детальної інформації та застосування комплексних фільтрів, що важко реалізувати лише засобами Bot API.

По-четверте, платформа демонструє ефективну синергію між веб-інтерфейсом (TMA) та можливостями бота (Bot API). Користувач може зручно налаштувати складні фільтри та підписки у візуальному інтерфейсі Mini App, а бот забезпечує надійне та своєчасне доставлення сповіщень про нові оголошення безпосередньо у чат. Це показує, як TMA та бот можуть доповнювати один одного.

Нарешті, розробка TMA базується на стандартних веб-технологіях, що робить її доступною для широкого кола веб-розробників та дозволяє використовувати сучасні frontend-фреймворки (як Vue.js у даному проєкті [11]), що також є важливим аспектом для демонстрації процесу розробки.

У порівнянні з альтернативами (окремий сайт, нативний додаток, чистий бот), Telegram Mini App пропонує унікальне поєднання гнучкості веб-розробки, глибокої інтеграції з популярним месенджером та зручного

користувацького досвіду, що робить цю технологію оптимальним вибором для досягнення мети даної курсової роботи – дослідження та демонстрації можливостей інтегрованих у Telegram веб-додатків.

РОЗДІЛ 3: Проєктування та розробка системи

3.1. Архітектура програмного комплексу

Для досягнення поставленої мети та реалізації необхідного функціоналу було розроблено програмний комплекс з модульною архітектурою. Такий підхід забезпечує гнучкість, масштабованість та полегшує розробку й підтримку окремих компонентів системи. Архітектура включає наступні основні модулі: Парсер даних, Базу даних, Серверну частину (Backend) та Клієнтську частину (Frontend - Mini App).

3.1.1. Парсер даних (Data Parser)

Оскільки застосунок потребує реальних даних для наповнення, то необхідний компонент, що буде їх збирати. Для цієї мети був розроблений окремий сервіс-парсер. Його основне завдання – періодично завантажувати HTML-сторінки зі списком та деталями оголошень оренди квартир з цільового сайту.

Було обрано сайт Rieltor.ua як джерело оголошень про оренду квартир. Загально можна було б розширити список джерел, наприклад, парсити оголошення квартир з “ОЛХ Нерухомість”, ЛУН, Dom.RIA, щоб була загальна агрегація оголошень з різних джерел у одній базі даних. Але для демонстраційного застосунку вистачить і одного джерела.\

У результаті парсер має вилучати з оголошень структуровану інформацію про квартири (ціна, адреса, характеристики, фото тощо). Спеціально для цієї цілі було розроблено окрему бібліотеку *rieltor_parser* [15] мовою програмування Rust. Вона використовує бібліотеку *reqwest* для

завантаження HTML-сторінок та бібліотеку *pest* для граматичного розбору HTML структури сторінок оголошень та списків оголошень. Парсер виокремлює ключову інформацію про квартиру (ціна, адреса, характеристики, фото тощо) та зберігає її у структурованому вигляді (*Apartment*).

У рамках проєкту парсер функціонує як окремий серверний процес (*rieltor_parser_server*), який періодично (з різними інтервалами для різних міст) завантажує списки нових оголошень, парсить деталі кожної нової квартири та зберігає їх у Базі Даних. Він також періодично перевіряє активність існуючих оголошень.

3.1.2. База даних

Для зберігання даних, зібраних парсером, а також інформації, специфічної для користувачів Mini App, використовується система управління базами даних. Вона слугує сховищем, з якого Серверна частина (Backend) отримує дані для відображення у Mini App та для перевірки підписок.

Обрана система керування базами даних – MongoDB [12], нереляційна документо-орієнтована БД, що добре підходить для зберігання гнучких за структурою даних, якими є оголошення про нерухомість.

База даних містить наступні колекції:

- Колекція *apartments*: Зберігає повну інформацію про кожне оголошення, отриману від парсера, включаючи історію змін ціни (*price_history*), історію оновлень (*update_history*) та статус активності (*is_active*).
- Колекції для користувацьких даних (*subscriptions, favourites*): Зберігають інформацію про підписки користувачів на пошукові

запити та список обраних ними оголошень, прив'язані до *user_id* з Telegram.

- Допоміжні колекції (наприклад, *subway_stations, districts, rieltors*): Зберігають унікальні сутності, виділені з оголошень, для можливої подальшої фільтрації. Наприклад, станції метро, райони, житлові комплекси, ріелторів та агенції нерухомості.

Парсер записує та оновлює дані в колекції *apartments* та допоміжних колекціях. Серверна частина (Backend) читає дані з *apartments* для відображення в Mini App та читає/записує дані в користувацькі колекції.

3.1.3. Серверна частина (Backend)

Backend є центральною ланкою, що забезпечує логіку роботи застосунку та взаємодію між компонентами. Саме тут проходить увесь процес взаємодії з БД, з Telegram Mini App, з користувачем. Вона реалізує API, через який Mini App може запитувати списки квартир, застосовувати фільтри, отримувати деталі окремого оголошення, зберігати обране та створювати підписки. Важливою функцією Backend є валідація автентичності запитів від Mini App за допомогою перевірки initData [3]. Також Backend відповідає за логіку роботи підписок: періодично перевіряє нові оголошення у Базі даних та, у разі знаходження відповідності збереженим підпискам користувачів, ініціює надсилання сповіщення через Telegram Бота.

3.1.4. Клієнтська частина (Frontend - Mini App)

Це основний компонент, що демонструє технологію Telegram Mini Apps. Являє собою веб-додаток (Single Page Application), розроблений за допомогою веб-технологій (HTML, CSS, JavaScript) та фреймворку Vue.js [11]. Він завантажується та виконується у вбудованому WebView клієнта

Telegram при натисканні користувачем відповідної кнопки або при переході на відповідне посилання, що ініціює запуск Mini App.

Frontend відповідає за візуальне представлення даних: відображення списків квартир, фільтрів, деталей оголошення. Він взаємодіє з користувачем, обробляє його дії (кліки, вибір фільтрів) та взаємодіє з Backend API (Надсилає HTTP-запити (GET, POST, DELETE) до API серверної частини для отримання списків оголошень, деталей, збереження підписок/обраного тощо), передаючи initData для автентифікації запитів на сервері. Ключовою особливістю є використання JavaScript API *window.Telegram.WebApp* [3] для інтеграції з Telegram: отримання *initData* для авторизації, адаптація інтерфейсу до теми Telegram (*themeParams*), використання нативних UI елементів, надсилання даних боту (*sendData*) або закриття вікна (*close*)

3.1.5 Telegram Bot

Telegram Бот слугує точкою входу для користувача та каналом для асинхронних сповіщень. Його основні функції:

- Обробка команд користувача (наприклад, /start).
- Надсилання повідомлення з кнопкою типу web_app, що запускає Frontend Mini App [1, 3].
- Надсилання користувачам сповіщень про нові квартири, що відповідають їхнім підпискам, та про оновлення цін на квартири з розділу “Обране”. Ці сповіщення ініціюються Серверною частиною (Backend). Бот реалізований за допомогою Telegram Bot API [1] та бібліотеки Telegraf.js [14].

Взаємодія компонентів відбувається наступним чином: Парсер наповнює Базу Даних. Користувач через Бота запускає Mini App. Mini App (Frontend) отримує *initData*, запитує дані у Backend API, відображає їх користувачеві. Користувач взаємодіє з Mini App, зберігаючи

фільтри/обране через Backend API у Базу Даних. Backend періодично перевіряє Базу Даних та ініціює надсилання сповіщень через Бота користувачам, чиї підписки спрацювали.

3.2 Вибір технологій

3.2.1 Технології парсингу

Для розробки Парсера даних було обрано мову програмування **Rust**. Цей вибір обґрунтований високою продуктивністю, надійністю та безпекою роботи з пам'яттю, що важливо для стабільної роботи сервісу, який постійно обробляє зовнішні дані.

Для безпосереднього аналізу HTML-структури сайту Rieltor.ua була використана бібліотека-парсер **Pest**. Pest є парсер-генератором, що працює на основі Parsing Expression Grammars (PEG), дозволяючи декларативно описувати складні граматики (в даному випадку, структуру HTML-сторінок) та генерувати ефективний код для їх розбору [15]. Такий підхід забезпечує більшу гнучкість та стійкість до змін у структурі цільового сайту порівняно з використанням регулярних виразів чи простих DOM-парсерів.

Сам парсер був оформлений у вигляді окремої бібліотеки **rieltor_parser**. Сервіс парсера, що виконує регулярні запити та взаємодіє з базою даних, побудований з використанням асинхронного рантайму Tokio, який дозволяє ефективно обробляти велику кількість одночасних операцій вводу-виводу (мережеві запити, робота з БД).

На основі бібліотеки **rieltor_parser** було створено окремий серверний застосунок (**rieltor_parser_server**), також написаний на Rust. Цей сервер виконує роль фонового сервісу, відповідального за оркестрацію процесу парсингу та взаємодію з базою даних. Він використовує асинхронний рантайм Tokio для ефективної обробки паралельних завдань. Сервер періодично, з налаштованими інтервалами для різних міст (напр., Київ

кожні 30 секунд, Львів кожні 45 секунд, Одеса кожні 60 секунд), запускає завдання парсингу списків нових оголошень з Rieltor.ua, використовуючи функціонал бібліотеки **rieltor_parser**.

Окрім отримання нових оголошень, сервер також реалізує логіку перевірки актуальності раніше збережених активних оголошень. Він періодично завантажує сторінки цих оголошень, перевіряє їх доступність та парсить їх знову для виявлення змін (наприклад, зміна ціни, опису, статусу). Сервер веде історію змін ціни та статусів для кожного оголошення (*price_history, update_history*) та позначає оголошення як неактивні, якщо сторінка стає недоступною або виникає помилка парсингу.

Важливою частиною сервера є механізм обробки потенційних блокувань з боку Cloudflare. Сервер аналізує відповіді від Rieltor.ua і, у разі виявлення ознак блокування, автоматично призупиняє свою роботу на певний час, щоб уникнути надмірного навантаження та можливого постійного блокування IP-адреси.

Для взаємодії з базою даних MongoDB сервер використовує офіційний Rust-драйвер **mongodb**. Він не лише зберігає нові та оновлює існуючі оголошення (*ApartmentEntity*), але й підтримує окремі колекції для унікальних сутностей (станції метро, райони, ріелтори, агентства), використовуючи внутрішній кеш (*EntityCache*) для оптимізації перевірок їх наявності перед додаванням нових записів. Також реалізовано завдання періодичного очищення бази даних від старих, давно деактивованих оголошень. Конфігурація сервера (адреса БД, інтервали) задається через змінні середовища.

3.2.2 База даних

Як система керування базами даних була обрана **MongoDB** [12]. Це NoSQL документо-орієнтована база даних, яка зберігає дані у форматі

BSON (бінарний JSON). Такий підхід добре пасує для зберігання оголошень про квартири, структура яких може дещо варіюватися. Гнучкість схеми MongoDB спрощує процес розробки та подальшої модифікації структури даних без необхідності складних міграцій. MongoDB також добре масштабується та має офіційні драйвери для багатьох мов програмування, включаючи Rust (використовується у Парсері) та Node.js (використовується у Backend).

3.2.3 Серверна частина

Для реалізації Backend API була обрана платформа Node.js [9] завдяки її асинхронній, неблокуючій моделі вводу-виводу, що добре підходить для обробки великої кількості одночасних запитів від клієнтів Mini App та взаємодії з базою даних. Велике співтовариство та екосистема модулів (npm) значно прискорюють розробку.

Як веб-фреймворк було використано Express.js [10] – популярний, мінімалістичний та гнучкий фреймворк, що надає зручні інструменти для маршрутизації запитів, обробки middleware та побудови RESTful API.

Для взаємодії з MongoDB з Node.js була обрана бібліотека **Mongoose** [13]. Вона надає рівень абстракції над нативним драйвером MongoDB, дозволяючи визначати схеми даних, валідувати їх, виконувати запити за допомогою зручного API та реалізовувати бізнес-логіку безпосередньо на рівні моделей даних.

Для обробки фонових завдань, таких як перевірка підписок та надсилання сповіщень користувачам, а також обробка додавання до обраного, використовується система черг **BullMQ**, яка працює поверх **Redis**. **BullMQ** [18] – це надійна та продуктивна бібліотека для Node.js, що дозволяє створювати черги завдань, обробляти їх у фоновому режимі (*workers*) та керувати їх станом. **Redis** [19] виступає як швидке сховище ключ-значення, що використовується **BullMQ** для зберігання метаданих

черг та стану завдань. Такий підхід дозволяє розвантажити основний процес API-сервера від тривалих операцій та забезпечити надійну обробку завдань навіть після перезапусків сервера.

Для взаємодії з зовнішніми API, зокрема для отримання актуальних курсів валют з сервісу **ExchangeRate-API**, використовується бібліотека **Axios** [20]. Вона надає простий та зручний інтерфейс для виконання HTTP-запитів. Актуальні курси потрібні для коректного відображення цін в різних валютах у Mini App.

Для реалізації функціоналу "Поділитися" оголошенням або контактом рієлтора, який генерує комбіноване зображення або картку контакту, було обрано **Firebase Admin SDK** [21] для взаємодії з **Firebase Storage**. Згенеровані зображення (колажі з фото квартир) завантажуються у **Firebase Storage** для отримання публічного URL, який потім використовується Telegram для відображення прев'ю при шерінгу. Це дозволяє обійти обмеження Telegram на пряме завантаження великих зображень у відповідь на inline запит.

Для створення комбінованих зображень (колажів) з фотографій квартир використовується бібліотека **Sharp** [22]. Це високоефективна бібліотека для Node.js, що дозволяє маніпулювати зображеннями: змінювати розмір, обрізати, накладати одне на одне, змінювати формат тощо. У проєкті Sharp використовується для створення колажу з перших чотирьох фотографій квартири, який потім завантажується у **Firebase Storage** при функції “Поділитись” чи просто передається у повідомлення для сповіщення про нові квартири.

3.2.4 Клієнтська частина

Frontend частина, що реалізує інтерфейс Mini App, розроблена з використанням JavaScript фреймворку **Vue.js** (версія 3) [11].

Чому **Vue.js**? Вибір **Vue.js** обґрунтований кількома факторами.

По-перше, він відомий своєю пологою кривою навчання та простотою інтеграції, що важливо для швидкої розробки прототипу.

По-друге, його компонентний підхід дозволяє легко структурувати інтерфейс на перевикористовувані блоки (наприклад, картка квартири, блок фільтрів), що покращує організацію коду.

По-третє, реактивна система Vue.js автоматично відстежує зміни даних і оновлює DOM, що значно спрощує розробку динамічних інтерфейсів, таких як списки оголошень, що оновлюються при застосуванні фільтрів.

У порівнянні з React, **Vue** часто вважається простішим для початківців завдяки чіткішому розділенню HTML (шаблони), CSS та JavaScript у однофайлових компонентах та більш прозорій системі реактивності. Порівняно з Angular, **Vue** є менш "жорстким" (less opinionated), надаючи більше гнучкості у виборі підходів та бібліотек для різних аспектів розробки, що добре підходить для проєктів середнього розміру та прототипування. Відмінна офіційна документація та активна спільнота також є вагомими перевагами.

Для навігації між різними екранами або "сторінками" всередині Mini App використовується **Vue Router** [23] – офіційна бібліотека маршрутизації для **Vue.js**. Вона дозволяє створювати Single Page Application (SPA), де переходи між розділами відбуваються без повного перезавантаження сторінки, що забезпечує швидкий та плавний користувацький досвід.

Для управління глобальним станом додатку (наприклад, дані авторизованого користувача, активні фільтри, список обраних квартир, обране місто) використовується **Pinia** [24] – офіційна та рекомендована бібліотека управління станом для **Vue 3**. Pinia надає простий та інтуїтивно зрозумілий спосіб визначення "сховищ" (stores), де зберігаються дані, до

яких можна отримати доступ та змінювати їх з будь-якого компонента застосунку. Це допомагає уникнути складнощів з передачею даних між компонентами через props та події ("prop drilling").

Для збірки проєкту, розробки з "гарячою" заміною модулів (Hot Module Replacement - HMR) та оптимізації фінальної збірки використовується інструмент **Vite** [25]. Vite забезпечує надзвичайно швидкий запуск сервера розробки та миттєве оновлення коду в браузері завдяки використанню нативних ES-модулів.

3.2.5 Взаємодія з ботом (Telegraf.js)

Для розробки логіки Telegram Бота та взаємодії з Telegram Bot API з середовища Node.js була використана бібліотека **Telegraf.js** [14].

Чому **Telegraf.js**? Хоча існують інші бібліотеки для роботи з Telegram Bot API на Node.js, такі як node-telegram-bot-api, **Telegraf.js** був обраний завдяки своїй потужній middleware-архітектурі. Вона дозволяє легко додавати, комбінувати та перевикористовувати логіку обробки оновлень (повідомлень, команд, колбеків). **Telegraf** надає зручний об'єкт *Context* (*ctx*), який передається через усі middleware і містить всю необхідну інформацію про поточне оновлення, методи для відповіді користувачеві та можливість зберігати стан між обробниками. Це робить код більш чистим та структурованим.

Також **Telegraf** має вбудовану підтримку сесій, інструменти для створення "сцен" (для покрокової взаємодії з користувачем, хоча у даному проєкті вони можуть не використовуватися активно), та пропонує лаконічний синтаксис для обробки різних типів оновлень (bot.command, bot.on('text'), bot.on('web_app_data') тощо).

Бібліотека активно підтримується, має хорошу документацію та велику спільноту, що полегшує пошук рішень та прикладів. У порівнянні з node-telegram-bot-api, яка працює більше на основі подій (EventEmitter),

middleware-підхід **Telegraf** часто вважається більш зручним для побудови складної логіки бота.

Telegraf.js використовується для:

- Обробки команди /start та надсилання клавіатури з кнопкою *web_app.*
- Обробки даних, що надходять з Mini App через *window.Telegram.WebApp.sendData()* (подія *web_app_data*).
- Надсилання сповіщень користувачам про нові квартири чи про зміну цін обраних квартир (викликається з фонових завдань BullMQ).

3.3 Розробка компонентів системи

3.3.1 Розробка парсера оголошень

Центральним елементом збору даних для системи є модуль парсингу, який складається з двох основних частин: бібліотеки **rieltor_parser** та серверного застосунку **rieltor_parser_server**.

Бібліотека rieltor_parser відповідає безпосередньо за розбір HTML-структури сторінок оголошень з сайту Rieltor.ua. Для аналізу HTML використовується бібліотека pest – генератор парсерів, що базується на граматиках виразів розбору (Parsing Expression Grammars, PEG). Це дозволяє декларативно описати очікувану структуру HTML-сторінки у файлі grammar.pest, а pest генерує ефективний код для її розбору.

```
/// Parses additional details about the apartment, such as house type, room planning, or general state from "Details" section.
details_description = { "<div class=\"offer-view-section-title\">Детані</div>" ~ "<div class=\"offer-view-section-text\">" ~ det_descr_text ~ "</div>" }
/// Extracts the whole text of details description.
det_descr_text      =  { (!"</div>" ~ (not_detail | detail))+ }
/// Skips the uninteresting information.
not_detail          = _{ (!"</div>" ~ !detail ~ ANY)+ }
/// A `detail` is defined by multiple components such as `house_type`, `room_planning`, `state`, `bargain`.
detail              = _{ house_type | room_planning | state | bargain }
/// Extracts the type of the apartment's house.
house_type          = _{ "Будинок -" ~ house_value ~ "," }
/// Extracts the room planning of the apartment.
room_planning       = _{ "Планування кімнат" ~ planning_value ~ "." }
/// Extracts the state of the apartment.
state               = _{ "Загальний стан квартири -" ~ state_value ~ "." }
/// Extracts the bargain possibility.
bargain             =  { "Торг доречний" }
```

Рисунок 4.1 - Зразок частини граматики Pest

Такий підхід є більш гнучким та стійким до змін у верстці цільового сайту порівняно з використанням, наприклад, лише регулярних виразів [15].

Бібліотека використовує reqwest для асинхронного завантаження HTML-сторінок за URL.

```
/// Fetches the HTML content of an apartment list page from the given URL.
///
/// # Arguments
/// * `url` - The URL of the apartment list page.
///
/// # Returns
/// A `Result` containing the HTML content as a `String`, or an error if the URL is invalid
/// or the request fails.
///
/// # Errors
/// - Returns an error if the URL does not match the expected `apartment_list_link` grammar rule.
/// - Returns an error if the HTTP request fails or if the response cannot be parsed.
1 usage  denisinside
pub async fn fetch_apartment_list_html_from_url(url: &str) -> Result<String> {
    ApartmentParser::parse(Rule::apartment_list_link, url)
        .map_err(|_| anyhow!("Incorrect apartment list link."))?;
    let response : Response  = reqwest::get(url).await?;
    let content : String  = response.text().await?;
    Ok(content)
}
```

Рисунок 4.2 - витягування HTML сторінки оголошення

Результатом роботи парсера є структурований об'єкт Apartment, що містить всю вилучену інформацію: ціну (Price), адресу (Address), характеристики (Characteristics, включаючи кількість кімнат, площу,

поверх), опис (Description), дозволи (Permits), дані про інфраструктуру (Infrastructure), рієлтора (Rieltor) та список фотографій.

```rust
/// The main structure representing an apartment.
10 usages  denisinside
#[derive(Clone, Serialize, Deserialize, Debug)]
pub struct Apartment {
    /// Unique identifier of the apartment.
    pub _id: String,
    /// Link to the apartment's webpage.
    pub link: String,
    /// The price of the apartment, including currency.
    pub price: Price,
    /// Address details of the apartment.
    pub address: Address,
    /// Various characteristics of the apartment, such as rooms, area, and floor.
    pub characteristics: Characteristics,
    /// Description of the apartment.
    pub description: Description,
    /// Permits or special tags associated with the apartment.
    pub permits: Permits,
    /// Information about nearby infrastructure.
    pub infrastructure: Infrastructure,
    /// Details about the realtor managing the apartment.
    pub rieltor: Rieltor,
    /// List of photo URLs for the apartment.
    pub photo: Vec<String>,
}
```

Рисунок 3.2 - Структура Apartment

Серверний застосунок rieltor_parser_server, також написаний на Rust з використанням асинхронного рантайму Tokio, виступає як фоновий сервіс, що керує процесом парсингу. Його основні завдання:

Періодичний запуск парсингу: Сервер з налаштованими інтервалами (різними для різних міст, наприклад, Київ – кожні 30 секунд, Львів – кожні 40 секунд) завантажує сторінки зі списками нових оголошень для кожного міста з Rieltor.ua, використовуючи бібліотеку rieltor_parser для вилучення посилань на окремі оголошення.

```
// Ініціалізація списку міст з URL
let mut cities = HashMap::new();
cities.insert("Київ".to_string(), ("https://rieltor.ua/flats-rent/?sort=bycreated".to_string(), 30));
cities.insert("Харків".to_string(), ("https://rieltor.ua/harkov/flats-rent/?sort=bycreated".to_string(), 90));
cities.insert("Львів".to_string(), ("https://rieltor.ua/lvov/flats-rent/?sort=bycreated".to_string(), 40));
cities.insert("Чернігів".to_string(), ("https://rieltor.ua/chernigov/flats-rent/?sort=bycreated".to_string(), 95));
cities.insert("Одеса".to_string(), ("https://rieltor.ua/odessa/flats-rent/?sort=bycreated".to_string(), 70));
cities.insert("Полтава".to_string(), ("https://rieltor.ua/poltava/flats-rent/?sort=bycreated".to_string(), 180));
cities.insert("Дніпро".to_string(), ("https://rieltor.ua/dnepr/flats-rent/?sort=bycreated".to_string(), 60));
cities.insert("Житомир".to_string(), ("https://rieltor.ua/zhitomir/flats-rent/?sort=bycreated".to_string(), 150));
```

Рисунок 3.3 - Дані для парсингу (Місто, посилання, інтервал)

Обробка нових оголошень: Для кожного нового, раніше не баченого оголошення, сервер завантажує його сторінку, парсить її за допомогою rieltor_parser та зберігає отриманий об'єкт Apartment у базу даних MongoDB як ApartmentEntity. При першому збереженні фіксується початкова ціна та створюється запис в історії оновлень (update_history) з типом "created".

```
#[derive(Clone, Debug, Serialize, Deserialize)]
struct ApartmentEntity {
    _id: String,
    apartment: Apartment,
    price_history: Vec<PriceChange>,
    update_history: Vec<UpdateHistory>,
    created_at: DateTime,
    updated_at: DateTime,
    is_active: bool,
}
```

Рисунок 3.4 - Структура ApartmentEntity

Перевірка актуальності існуючих оголошень: Сервер регулярно перевіряє активні оголошення, що вже є в базі. Він завантажує їх сторінки, парсить знову та порівнює отримані дані з наявними.

Відстеження змін: Якщо виявлено зміни (наприклад, ціна, опис, статус), сервер оновлює запис у базі даних, додає запис про зміну до price_history (якщо змінилась ціна) та до update_history з типом "updated" або "price_changed", фіксуючи час оновлення.

Деактивація оголошень: Якщо сторінка оголошення стає недоступною (помилка завантаження) або виникає помилка парсингу (що може свідчити про зняття оголошення або значну зміну верстки), сервер позначає оголошення як неактивне (is_active: false) та додає відповідний запис до update_history.

Керування унікальними сутностями: Для оптимізації та можливості подальшої фільтрації, сервер виділяє унікальні сутності з оголошень (станції метро, райони, житлові комплекси, рієлтори, агенції нерухомості) та зберігає їх в окремих колекціях MongoDB, використовуючи внутрішній кеш (EntityCache) для уникнення дублювання записів.

Очищення бази даних: Сервер періодично видаляє з бази даних записи про оголошення, які були деактивовані давно (наприклад, більше 3 днів тому), щоб підтримувати базу в актуальному стані.

Значною проблемою може бути блокування адреси через часті спроби отримання сторінок. У такому разі, Cloudflare блокує наші запити на перегляд сторінки. Для запобіганню цьому була запроваджена логіка інтервалів для запобіганню одночасних запитів. Такоє сервер аналізує відповіді від Rieltor.ua. У разі виявлення ознак блокування Cloudflare (за HTTP-статусом або ключовими словами в HTML), він автоматично призупиняє свою роботу на певний час (наприклад, 15 хвилин), щоб уникнути постійного блокування IP-адреси.

```rust
fn is_cloudflare_block_html(html: &str, status: StatusCode) -> bool {
    // Проста перевірка за ключовими словами в HTML
    let lower_html : String = html.to_lowercase();
    let keywords : [&str; 6] = [
        "cloudflare", "checking your browser", "access denied",
        "checking connection", "ray id", "ddos protection"
    ];

    if keywords.iter().any(|&word : &str | lower_html.contains(word)) {
        return true;
    }

    matches!(status, StatusCode::FORBIDDEN | StatusCode::SERVICE_UNAVAILABLE | StatusCode::TOO_MANY_REQUESTS)
}
```

Рисунок 3.5 - Метод, що перевіряє наявність блокування Cloudflare

Взаємодія з MongoDB здійснюється за допомогою офіційного Rust-драйвера mongodb. Конфігурація сервера (адреса БД, інтервали парсингу та перевірки) задається через змінні середовища. Таким чином, модуль парсингу забезпечує систему актуальними та структурованими даними про оголошення оренди квартир.

3.3.2 Розробка бази даних та моделей

Взаємодія з MongoDB у серверній частині (Node.js) здійснюється за допомогою бібліотеки Mongoose [13]. Mongoose надає зручний Object Data Modeling (ODM) інтерфейс, що дозволяє визначати схеми даних, валідувати їх та виконувати запити до бази даних у об'єктно-орієнтованому стилі. Це покращує структуру коду та полегшує роботу з даними.

Основні моделі даних, визначені за допомогою Mongoose:

Apartment: Ця модель відповідає колекції apartments, куди парсер **rieltor_parser_server** записує дані. Вона зберігає повну інформацію про кожне оголошення, отриману від парсера (ідентифікатор, посилання, ціна, адреса, характеристики, опис, фото тощо), а також додаткові метадані, що додаються сервером парсингу: історію змін ціни (*price_history*), історію оновлень оголошення (*update_history*), статус активності (*is_active*), та часові мітки створення (createdAt) і останнього оновлення (updatedAt).

UserSubscription: Зберігає налаштування підписок користувачів у колекції usersubscriptions. Кожен документ містить посилання на Telegram ID користувача (userId), набір критеріїв фільтрації (місто, ціновий діапазон, кількість кімнат тощо), та дату створення/оновлення.

UserFavourite: Зберігає список обраних оголошень для кожного користувача у колекції userfavourites. Документ містить Telegram ID користувача (userId) та масив ідентифікаторів (apartmentIds) обраних оголошень.

Також база даних містить таблиці для таких об’єктів, як станції метро, райони, рієлтори, агенції, визначні місця. Їх унікальним айді є об’єднанням міста та назви або інших полей. Вони також заповнюються під час парсингу сервером.

Ці моделі дозволяють ефективно зберігати та отримувати як загальні дані про квартири, так і персоналізовану інформацію для кожного користувача Telegram Mini App.

3.3.3 Розробка API серверної частини

Архітектура та структура

Архітектура Backend побудована за класичною трирівневою моделлю: Routes → Controllers → Services. Така структура сприяє чіткому розподілу відповідальності, полегшує тестування та подальшу підтримку коду:

1. **Routes (Маршрути)**

Визначають кінцеві точки API (endpoints) та HTTP-методи (GET, POST, PUT, DELETE), які приймає сервер. Вони відповідають за прийом запитів та передачу їх відповідним контролерам. Маршрути організовані в окремих файлах (apartmentRoutes.js, subscriptionRoutes.js, favouritesRoutes.js, currencyRoutes.js, botInteractionRoutes.js) для логічного групування ендпоінтів.

```
const router :Router = express.Router();

// GET /api/apartments - отримати всі оголошення з фільтрами
router.get( path: '/', getApartments);

// GET /api/apartments/cities - отримати список міст
router.get( path: '/cities', getCities);

// GET /api/apartments/districts - отримати райони за містом
router.get( path: '/districts', getDistricts);

// GET /api/apartments/subway-stations - отримати станції метро за містом
router.get( path: '/subway-stations', getSubwayStations);

// GET /api/apartments/residential-complexes - отримати ЖК за містом
router.get( path: '/residential-complexes', getResidentialComplexes);

// GET /api/apartments/landmarks - отримати landmarks за містом
router.get( path: '/landmarks', getLandmarks);
```

Рисунок 3.6 - Приклад визначення маршруту в routes/apartmentRoutes.js

2. **Controllers (Контролери)**

Виступають як проміжний шар між маршрутами та сервісами. Контролер отримує оброблений запит (request) та об'єкт відповіді (response) від Express, викликає необхідні методи сервісів для виконання бізнес-логіки, обробляє результати (або помилки) та формує HTTP-відповідь для клієнта. Вони знаходяться у директорії controllers.

```
export const getApartmentById = async (req, res) : Promise<...> => { Show usages
    try {
        const { id } = req.params;
        const apartment : ... | undefined = await apartmentService.getApartmentById(id);

        if (!apartment) {
            return res.status(404).json({
                success: false,
                message: 'Apartment not found'
            });
        }

        res.json({
            success: true,
            data: apartment
        });
    } catch (error) {
        console.error('Error in getApartmentById:', error);
        res.status(500).json({
            success: false,
            message: 'Failed to get apartment',
            error: error.message
        });
    }
};
```

Рисунок 3.7 - Приклад методу контролера в controllers/apartmentController.js

3. Services (Сервіси)

Містять основну бізнес-логіку застосунку. Сервіси взаємодіють з моделями даних Mongoose для роботи з базою даних (вибірка, створення, оновлення, видалення даних), виконують необхідні обчислення, валідацію бізнес-правил та підготовку даних для контролерів. Вони розташовані у директорії services.

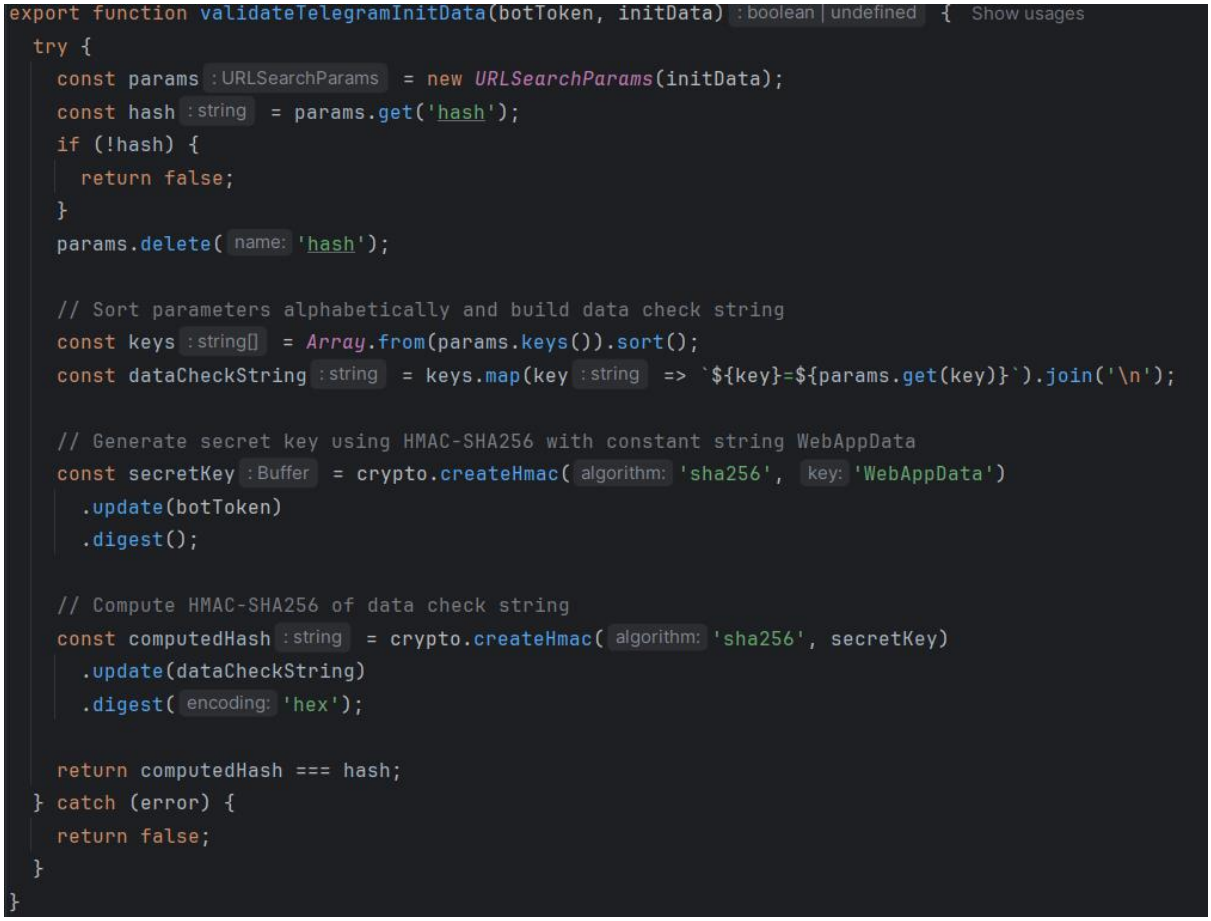
```
async getApartmentById(id) : Promise<... { Show usages
    try {
        const apartment : Query<...> & ObtainSchemaGeneric<module:mon... = await Apartment.findById(id);
        return apartment;
    } catch (error) {
        console.error("Error fetching apartment by id:", error);
        throw error;
    }
}
```

Рисунок 3.8 - Приклад методу сервісу в services/apartmentService.js

Валідація запитів з Mini App

Ключовим аспектом безпеки при взаємодії з Telegram Mini App є перевірка автентичності запитів, що надходять від клієнта. Оскільки Mini App запускається в середовищі Telegram, він отримує спеціальні дані ініціалізації (*initData*), підписані токеном бота. Ці дані містять інформацію про користувача та параметри запуску [3].

Для перевірки цих даних на Backend реалізовано спеціальний middleware **telegramValidatorMiddleware**. Цей проміжний обробник перехоплює всі запити до захищених ендпоінтів (наприклад, /api/apartments, /api/subscriptions, /api/favourites), вилучає initData (який клієнт Mini App передає, наприклад, у заголовку Authorization), та проводить криптографічну перевірку підпису за допомогою секретного ключа, отриманого з токена бота. Якщо перевірка успішна, middleware додає інформацію про користувача до об'єкта req (наприклад, req.user) і передає керування далі контролеру. Якщо перевірка не проходить, запит відхиляється з помилкою 401 Unauthorized. Це гарантує, що тільки запити, ініційовані з автентифікованого середовища Telegram Mini App, можуть отримати доступ до даних користувача.



```
export function validateTelegramInitData(botToken, initData) : boolean | undefined { Show usages
  try {
    const params : URLSearchParams = new URLSearchParams(initData);
    const hash : string = params.get('hash');
    if (!hash) {
      return false;
    }
    params.delete( name: 'hash');

    // Sort parameters alphabetically and build data check string
    const keys : string[] = Array.from(params.keys()).sort();
    const dataCheckString : string = keys.map(key : string => `${key}=${params.get(key)}`).join('\n');

    // Generate secret key using HMAC-SHA256 with constant string WebAppData
    const secretKey : Buffer = crypto.createHmac( algorithm: 'sha256', key: 'WebAppData')
      .update(botToken)
      .digest();

    // Compute HMAC-SHA256 of data check string
    const computedHash : string = crypto.createHmac( algorithm: 'sha256', secretKey)
      .update(dataCheckString)
      .digest( encoding: 'hex');

    return computedHash === hash;
  } catch (error) {
    return false;
  }
}
```

Рисунок 3.9 - Метод валідації initData користувача

```
[TelegramValidator] Incoming request to /api/favourites/467676002/apartments
[TelegramValidator] Extracted initData: query_id=AAFiK-AbAAAAAGIr4Bu110dm&user=%7B%22id%22%3A467676002%2C%22first_name%22%3A%22%D0%94%D0%B5%D0%BD%D0%B8%D1%81
%22%2C%22last_name%22%3A%22%22%2C%22username%22%3A%22Barson29%22%2C%22language_code%22%3A%22uk%22%2C%22is_premium%22%3Atrue%2C%22allows_write_to_pm%22%3Atrue
%2C%22photo_url%22%3A%22https%3A%5C%2F%5C%2Ft.me%5C%2Fi%5C%2Fuserpic%5C%2F320%5C%2F921Rv9Q5AI75_xVGPvUI05URCRocu9ZMUDRgVY8C7u0.svg%22%7D&auth_date=174646712
&signature=r0NKdoJDIcURv9zYZdqM7e9LBxLDalKNmuIZU8aA-AnPV9PCftGgnyApf1dXLIilNTmOAExE--h4nyo7-P_oBg&hash=dd25bd5f7f68349d7279b441e71a6ade205d229f19bde6a67ef045
5c97941bb3
[TelegramValidator] Raw initData: query_id=AAFiK-AbAAAAAGIr4Bu110dm&user=%7B%22id%22%3A467676002%2C%22first_name%22%3A%22%D0%94%D0%B5%D0%BD%D0%B8%D1%81%22%2C
%22last_name%22%3A%22%22%2C%22username%22%3A%22Barson29%22%2C%22language_code%22%3A%22uk%22%2C%22is_premium%22%3Atrue%2C%22allows_write_to_pm%22%3Atrue%2C%22
photo_url%22%3A%22https%3A%5C%2F%5C%2Ft.me%5C%2Fi%5C%2Fuserpic%5C%2F320%5C%2F921Rv9Q5AI75_xVGPvUI05URCRocu9ZMUDRgVY8C7u0.svg%22%7D&auth_date=1746467127&signa
ture=r0NKdoJDIcURv9zYZdqM7e9LBxLDalKNmuIZU8aA-AnPV9PCftGgnyApf1dXLIilNTmOAExE--h4nyo7-P_oBg&hash=dd25bd5f7f68349d7279b441e71a6ade205d229f19bde6a67ef0455c9794
1bb3
[TelegramValidator] Received hash: dd25bd5f7f68349d7279b441e71a6ade205d229f19bde6a67ef0455c97941bb3
[TelegramValidator] DataCheckString: auth_date=1746467127
query_id=AAFiK-AbAAAAAGIr4Bu110dm
signature=r0NKdoJDIcURv9zYZdqM7e9LBxLDalKNmuIZU8aA-AnPV9PCftGgnyApf1dXLIilNTmOAExE--h4nyo7-P_oBg
user={"id":467676002,"first_name":"Денис","last_name":"","username":"Barson29","language_code":"uk","is_premium":true,"allows_write_to_pm":true,"photo_url":"
https:\/\/t.me\/i\/userpic\/320\/921Rv9Q5AI75_xVGPvUI05URCRocu9ZMUDRgVY8C7u0.svg"}
[TelegramValidator] Computed hash: dd25bd5f7f68349d7279b441e71a6ade205d229f19bde6a67ef0455c97941bb3
[TelegramValidator] Hash match: true
[TelegramValidator] Request authorized
```

Рисунок 3.10 - Логи повного процесу валідації даних користувача

Основні API ендпоінти

/api/apartments: Отримання списку оголошень з можливістю гнучкої фільтрації за різними параметрами (місто, ціна, кімнати, поверх, наявність

метро, дозволи для тварин/дітей тощо), пагінації та сортування. Отримання детальної інформації про конкретне оголошення за його ID.

/api/subscriptions: Створення, отримання, оновлення та видалення підписок користувача на пошукові запити. Дозволяє користувачам зберігати свої фільтри та отримувати сповіщення про нові відповідні оголошення.

/api/favourites: Додавання та видалення оголошень зі списку обраного користувача. Отримання списку ID обраних оголошень.

/api/currency: Надання актуальних курсів валют (використовує currencyService для отримання та кешування даних з зовнішнього API) для коректного відображення цін у Mini App.

/api/bot: Спеціальні ендпоінти для взаємодії, що не обов'язково ініціюються з Mini App і не вимагають валідації initData. Наприклад, можуть використовуватись для обробки inline-запитів бота або інших внутрішніх потреб системи.

3.3.4 Розробка Telegram бота

Розробка бота починається зі створення його екземпляра через офіційний бот @BotFather у Telegram. Цей процес включає вибір імені та унікального username для бота, після чого @BotFather генерує унікальний токен доступу (Bot Token) [5]. Цей токен є ключем для взаємодії з Telegram Bot API [1] і має зберігатися конфіденційно.

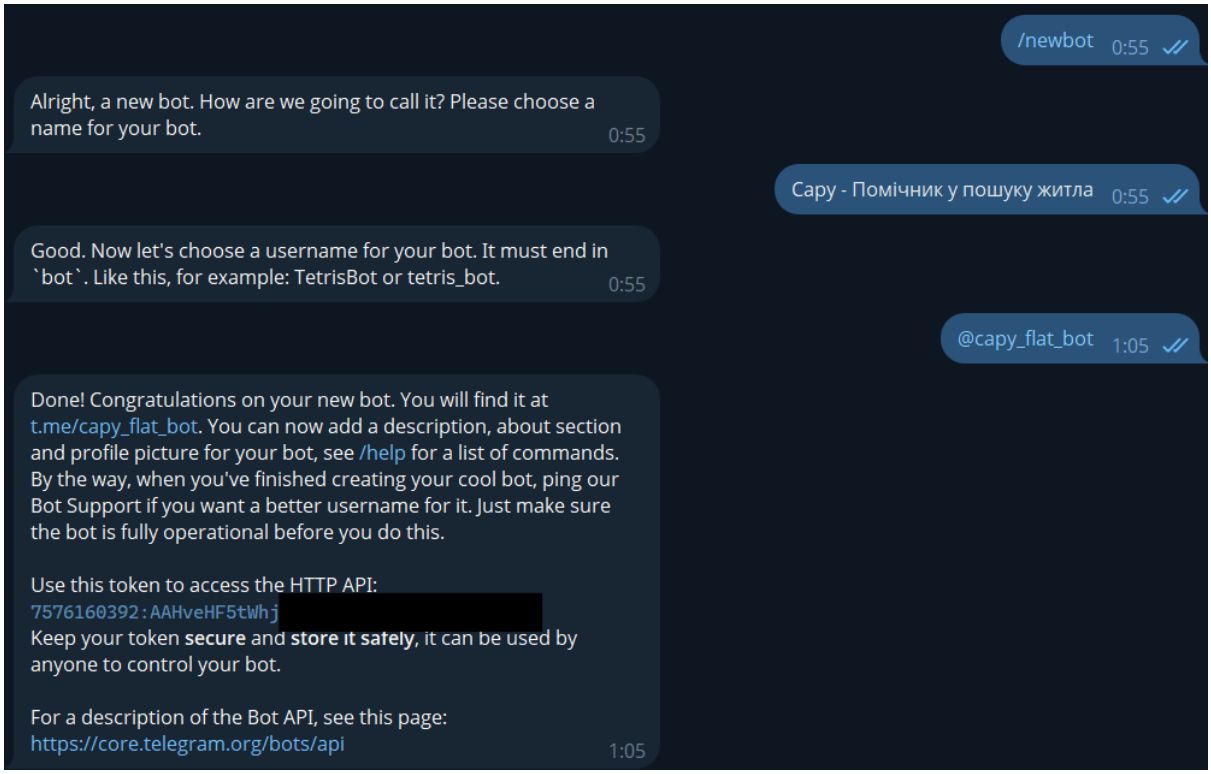


Рисунок 3.11 - Процес створення бота у @BotFather

Для взаємодії з Telegram Bot API з середовища Node.js використовуємо Telegraf.js [14]. Telegraf надає зручний об'єкт контексту (ctx), що передається через всі обробники і містить всю інформацію про оновлення та методи для відповіді користувачеві. Це спрощує написання чистого та підтримуваного коду порівняно з альтернативними бібліотеками.

Основні функції бота:

- **Обробка команди /start**

Це стандартна точка входу для багатьох ботів. При отриманні цієї команди бот надсилає привітальне повідомлення разом з Inline-клавіатурою. Ключовим елементом цієї клавіатури є кнопки типу web_app, які містять URL відповідних сторінок Mini App (/search, /favourites, /new, /subscription-settings). Натискання на таку кнопку відкриває Mini App безпосередньо у вікні Telegram [3].

```
// bot.js
bot.start( fns: (ctx) : void => {
    const chat_id : number = ctx.chat.id;
    console.log(`Received /start command from chat_id: ${chat_id}`);
    startMessage(ctx, chat_id);
});

// messages.js
export function startMessage(ctx, chat_id) {  Show usages
    ctx.telegram.sendPhoto(chat_id, Input.fromLocalFile( path: "data/images/hello.png"), extra: {
        caption: "Привіт! Я бот для пошуку оголошень про квартири у твоєму місті.\n\nОбери опцію:",
        reply_markup: {
            inline_keyboard: [
                [{ text: "🔍 Пошук", web_app: { url: `${WEB_APP_URL}/search` } }],
                [{ text: "⭐ Обране", web_app: { url: `${WEB_APP_URL}/favourites` } }],
                [{ text: "🔔 Нові", web_app: { url: `${WEB_APP_URL}/new` } }],
                [{ text: "⚙️ Налаштування підписок", web_app: { url: `${WEB_APP_URL}/subscription-settings` } }],
                [{ text: "📄 Шаблони договорів", callback_data: "share_contract" }],
            ],
        },
    });
}
```

Рисунок 3.12 - Приклад обробника /start в bot/bot.js та функції startMessage в bot/messages.js:

- **Запуск Mini App**

Як показано вище, бот використовує Inline-кнопки з полем web_app, що містить URL Mini App. Це основний спосіб запуску веб-додатку користувачем.

- **Обробка Callback Queries**

Бот реагує на натискання інших Inline-кнопок, що не є web_app. У даному проєкті це використовується для:

1. Надсилання шаблонів договорів оренди (callback_data: "share_contract"). Бот надсилає файли документів (PDF, DOCX) у відповідь на запит.
2. Надсилання контактної інформації рієлтора (callback_data: "get_rieltor_contact_"). Бот отримує ID квартири з callback_data, знаходить відповідне оголошення через apartmentService, витягує дані рієлтора та надсилає їх користувачеві за допомогою методу sendContact [1].

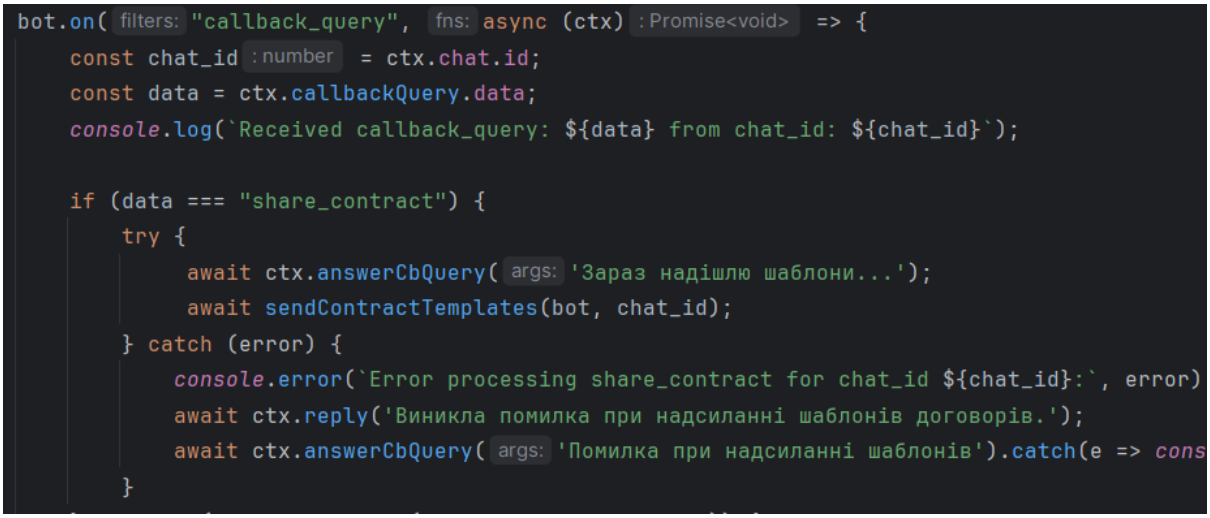


Рисунок 3.13 - Приклад обробки callback

- **Надсилання сповіщень**

Ця функція ініціюється серверною частиною (через систему черг BullMQ), але виконується ботом. Бот отримує ID користувача та список квартир (нових або оновлених обраних) і надсилає відповідні повідомлення:

1. Нові квартири: Для кожної нової квартири, що відповідає підписці користувача, бот надсилає фотоколаж (згенерований за допомогою combineApartmentImages) та форматований опис квартири з кнопками для переходу в Mini App або на сайт Rieltor.ua (sendApartmentsWithoutContext в messages.js).
2. Зміна ціни в обраному: Аналогічно надсилається повідомлення про зміну ціни для квартири зі списку обраного (sendFavouriteApartmentsUpdates в messages.js).

```
export async function sendApartmentsWithoutContext(bot, chat_id, apartments) : Promise<void> { Show usages
    try {
        await bot.telegram.sendMessage(chat_id, text: `🔔 Знайдено нові квартири за вашими критеріями:`);
        for (const apartment of apartments) {
            const replyMarkup : {inline_keyboard: ...} = {
                inline_keyboard: [
                    [
                        { text: "➡️ Детальніше в додатку", web_app: { url: `${WEB_APP_URL}apartment/${apartment._id}` } },
                        { text: "🆕 Всі нові", web_app: { url: `${WEB_APP_URL}/new` } }
                    ],
                     [{ text: "🌐 Відкрити на сайті", url: apartment.apartment.link }]
                ],
            };
            await sendSingleApartment(bot, chat_id, apartment, replyMarkup);
            await new Promise( executor: resolve => setTimeout(resolve, timeout: 300));
        }
    } catch (error) {
        console.error('Error sending new apartments:', error);
        await bot.telegram.sendMessage(chat_id, text: 'Виникла помилка при відправці інформації про нові квартири.');
    }
}
```

Рисунок 3.14 - Приклад надсилання сповіщень про нові квартири

- **Обробка web_app_data**

Бот може отримувати дані, надіслані з Mini App через window.Telegram.WebApp.sendData(). У поточному проєкті цей механізм використовується мінімально (лише логування), оскільки основна взаємодія Mini App з Backend відбувається через REST API.

- **Підготовка повідомлень для шерінгу (Inline Mode Interaction)**

Хоча бот не працює в повноцінному Inline Mode, він використовує схожий механізм для функції "Поділитися". Backend API (ендпоінти в routes/botInteractionRoutes.js) викликає метод bot.telegram.callApi('savePreparedInlineMessage'). Цей метод дозволяє підготувати форматоване повідомлення (з фото, текстом, кнопками) від імені бота, яке користувач потім може легко переслати в будь-який чат. Це використовується для шерінгу оголошень, контактів рієлторів та шаблонів договорів.

```
const response :... = await bot.telegram.callApi( method: 'savePreparedInlineMessage', payload:
    user_id: parseInt(userId, radix: 10),
    result: JSON.stringify(inlineQueryResult), // Об'єкт з даними повідомлення
    allow_user_chats: true,
    // ... інші параметри ...
});
// response.id повертається клієнту Mini App для ініціації шерінгу
```

Рисунок 3.15 - Приклад виклику для підготовки шерінгу квартири в routes/botInteractionRoutes.js:

Також Telegram потребує в Inline Messages пряме URL на зображення. Через те, що проєкт будується локально на комп'ютері, використовується тунель, щоб ззовні можна було доступитись д локальних портів, немає можливості вставити зображення квартир. Для цього було використано Firebase Storage. Спочатку створюємо зображення, завантажуємо його у сховище, потім звідти беремо URL та вставляємо у Inline Message.

```
if (apartment.photo && apartment.photo.length > 0) {
    if (storageBucket) {
        try {
            combinedImageBuffer = await combineApartmentImages(apartment.photo);
            storageFilePath = `share-images/apartments/${apartmentId}-${uuidv4()}.jpeg`;
            const file :File = storageBucket.file(storageFilePath);
            await file.save(combinedImageBuffer, options: {
                metadata: { contentType: 'image/jpeg', cacheControl: 'public, max-age=300' }
            });
            await file.makePublic();
            imageUrlForSharing = file.publicUrl();
            console.log(`Uploaded image to Storage. URL: ${imageUrlForSharing}`);
        } catch (uploadError) {
            console.error(`Failed to combine or upload image for apartment ${apartmentId}:`, uploadError);
            console.warn( message: 'Using first photo URL as fallback');
        }
    } else {
        console.log('Skipping image upload; using direct photo URL');
    }
}
```

Рисунок 3.16 - Використання FireBase Storage для отримання URL картинки

Таким чином, Telegram бот виступає як важливий елемент інтеграції, що забезпечує запуск Mini App, обробляє прості взаємодії та доставляє асинхронні сповіщення, роблячи досвід користувача більш зручним та інтерактивним.

3.3.5 Розробка інтерфейсу Telegram Mini App

Компонентний підхід Vue дозволяє розбити інтерфейс на логічні, перевикористовувані частини (наприклад, картка оголошення, фільтри, кнопки). Структура frontend-проєкту (frontend/) включає:

src/main.js: Точка входу застосунку, ініціалізація Vue, Pinia, Vue Router та обробка startParam від Telegram.

У залежності від стартового параметра буде визначено на якій сторінці відкриється застосунок. Так як View зберігає цей параметр, то при перезавантаженні сторінки, де б не був користувач, знову опиниться на сторінці вказаній у параметрі. Для цього він після першої переадресації видаляється, перевіряючи у Session Storage чи не був у користувача вже цей параметр.

src/router/index.js: Налаштування маршрутизації за допомогою Vue Router [23], що дозволяє навігацію між різними екранами (Views) без перезавантаження сторінки.

```
const router : Router = createRouter( options: {
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      redirect: '/profile',
    },
    {
      path: '/profile',
      name: 'profile',
      component: () : Promise<{...}> => import('../views/ProfileView.vue'),
    },
    {
      path: '/search',
      name: 'search',
      component: () : Promise<{...}> => import('../views/SearchView.vue'),
    },
    {
      path: '/apartment/:id',
      name: 'apartment-details',
      component: () : Promise<{...}> => import('../views/ApartmentDetailsView.vue')
    },
    // і так далі...
```

Рисунок 3.17 - Побудований Vue Router

src/stores/: Управління глобальним станом за допомогою Pinia [24]. Створено окремі сховища (city.js, favourites.js, subscriptions.js) для керування обраним містом, списком обраних квартир та підписками користувача.

src/views/: Компоненти, що представляють окремі сторінки/екрани застосунку (наприклад, SearchView.vue, ApartmentDetailsView.vue, ProfileView.vue).

src/components/: Менші, перевикористовувані UI-компоненти (наприклад, ApartmentList.vue, BackButton.vue).

src/api.js: Модуль для централізованого виконання запитів до Backend API. Важливо, що він автоматично додає заголовок X-Telegram-Init-Data з window.Telegram.WebApp.initData до всіх запитів для валідації на сервері.

src/useTelegram.js: Composable-функція Vue 3, що інкапсулює логіку взаємодії з window.Telegram.WebApp API, надаючи доступ до об'єкта tg, даних користувача (user), теми (theme), параметрів запуску (startParam) та інших функцій Telegram.

```
function initTelegram() : void  {  Show usages
  if (window.Telegram && window.Telegram.WebApp) {
    tg = window.Telegram.WebApp
    log( args: 'Telegram WebApp object:', window.Telegram.WebApp)
    log( args: 'Raw initData:', tg.initData)
    log( args: 'Parsed initDataUnsafe:', tg.initDataUnsafe)
    try {
      tg.ready()
    } catch (e) {
      log( args: 'Error calling tg.ready()', e)
    }
    if (tg.initDataUnsafe && tg.initDataUnsafe.user) {
      user.value = tg.initDataUnsafe.user
    } else {
      log( args: 'No user in initDataUnsafe:', tg.initDataUnsafe)
    }
    startParam.value = tg.initDataUnsafe?.start_param || null
    theme.value = tg.colorScheme || 'light'
    isReady.value = true

    checkHomeScreenStatus()

    if (tg.onEvent) {
      tg.onEvent('homeScreenAdded', () : void  => {
        homeScreenStatus.value = 'added';
        log( args: 'Event: homeScreenAdded');
      });
    }
  } else {
    log( args: 'Telegram WebApp not found. window.Telegram:', window.Telegram)
  }
}
```

Рисунок 3.18 - Отримання даних про користувача у src/useTelegram.js

Основні компоненти та екрани

SearchView.vue: Реалізує форму пошуку з усіма доступними фільтрами (місто, кімнати, ціна, площа, поверх, район, метро, ЖК, орієнтири, дозволи, комісія). Використовує компонент @vueform/slider для зручного вибору діапазонів та динамічно завантажує списки районів, станцій метро тощо з API.

ApartmentList.vue та ApartmentCard.vue: Відповідають за відображення списку знайдених оголошень. ApartmentCard.vue показує основну інформацію про квартиру у компактному вигляді.

ApartmentDetailsView.vue: Детальна сторінка оголошення. Відображає галерею фотографій (з можливістю перегляду у повноекранному модальному вікні), повну інформацію про квартиру, історію зміни ціни, блок з інформацією про рієлтора та кнопки дій ("Набрати", "Поділитися").

FavouritesView.vue: Показує список квартир, доданих користувачем до обраного (використовує favourites.js store).

SubscriptionSettings.vue: Дозволяє користувачеві переглядати, редагувати та видаляти свої підписки на пошукові запити (використовує subscriptions.js store).

ProfileView.vue: Відображає інформацію про користувача Telegram, отриману через useTelegram.js, та надає доступ до додаткових розділів ("Безпека", "Договори").

ContractsView.vue: Сторінка з шаблонами договорів оренди, що дозволяє переглянути PDF та завантажити/поділитися файлами.

Інтеграція з Telegram WebApp API

Взаємодія з Telegram здійснюється через composable useTelegram.js, який надає доступ до window.Telegram.WebApp:

Авторизація: При старті useTelegram.js отримує tg.initDataUnsafe для ідентифікації користувача (user.value). Ці дані далі передаються в api.js для валідації на бекенді.

Тема: useTelegram.js зчитує tg.colorScheme і tg.themeParams, а відповідні CSS-змінні (--tg-theme-bg-color тощо) використовуються у стилях компонентів для адаптації інтерфейсу до теми Telegram користувача.

Параметри запуску (startParam): main.js перевіряє наявність startParam при завантаженні. Якщо параметр присутній (наприклад, користувач перейшов за посиланням типу https://t.me/bot_name?startapp=<apartment_id>), роутер автоматично перенаправляє користувача на сторінку відповідного оголошення (ApartmentDetailsView.vue) або рієлтора (RieltorView.vue).

Cloud Storage: Сховище stores/city.js використовує tg.CloudStorage.setItem/getItem (з fallback на localStorage) для збереження обраного користувачем міста між сесіями.

Нативні дії: Використовуються tg.showAlert для показу простих сповіщень, tg.close() для закриття Mini App, tg.openLink для відкриття зовнішніх посилань (наприклад, посилання на оригінальне оголошення на Rieltor.ua).

Функція "Поділитися": Компонент ApartmentDetailsView.vue викликає відповідні функції з api.js (prepareApartmentShare, prepareRieltorShare), які звертаються до Backend API для підготовки повідомлення через bot.telegram.callApi('savePreparedInlineMessage'). Отриманий preparedMessageId передається у tg.shareMessage(), що відкриває стандартний діалог Telegram для вибору чату.

Функція "Набрати": Кнопка "Набрати" у ApartmentDetailsView.vue викликає функцію sendRieltorContactToUser з api.js, яка звертається до ендпоінту /api/bot/send-rieltor-contact-to-user. Бекенд, у свою чергу, надсилає контакт рієлтора користувачеві в особистий чат з ботом за допомогою методу bot.telegram.sendContact.

3.3.6 Розробка функціоналу

Реалізація основного функціоналу застосунку базується на тісній взаємодії між Frontend (Mini App), Backend API та фоновими процесами обробки даних.

Пошук та фільтрація

Механізм пошуку та фільтрації є центральною частиною взаємодії користувача з Mini App. Коли користувач встановлює критерії у формі пошуку (SearchView.vue) та натискає кнопку "Знайти квартири", frontend надсилає GET-запит до ендпоінту /api/apartments Backend API. Параметри фільтрації передаються як query-параметри URL.

На стороні Backend, контролер apartmentController.js (getApartments) приймає ці параметри, парсить їх (перетворюючи рядки JSON у об'єкти чи масиви за потреби) та передає у вигляді об'єкта subscriptionOptions до сервісу apartmentService.js (getApartments).

Сервіс apartmentService.js динамічно будує запит до MongoDB, використовуючи Mongoose. Логіка побудови запиту враховує різні типи фільтрів:

- Діапазони

Для ціни, площі та поверху використовуються оператори $gte (більше або дорівнює) та $lte (менше або дорівнює).

- Ціна

Особливістю є динамічна конвертація діапазону ціни. Якщо користувач обрав валюту, відмінну від базової (UAH), сервіс використовує актуальні курси валют (отримані з currencyService) для перерахунку мінімального та максимального значення в усі підтримувані валюти (UAH, USD, EUR) та будує $or умову, щоб знайти квартири, ціна яких в їхній оригінальній валюті потрапляє у відповідний конвертований діапазон.

```
let baseRates :{}  = {};
let lastFetch :number  = 0;
const FETCH_INTERVAL :number  = 1000 * 60 * 8;
const BASE_CURRENCY :string  = 'Uah';

async function fetchRates() :Promise<void>  {  Show usages
    try {
        const response :AxiosResponse<any>  = await axios.get( url: 'https://api.exchangerate.host/live', config: {
            params: {
                access_key: API_KEY,
                source: BASE_CURRENCY.toUpperCase(),
                currencies: 'USD,EUR'
            }
        });
        const quotes :string | {...}  = response.data.quotes || {};
        baseRates = {
            [BASE_CURRENCY]: 1,
            Usd: quotes[`${BASE_CURRENCY.toUpperCase()}USD`],
            Eur: quotes[`${BASE_CURRENCY.toUpperCase()}EUR`]
        };
        lastFetch = Date.now();
        console.log('Currency rates updated:', baseRates);
    } catch (error) {
        console.error('Failed to fetch currency rates:', error);
    }
}

fetchRates();
setInterval(fetchRates, FETCH_INTERVAL);
```

Рисунок 3.19 - Отримання актуального курсу за допомогою currencyService

- Множинний вибір

Для фільтрів за кількістю кімнат, районами, станціями метро, ЖК та орієнтирами використовується оператор $in, що знаходить документи, де відповідне поле містить будь-яке зі значень, обраних користувачем.

- Булеві значення

Для параметрів типу "дозволено з тваринами", "дозволено з дітьми", "без комісії" просто встановлюється відповідне значення true або false у запиті.

Сформований запит виконується до колекції apartments, і результат (список відповідних оголошень) повертається через контролер до Mini App для відображення.

Обране

Функціонал "Обране" дозволяє користувачам зберігати оголошення, що їх зацікавили. Взаємодія відбувається через API ендпоінти

/api/favourites, які обробляються favouritesController.js та favouritesService.js.

Дані зберігаються у колекції userfavourites, де кожен документ відповідає одному користувачеві (ідентифікованому за Telegram userId) і містить масив favourites з ID обраних ним оголошень (Apartment._id). Модель визначена у models/userFavourites.js.

Frontend надсилає POST або DELETE запити до API, передаючи userId та apartmentId. favouritesService оновлює відповідний документ у базі даних, додаючи або видаляючи ID квартири з масиву favourites.

Для сповіщення користувачів про зміни цін в обраних квартирах використовується фоновий процес, реалізований за допомогою BullMQ у файлі jobs/favouritesQueue.js. Цей процес:

1. Запускається періодично (наприклад, щохвилини).
2. Отримує всіх користувачів, що мають обрані квартири, з колекції userfavourites.
3. Для кожного користувача отримує повні дані його обраних квартир з колекції apartments.
4. Перевіряє поле update_history кожної квартири. Якщо знайдено запис з event_type: 'price_changed', дата якого (timestamp) пізніша за дату останньої перевірки для цього користувача (lastCheckedAt з userfavourites), ця квартира вважається оновленою.
5. Якщо знайдено оновлені квартири, викликається функція sendFavouriteApartmentsUpdates (bot/messages.js), яка надсилає користувачеві сповіщення через Telegram бота.
6. Оновлює поле lastCheckedAt для користувача у колекції userfavourites, щоб уникнути повторних сповіщень про ту ж саму зміну.

Підписки та сповіщення

Механізм підписок дозволяє користувачам зберігати налаштування фільтрів та отримувати автоматичні сповіщення про нові оголошення, що відповідають цим критеріям.

Користувач може створювати, переглядати, редагувати та видаляти підписки через інтерфейс Mini App (SubscriptionSettings.vue). Ці дії обробляються відповідними ендпоінтами API /api/subscriptions (subscriptionController.js та subscriptionService.js). Дані підписок зберігаються у колекції usersubscriptions, де кожен документ містить userId, об'єкт subscriptionOptions з параметрами фільтрації, дату останнього сповіщення (lastNotifiedAt) та масив ID вже надісланих оголошень (notifiedApartmentIds). Модель визначена у models/userSubscription.js.

Фоновий процес jobs/subscriptionQueue.js, також реалізований на BullMQ, відповідає за пошук нових оголошень:

1. Запускається періодично (наприклад, щохвилини).
2. Отримує всі активні підписки з колекції usersubscriptions.
3. Для кожної підписки викликає apartmentService.getApartments, передаючи збережені subscriptionOptions, щоб знайти всі актуальні оголошення, що відповідають критеріям.
4. Порівнює дату створення знайдених оголошень (created_at) з датою останнього сповіщення для даної підписки (lastNotifiedAt). Оголошення, створені пізніше lastNotifiedAt, вважаються новими для цієї підписки.
5. Якщо знайдено нові оголошення, викликається функція sendApartmentsWithoutContext (bot/messages.js), яка надсилає сповіщення користувачеві через Telegram бота.
6. Щоб уникнути повторних сповіщень, ID надісланих оголошень додаються до масиву notifiedApartmentIds у документі підписки, а поле lastNotifiedAt оновлюється поточним часом.

Перегляд сповіщень у Mini App: Розділ "Нові" (NewApartments.vue) у Mini App використовує ендпоінт /api/subscriptions/review/:userId, який викликає subscriptionService.getApartmentsForReview. Цей сервіс знаходить усі ID квартир з масивів notifiedApartmentIds для всіх підписок користувача, отримує повні дані цих квартир з колекції apartments та повертає їх для відображення. Коли користувач переглядає сповіщення і, наприклад, видаляє його зі списку, frontend викликає /api/subscriptions/review/:userId/:apartmentId (DELETE), що видаляє ID квартири з notifiedApartmentIds у відповідній підписці.

Таким чином, завдяки комбінації REST API для синхронних дій користувача та фонових черг для асинхронної обробки даних і сповіщень, система забезпечує реалізацію ключового функціоналу пошуку, збереження та відстеження оголошень.

Висновки

У даній курсовій роботі було досліджено можливості та процес розробки застосунків з використанням технології Telegram Mini Apps на прикладі створення системи для пошуку оренди квартир. Було розглянуто ключові аспекти платформи, включаючи Telegram Bot API та безпосередньо Web Apps API, проаналізовано існуючі підходи та спроєктовано архітектуру програмного комплексу.

Актуальність цієї роботи зумовлена зростаючою популярністю месенджера Telegram як платформи та технології Mini Apps, що відкриває нові можливості для створення інтегрованих та зручних сервісів безпосередньо всередині месенджера. Продемонстрований підхід показує переваги такої інтеграції: безшовна автентифікація, адаптивний інтерфейс, миттєві сповіщення та загалом кращий користувацький досвід порівняно з

традиційними веб-сайтами чи окремими мобільними додатками для вирішення подібних задач.

Система, побудована в ході виконання курсової роботи, демонструє практичне застосування широкого спектру можливостей Telegram Mini Apps та слугує наочним прикладом реалізації подібного проєкту. Вона може бути використана як основа для подальшого розвитку у повноцінний сервіс шляхом розширення функціоналу (наприклад, додавання нових джерел даних, розширення фільтрів, впровадження можливостей для рієлторів) або адаптації для вирішення інших прикладних задач, що вимагають інтеграції веб-додатків у середовище месенджера.

Використані джерела

1. Telegram Bot API Documentation. [Електронний ресурс]. URL: https://core.telegram.org/bots/api
2. Bots: An introduction for developers. [Електронний ресурс]. URL: https://core.telegram.org/bots
3. Telegram Mini Apps Documentation (Web Apps). [Електронний ресурс]. URL: https://core.telegram.org/bots/webapps
4. Telegram Bot API Changelog. [Електронний ресурс]. URL: https://core.telegram.org/bots/api-changelog
5. BotFather - офіційний бот для керування ботами Telegram. [Електронний ресурс]. URL: https://t.me/BotFather
6. DurgerKingBot - демонстраційний Telegram Mini App. [Електронний ресурс]. URL: https://t.me/DurgerKingBot
7. Flated Bot - Telegram-бот для пошуку та оренди квартир. [Електронний ресурс]. URL: https://www.flatedbot.com/
8. Findly Chatbot - Інформація про Telegram-бота для пошуку квартир без комісії. [Електронний ресурс]. URL: https://www.instagram.com/findly_chatbot (Дата звернення: ДД.ММ.РРРР - вставити актуальну дату)
9. Node.js Documentation. [Електронний ресурс]. URL: https://nodejs.org/
10. Express.js Documentation. [Електронний ресурс]. URL: https://expressjs.com/
11. Vue.js Documentation. [Електронний ресурс]. URL: https://vuejs.org/
12. MongoDB Documentation. [Електронний ресурс]. URL: https://www.mongodb.com/docs/
13. Mongoose Documentation. [Електронний ресурс]. URL: https://mongoosejs.com/docs/guide.html

14. Telegraf.js Documentation. [Електронний ресурс]. URL: https://telegraf.js.org/
15. Rieltor Parser Documentation - документація до парсера, використаного у проєкті. [Електронний ресурс]. URL: https://docs.rs/rieltor_parser/latest/rieltor_parser/
16. Telegram Bot API: Error Handling. [Електронний ресурс]. URL: https://core.telegram.org/bots/api#error-handling
17. Article about creating an online store in Telegram Mini App. [Електронний ресурс]. URL: https://bazucompany.com/blog/creating-an-online-store-with-telegram-mini-apps-a-comprehensive-guide/
18. BullMQ Documentation. [Електронний ресурс]. Режим доступу: https://docs.bullmq.io/
19. Redis Documentation. [Електронний ресурс]. Режим доступу: https://redis.io/docs/
20. Axios Documentation. [Електронний ресурс]. Режим доступу: https://axios-http.com/docs/intro
21. Firebase Admin SDK Documentation. [Електронний ресурс]. Режим доступу: https://firebase.google.com/docs/admin/setup
22. Sharp Documentation. [Електронний ресурс]. Режим доступу: https://sharp.pixelplumbing.com/
23. Vue Router Documentation. [Електронний ресурс]. Режим доступу: https://router.vuejs.org/
24. Pinia Documentation. [Електронний ресурс]. Режим доступу: https://pinia.vuejs.org/
25. Vite Documentation. [Електронний ресурс]. Режим доступу: https://vitejs.dev/