РЕФЕРАТ

Пояснювальна записка: 134 с., 41 рис., 3 дод., 25 джерел.

Об'єкт розробки: програмне забезпечення для подачі оголошень та зв’язку між сусідами на прикладі Telegram бота.

Мета кваліфікаційної роботи: розробити програмне забезпечення для подачі оголошень та зв’язку між сусідами на прикладі Telegram бота.

У вступі виконується аналіз сучасного стану проблеми, уточнюється постановка завдання, мета кваліфікаційної роботи та галузь її застосування, обґрунтовується актуальність теми.

У першому розділі проводиться дослідження предметної галузі та існуючих рішень, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання.

У другому розділі обирається платформа для розробки, виконується проектування програми і її розробка, наводиться опис алгоритму і структури функціонування системи, визначаються вхідні і вихідні дані, наводяться характеристики складу параметрів технічних засобів, описується робота програми.

В економічному розділі визначається трудомісткість розробленого програмного продукту, проводиться підрахунок вартості роботи по створенню застосунку та розраховується час на його створення.

Практичне значення полягає у розробці програмного забезпечення, за допомогою якого можна швидко подати оголошення або знайти створене у певному радіусі. Також для покращення UX було вирішено використовувати Telegram bot API, адже Telegram э одним з найпопулярніших додатків, тобто користувачам не буде потрібно завантажувати сторонній додаток та розбиратися в ньому.

Актуальність програмного продукту визначається великою популярністю засобів для подачі електронних об’яв.

Список ключових слів: БОТ, TELEGRAM, ПОДАЧА ОГОЛОШЕНЬ, API, РОБОТА З КООРДИНАТАМИ, POSTGIS, БЕЗСЕРВЕРНА АРХІТЕКТУРА.

ABSTRACT

Explanatory note: 134 p., 41 figs., 3 apps., 25 sources.

Object of development: software application for submitting ads and communication between neighbors based on the example of Telegram bot.

Purpose of the qualification work: development of a tool for fast assignment placing and communication between neighbors using the Telegram bot API.

In the introduction it is analyzed the current state of the problem, clarified the problem, the purpose of the qualification work and the scope of its application, substantiated the relevance of the topic.

In the first section the research of the subject area and existing decisions is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed.

In the second section the platform for development is chosen, the program design and its development are carried out, the description of algorithm and structure of functioning of system is given, input and output data are defined, characteristics of structure of parameters of technical means are given, work of the program is described.

In the economic section it is determined the complexity of the developed software product, calculated the cost of work to create an application and calculated the time to create it.

The practical value is to develop a tool with which you can quickly place an assignment or find created within a certain radius. It was also decided to use the Telegram bot API to improve UX, because Telegram is one of the most popular applications, i.e., users will not need to download a third-party application.

The relevance of the software product is determined by the great popularity of electronic advertising tools.

Keywords: BOT, TELEGRAM, PLACING ASSIGNMENTS, API, WORK WITH COORDINATES, POSTGIS, SERVERLESS ARCHITECTURE.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ООП – об’єктно-орієнтоване програмування;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп'ютер;

СКБД – система керування базою даних

ЗМІСТ

РЕФЕРАТ ... 3
ABSTRACT ... 4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ... 5
ВСТУП ... 8
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ ... 10
1.1. Загальні відомості з предметної галузі ... 10
1.2. Призначення розробки та галузь застосування ... 14
1.3. Підстава для розробки ... 15
1.4. Постановка завдання ... 16
1.5. Вимоги до програми або програмного виробу ... 17
1.5.1. Вимоги до функціональних характеристик ... 17
1.5.2. Вимоги до інформаційної безпеки ... 17
1.5.3. Вимоги до складу та параметрів технічних засобів ... 17
1.5.4. Вимоги до інформаційної та програмної сумісності ... 18
РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ ... 19
2.1. Функціональне призначення програми ... 19
2.2. Опис застосованих математичних методів ... 20
2.3. Опис використаної архітектури та шаблонів проектування ... 26
2.4. Опис використаних технологій та мов програмування ... 30
2.5. Опис структури програми та алгоритмів її функціонування ... 34
2.6. Обґрунтування та організація вхідних та вихідних даних програми ... 42
2.7. Опис розробленого програмного продукту ... 42
2.7.1. Використані технічні засоби ... 42
2.7.2. Використані програмні засоби ... 42
2.7.3. Виклик та завантаження програми ... 44
2.7.4. Опис інтерфейсу користувача ... 44
РОЗДІЛ 3 ... 58
ЕКОНОМІЧНИЙ РОЗДІЛ ... 58
ВИСНОВКИ ... 65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ... 66

ДОДАТОК А Код програми 68
ДОДАТОК Б Відгук керівника економічного розділу 133
ДОДАТОК В Перелік файлів на диску 134

ВСТУП

Написавши «подати об’яву» у рядок запиту можна побачити безліч ресурсів, які на це націлені. Зазвичай це сайти, де кожен бажаючий може подати своє оголошення, а всі відвідувачі сайту – прочитати його. Електронна дошка оголошень, як правило, поділена на кілька тематичних розділів, відповідно до змісту оголошень.

Більшість електронних дошок – безкоштовні. Для розміщення свого оголошення користувачеві потрібно лише ввести в спеціальній формі його тему, своє ім'я / псевдонім або назву організації, а також координати: адреса електронної пошти, поштова адреса, телефон, URL свого сайту і т. П. (Набір даних залежить від конкретного ресурсу). Як правило, відображаються тільки імена авторів і теми оголошень, а для перегляду повного тексту оголошення користувач повинен клацнути по посиланню, що веде до нього. У деяких дошках оголошення можуть подавати тільки зареєстровані користувачі, в деяких - усі. Зараз в інтернеті існує тисячі і навіть десятки тисяч дошок оголошень. Зазвичай кожна з них присвячується якомусь одному виду оголошень. Існують національні дошки оголошень, призначені для жителів конкретної місцевості.

Не дивлячись на різноманіття електронних дошок, багато з них мають свій UI (веб-версія або мобільний додаток) і для використання потрібно реєструватися в їх базі та знайомитися з новим UI.

Виходячи з вищеописаної інформації, було прийнято рішення розробити платформу для подачі оголошень на основі Telegram боту. Дане програмне забезпечення є актуальним через те, що воно, на відміну від більшості подібних, буде влаштоване у один з найпопулярніших месенджерів, отже для початку користування платформою юзерам не буде потрібно реєструватися у сторонній базі даних та знайомитися з новим UI.

Завдання даної кваліфікаційної роботи та об'єкт його діяльності безпосередньо пов'язані з освітньою програмою «Інженерія програмного

забезпечення» та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври спеціальності 121 «Інженерія програмного забезпечення» галузі знань 12 «Інформаційні технології». Виконання кваліфікаційної роботи надає можливості отримання автору кваліфікації «бакалавр з інженерії програмного забезпечення».

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Електронні дошки оголошень бувають двох видів: модеровані (ті, у яких є так званий модератор - людина, яка контролює роботу цієї дошки) і немодеровані - працюють автоматично.

Більшість дошок оголошень в інтернеті припускають розміщення оголошень про продаж і купівлю товарів і послуг. Але є і такі сервіси, які пропонують розміщувати оголошення про передачі різних речей в дар, а також про пошук нових господарів для домашніх тварин, які передаються в добрі руки безкоштовно. Безкоштовні оголошення - це найкращий спосіб заявити про себе, свої товари і послуги в мережі Інтернет. На сторінках дошок оголошень представлені тисячі безкоштовних оголошень про нерухомість, авто, послуги, пошук роботи, знайомства, купівлі та продажу обладнання, комп'ютерів, засобів зв'язку і до решти. Дошки оголошень дозволяють розмістити оголошення з фотографією і без додаткової реєстрації. Ви можете розмістити не тільки безкоштовне оголошення, але і оголошення на комерційній основі - для максимальної ефективності.

Існують спеціальні веб-сервіси, що працюють з дошками оголошень. Також існують і спеціалізовані дошки оголошень, адміністрацією яких вітається додавання строго тематичних оголошень по одній або кількох галузях промисловості, видах товарів чи послуг.

Проаналізуємо декілька найпопулярніших рішень.

Одна з найпопулярніших українських дошок оголошень – OLX (рис. 1.1).

Головний принцип, закладений в основу дизайну і структури, - мінімалізм. За допомогою рядка пошуку кожен користувач може здійснити

пошук по 13 мільйонам оголошень. Кожне з них перевіряється модераторами і видаляється в разі виявлення порушень.

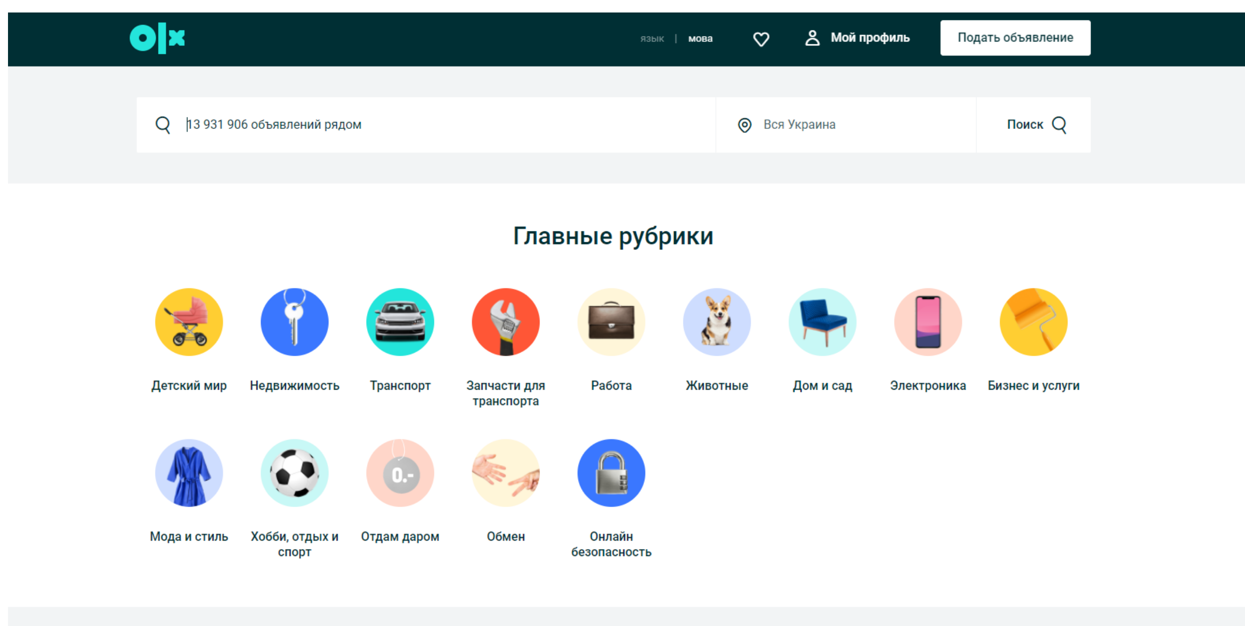

Рис. 1.1. Головна сторінка платформи OLX

Подача оголошень безкоштовна, але в обов'язковому порядку знадобиться зареєструватися і підтвердити номер мобільного за допомогою SMS. Останнім часом сервіс взяв курс на комерціалізацію, тому для успішної публікації в деяких рубриках доводиться купувати платні опції. Зокрема, це стосується будь-яких оголошень, пов'язаних з купівлею або продажем об'єктів нерухомості.

Під час створення оголошень пропонується вказати докладний заголовок, вибрати рубрику і додати опис (не більше 9000 знаків). Багатьом користувачам, як показує практика, вистачає всього декількох абзаців.

Максимальна кількість фото, які можна додати - не більше 8. Сервіс досить точно визначає розташування користувача, тому практично завжди пропонує релевантні оголошення.

Гіперпростір оголошень IZI.ua (рис. 1.2) є одним з проектів IT-компанії EVO. Завдяки добре продуманої стратегії і розширеним можливостям, ця

"молода" дошка оголошень вже встигла зайняти гідну позицію на українському ринку.

У своєму онлайн-просторі сервіс дає можливість:

- представникам бізнесу: створити привабливі оголошення та вибудувати безпечну комунікацію з клієнтами;
- споживачам: швидко знайти постачальника необхідної послуги або товару.

Крім веб-версії у IZI.ua є мобільні додатків як для Android, так і для iOS.

IZI.ua володіє наступними перевагами:

- інтуїтивно зрозумілий інтерфейс;
- безкоштовна доставка і повернення покупок в будь-яке відділення служби доставки Justin;
- якісний супровід та підтримка роботи сайту;
- відсутність обмежень для безкоштовних оголошень;
- наявність функції безпечної покупки;
- можливість автоматично перенести на IZI свої оголошення з інших онлайн-майданчиків.

Проте у порівнянні з іншими майданчиками IZI.ua залишається досить молодим брендом. Ресурс тільки почав завойовувати довіру продавців і покупців. Злагоджена робота команди IZI.ua дає відмінні результати. І хоч ця дошка поки не така популярна як, наприклад, OLX, в цьому теж є переваги: конкуренція в кожній категорії товарів нижче.

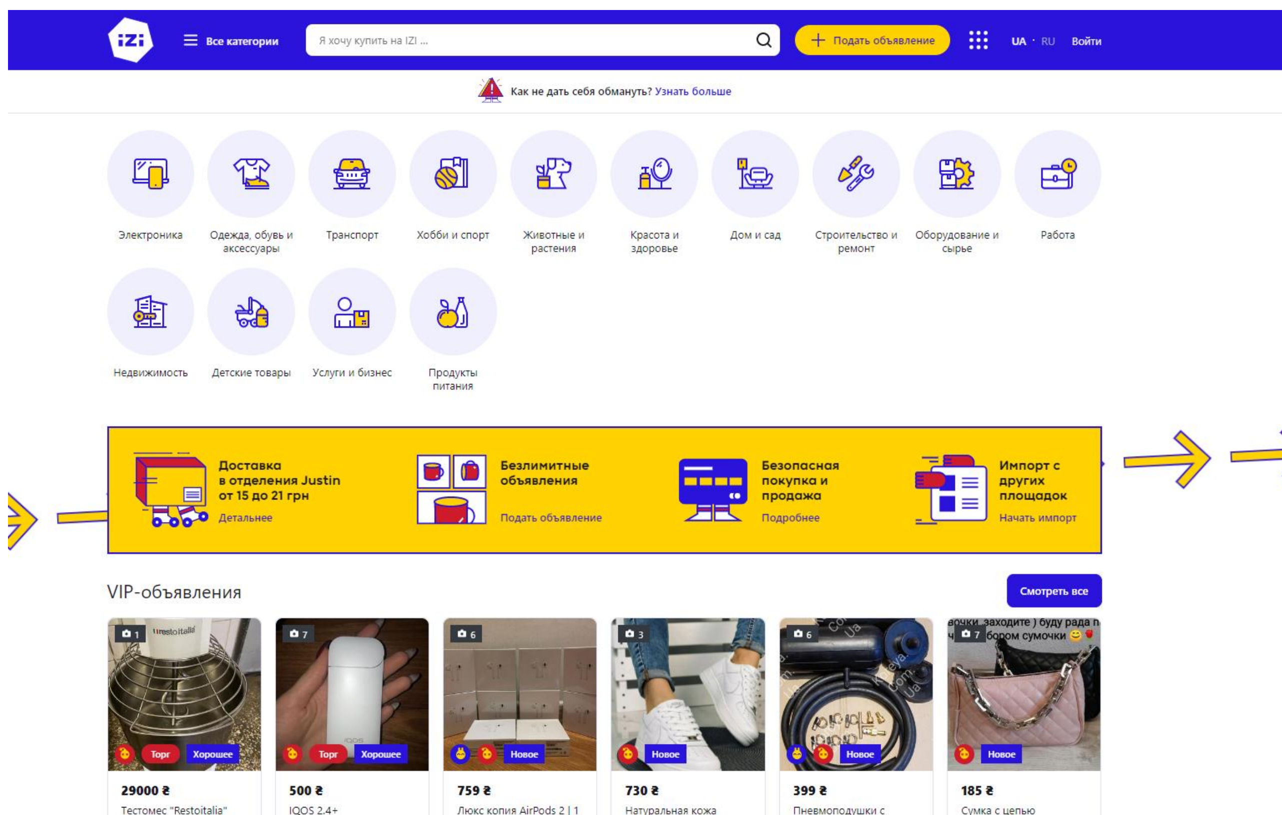

Рис. 1.2. Головна сторінка платформи IZI.ua

Бесплатка – це не просто дошка оголошень, а цілий бізнес-портал (рис. 1.3). Тут можна знайти пропозиції від продавців в самих різних сферах життя. Трошки ускладнює використання сервісу велика кількість реклами. Вона пропонує розпродаж залишків черевик і кросівок зі складу ECCO, стверджуючи, що "так дешево ще не було".

Мобільним користувачам пропонуються мобільні додатки. Вони доступні для Андроїд девайсів і айфонів. Якщо вам потрібно охопити велику кількість користувачів, скористайтеся можливістю придбання реклами. Сервіс пропонує банери, стверджуючи, що щомісячна відвідуваність порталу становить більше одного мільйона чоловік.

У нижній частині веб-сайту Бесплатка пропонує список популярних запитів, які користувачі вводять в пошукові системи перед тим, як щось продати або купити. Це стосується оренди з правом викупу, пневматичних пістолетів, човнових моторів, оптових поставок меду.

На бесплатці продається безліч товарів не тільки масового попиту, але і досить екзотичних. Серед них, зокрема, роги тварин, двигуни, гільзи для сигарет і багато інших.

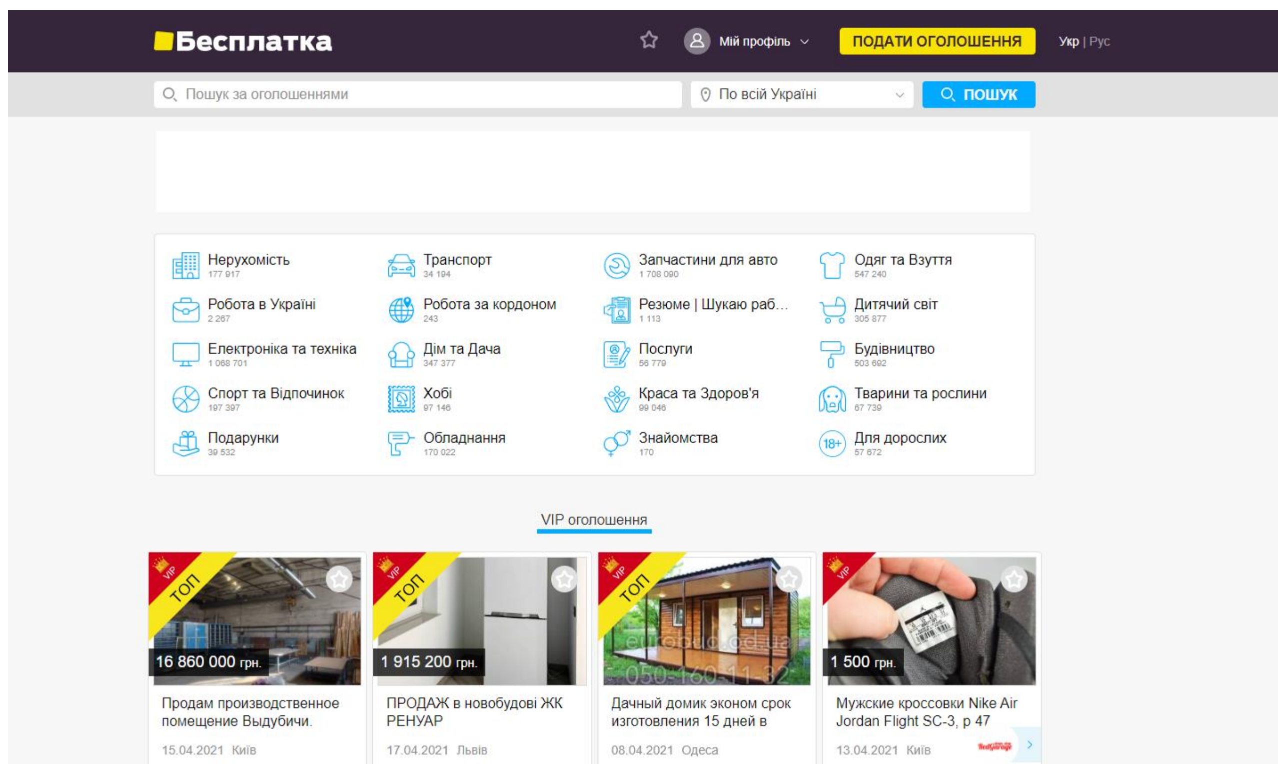

Рис. 1.3. Головна сторінка платформи BESPLATKA.UA

1.2. Призначення розробки та галузь застосування

Програмний продукт, що був виконаний для кваліфікаційної роботи, має назву «Розробка програмного забезпечення для подачі оголошень та зв'язку між сусідами на прикладі Telegram бота».

Основні терміни та ключові слова:

Telegram – багатоплатформовий месенджер з функціями, що дозволяє обмінюватися текстовими, голосовими та відеоповідомленнями, стікерами та фотографіями, файлами багатьох форматів.

Telegram bot – це просто Telegram акаунти, якими керує програмне забезпечення, а не люди, і вони часто мають функції ШІ. Вони можуть робити

що завгодно – навчати, грати, шукати, транслювати, нагадувати або інтегруватися з іншими службами.

Telegram bot API – це інтерфейс на основі HTTP, створений для розробників, які прагнуть створювати ботів для Telegram.

PostGIS – відкрите програмне забезпечення, що додає підтримку географічних об'єктів в реляційну базу даних PostgreSQL.

Serverless architecture – стратегія організації платформних хмарних послуг, при якій хмара автоматично і динамічно управляє виділенням обчислювальних ресурсів в залежності від призначеної для користувача навантаження.

Розроблений продукт має використовуватися у побуті, де люди зможуть викласти об'яву, а люди поруч відгукнутися на неї. Проте засіб не накладає обмежень, користувач сам зможе задати бажаний радіус пошуку об'яв.

Призначення розробки – надати можливість швидко подати об'яву, використовуючи свій улюблений месенджер.

1.3. Підстава для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- спеціальність 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 317-с від 07.06.2021 р;

– завдання на кваліфікаційну роботу на тему «Розробка програмного забезпечення для подачі оголошень та зв'язку між сусідами на прикладі Telegram бота».

1.4. Постановка завдання

Метою завдання є розробка програмного забезпечення для подачі оголошень та зв'язку між сусідами на прикладі Telegram бота. Особливістю розроблюваної дошки оголошень є її інтеграція в телеграм, адже перелічені популярні рішення є самостійними сайтами/додатками. Усі телеграм боти дуже схожі один на одного, тому користувач не буде завантажувати окремий додаток або, як мінімум знайомитись з новим інтерфейсом, бо вся взаємодія з дошкою оголошень буде відбуватися у його улюбленому месенджері. Основними характеристиками розроблюваної програми мають бути:

– можливість подати об'яву з бажаним описом;

– можливість встановити бажаний радіус пошуку об'яв інших користувачів;

– можливість пошуку об'яв у заданому радіусі;

– можливість з'єднатися з автором об'яви;

– можливість додавати об'яву в обране;

– можливість помітити об'яву як спам.

Поставлена мета може бути досягнена при виконані наступних вимог:

– вивчення предметної галузі розв'язуваного завдання (Telegram bot API);

– вибір релевантної архітектури програми;

– розробка структур вхідних і вихідних даних для проектованого програмного забезпечення;

– написання програмного коду застосунку.

Кінцева програма має представляти собою Telegram бота.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт має дотримуватися наступних функціональними вимог:

- інтуїтивно зрозумілий інтерфейс користувача;
- змога зберігати дані про об’яви;
- додавання, видалення і редагування інформації в базі даних системи.

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення надійного функціонування системи необхідно реалізувати наступні вимоги:

- валідація введених даних: перевірка на відповідність типів, на введення обов’язкових полів даних;
- обробка виняткових ситуацій;
- виведення повідомлень у разі виникнення помилок.

1.5.3. Вимоги до складу та параметрів технічних засобів

Розроблюваний продукт буде Telegram ботом, передбачити очікуване навантаження такого ботів досить складно. Саме цьому, щоб не розраховувати оптимальні вимоги до технічних засобів можна використати рішення, яке буде автоматично підлаштовуватися під потреби. Найпопулярнішою послугою для реалізації такого кейсу є AWS Lambda, яка автоматично привертає необхідні обчислювальні ресурси для обробки запиту.

1.5.4. Вимоги до інформаційної та програмної сумісності

Програмний продукт потребує від користувача мати середовище з такими складовими:

- одна з операційних систем: Windows, Linux, Mac OS, Android, IOS;
- мобільний додаток Telegram;
- постійний доступ до мережі Інтернет.

Бот буде розроблятися на мові JavaScript з використанням Node.js, хоча це не обов’язково, так як Telegram bot API підтримує багато інших мов (Python, Java, C#, PHP). Також необхідна база даних, наприклад PostgreSQL.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має бути стороння програма, яка працює всередині Telegram, адже боти це просто Telegram акаунти, які керуються програмним забезпеченням, а не людиною.

Розроблювана програма має дотримуватися наступних функціональних характеристик:

– ведення бази даних користувачів та їх даних;
– ведення бази даних оголошень;
– можливість створення оголошень;
– можливість пошуку оголошень;
– взаємодія з оголошенням (додати до вибраних, відмітити як «Спам»).

Кінцевий продукт буде володіти наступними експлуатаційними характеристиками:

– знайомий UX (не потрібно завантажувати сторонній додаток та знайомитися з його інтерфейсом, вся взаємодія з системою виконується через Telegram, яким користуються майже усі);
– швидкість подачі оголошення (немає набридливих функцій, оголошення можна подати менш ніж за хвилину);
– немодерована система (користувачі самі можуть поскаржитися на об’яву; при досягненні певної кількості скарг, вона буде автоматично видалена.), отже замовник не буде витрачати зайві кошти на модерацію продукту.

2.2. Опис застосованих математичних методів

У своїх налаштуваннях користувач може вибрати радіус пошуку об’яв. Для роботи з координатами буде використовуватися просторовий розширювач бази даних для PostgreSQL – PostGIS, про який розказується у наступних розділах. На даному етапі треба розуміти, що потрібен функціонал для визначення, чи знаходяться об’єкти в межах вказаної відстані.

Кажучи про географічні координати ми не можемо використовувати прямокутну систему координат (рис. 2.1). Географічні координати не представляють лінійної відстані від початку координат, як нанесено на площину. Це сферичні координати, які описують кутові координати на глобусі. У сферичних координатах точка задається кутом повороту від контрольного меридіана (довгота) і кутом від екватора (широта).

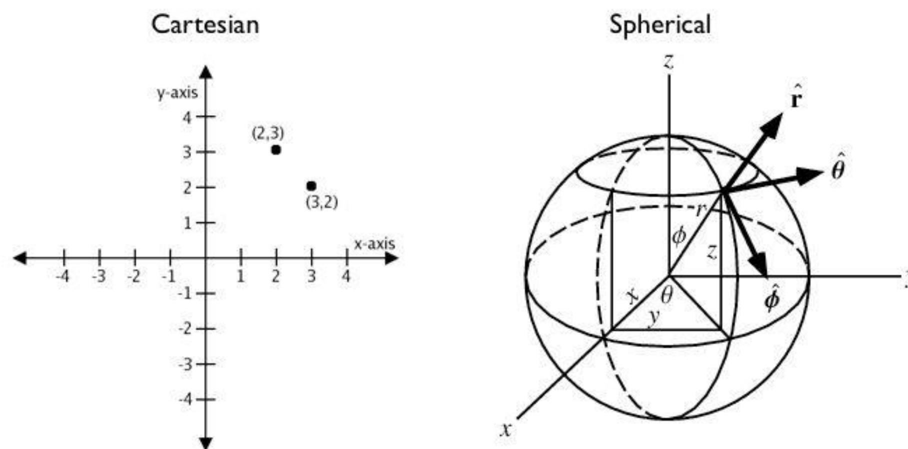

Рис. 2.1. Декартові та сферичні координати

Можна розглядати географічні координати як приблизні декартові координати і робити на їх основі просторові обчислення. Однак вимірювання відстані, довжини та площі буде неточним. Оскільки сферичні координати вимірюють кутову відстань, одиниці вимірювання рахуються в «градусах». Приблизні результати з використанням декартової системи можуть стати страшенно неправильними.

Рис. 2.2. Мапа з відстанню між Лос-Анджелесом та Парижем

Для прикладу, розрахуємо відстань між Лос-Анджелесом (33° 56′ 34.0476″ п. ш. 118° 24′ 36.1512″ з. д.) та Парижем (49° 0 35.0064 п. ш. 2° 32′ 52.0008″ з. д.) через Декартову систему координат, тобто відстань напряму, крізь Землю (рис. 2.2).

Технічно, широта і довгота представляють собою частину сферичних координат в тривимірному просторі. Радіус – це радіус Землі, кут нахилу - це широта, і азимут - це довгота. Якщо ми перетворимо сферичні координати в декартові координати x, y, z, ми зможемо легко знайти відстань між цими точками, використовуючи формулу Евклідової відстані (формула 2.1):

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.1}$$

Для перетворення сферичних координат до декартових використовуємо наступні формули 2.2:

$$\begin{gathered} x = R \cos\theta \cos\emptyset \\ y = R \cos\theta \sin\emptyset \\ z = R \sin\emptyset, \end{gathered} \tag{2.2}$$

де θ – широта, а $\emptyset$ – довгота.

Тепер пара тонких моментів, які виникають через той факту, що геодезія наближено представляє форму Землі у вигляді сплющенного сфероіда, або еліпсоїда обертання. Тому, коли ми говоримо про координати, ми насправді говоримо про координати на поверхні референц-еліпсоїда використовуваного в конкретній системі координат, в нашому випадку це WGS 84. І розрахована відстань, відповідно, буде відстанню між двома точками на поверхні референц-еліпсоїда.

Це призводить до необхідності врахування таких факторів:

– радіус Землі (радіус еліпсоїда) не є постійною величиною, і залежить від широти даної точки. Таким чином ми повинні розрахувати значення радіуса для кожної точки окремо.

– широта зазначена в WGS 84 - це геодезична широта, яка визначається кутом між площиною екватора і нормаллю до поверхні еліпсоїда в даній точці, на відміну від геоцентричної широти, яка визначається кутом між екваторіальній площиною і центром еліпсоїда. Так як в нашій декартовій системі координат початок координат знаходиться в центрі Землі, ми повинні перейти від геодезичних координат до геоцентричним для обох точок.

Враховуючи ці чинники, треба використати такі формули для перетворення широти і радіусу (формули 2.3):

$$\tan\alpha = \frac{b}{a}\tan\beta \qquad (2.3)$$

$$R = \sqrt{\frac{(a^2\cos\beta)^2 + (b^2\sin\beta)^2}{(a\cos\beta)^2 + (b\sin\beta)^2}},$$

де α – геоцентрічна широта, b – мала піввісь референц-еліпсоїда, a – велика піввісь референц-еліпсоїда, а β – геодезічна широта.

Отже, підбиваючи підсумок, можна сказати, що для того, щоб розрахувати відстань напряму між двома точками за заданими координатами, треба зробити наступне:

– розрахувати значення радіуса Землі в кожній точці в залежності від широти;

– розрахувати геоцентричну широту в кожній точці;

– перейти від сферичних координат до декартових використовуючи довготу, розрахований радіус і розраховану геоцентричну широту;

– розрахувати відстань використовуючи формулу Евклідової відстані;

Відстань між Лос-Анджелесом та Парижем крізь Землю, використовуючи евклідову відстань буде 8146.957 кілометрів.

Проте, як можна було здогадатися цей підхід не працює для обчислення реальної відстані між двома координатами на планеті, адже евклідова метрика призначена для обчислення відстані на площині, а поверхня Землі - це все-таки фігура, дуже близька до сфери. Для вирішення такого завдання потрібно звернутися до рідко використовуваних тригонометричним функціям.

Одна з таких функцій, називається синус-верзус (формула 2.4), або, по-іншому, версінус. Він являє собою відстань від центральної точки дуги, вимірюваної подвоєним даними кутом, до центральної точки хорди, що стягує дугу. Обчислюється версінус за формулою:

$$\text{versin}(\theta) = 2\sin^2\left(\frac{\theta}{2}\right) \quad (2.4)$$

Гаверсінус - це просто половина версінуса (формула 2.5), і саме ця функція допоможе нам у вирішенні завдання з пошуком відстані:

$$\text{hav}(\theta) = \frac{\text{versin}(\theta)}{2} = \sin^2\left(\frac{\theta}{2}\right) \quad (2.5)$$

Для будь-яких двох точок на сфері гаверсінус центрального кута між ними обчислюється за формулою 2.6:

$$\text{hav}\left(\frac{d}{r}\right) = \text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\text{hav}(\lambda_2 - \lambda_1), \quad (2.6)$$

де d – це центральний кут між двома точками, що лежать на великому колі,

r – радіус сфери, φ_1 та φ_2 – широти першої і другої точок в радіанах, λ_1та λ_2 – довготи першої і другої точок в радіанах.

Позначимо гаверсінус відношення довжини до радіусу як змінну h (формула 2.7):

$$\text{hav}\left(\frac{d}{r}\right) = h \quad (2.7)$$

Тоді довжину d можна винести за знак рівності (формула 2.8):

$$d = r\,\text{hav}^{-1}(h) \quad (2.8)$$

А для того, щоб позбутися від дробу, висловимо гаверсінус через арксинус (формула 2.9):

$$d = 2r \arcsin\sqrt{h} \quad (2.9)$$

Розкриємо змінну h ():

$$d = 2r \arcsin\sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\,\text{hav}(\lambda_2 - \lambda_1)} \quad (2.10)$$

Підставимо формулу гаверсінуса і отримаємо формулу 2.11 обчислення відстані:

$$d = 2r \arcsin \sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\ \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)} \qquad (2.11)$$

Перш, ніж підставляти координати в формулу, їх потрібно перевести в радіани та не забути, що Земля не є ідеальною сферою і її радіуси трохи варіюються (рис. 2.3). Скористаємося усередненим значенням радіуса, яке, відповідно до стандарту WGS84 приблизно дорівнює 6371 км.

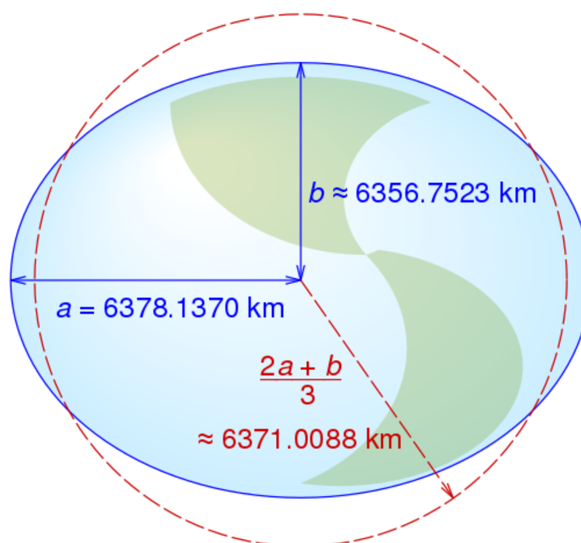

Рис. 2.3. Радіуси Землі

Результатом буде 8827.932 кілометри. Тобто похибка обчислення через евклідову відстань дорівнює майже 681 кілометрам. Це означає, що для обчислення відстані між віддаленими точками на планеті потрібно використовувати великі круги, коли ж працюємо в масштабах міста або району, можна знехтувати і використати евклідову відстань.

2.3. Опис використаної архітектури та шаблонів проектування

В якості архітектурного підходу для створення кінцевого програмного продукту була обрана безсерверна архітектура (рис. 2.4).

Розміщення програмного забезпечення в Інтернеті, як правило, передбачає управління якоюсь серверною інфраструктурою. Зазвичай це означає віртуальний або фізичний сервер, яким потрібно керувати, а також операційну систему та інші процеси хостингу веб-серверів, необхідні для запуску вашої програми. Використання віртуального сервера від хмарного провайдера, такого як Amazon або Microsoft, насправді означає усунення фізичних проблем, пов’язаних із апаратним забезпеченням, але все ж вимагає певного рівня управління операційною системою та процесами програмного забезпечення веб-сервера.

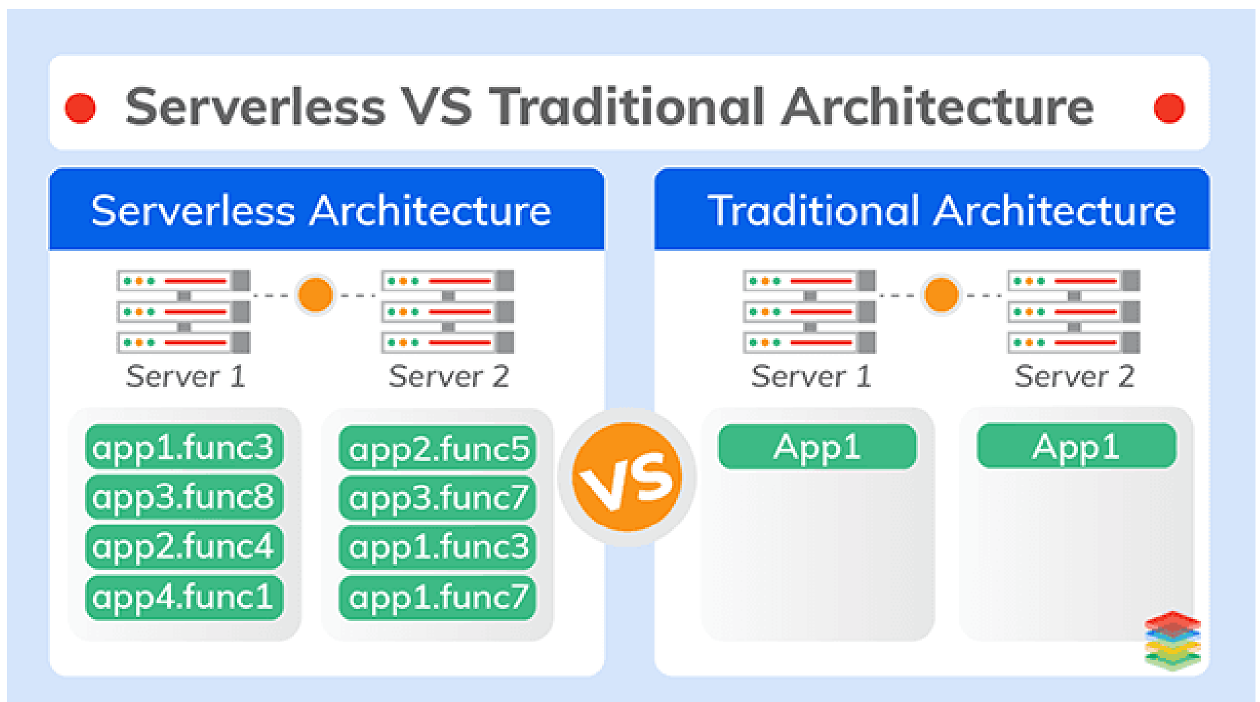

Рис. 2.4. Порівняння серверної та безсерверної архітектури

PaaS, або «платформа як послуга», такі продукти, як Heroku, Azure Web Apps та AWS Elastic Beanstalk, пропонують багато тих самих переваг, що і безсерверні (іноді їх називають «функція як служба» або FaaS). Вони усувають

необхідність управління серверним обладнанням та програмним забезпеченням. Основна відмінність полягає у способі складання та розгортання програми, а отже, у масштабованості програми.

За допомогою PaaS ваша програма розгортається як єдине ціле і розробляється традиційним способом із використанням певних веб-фреймворків, таких як ASP.NET, Flask, Ruby on Rails, Java-сервлети тощо. Масштабування здійснюється лише на рівні всього додатка. Ви можете вирішити запустити кілька екземплярів програми для обробки додаткового навантаження.

За допомогою FaaS ви компонуєте свою програму в окремі автономні функції. Кожна функція розміщується у постачальника послуг FaaS і може масштабуватися автоматично, коли частота виклику функції збільшується або зменшується. Це виявляється дуже економічно вигідним способом оплати обчислювальних ресурсів. Ви платите лише за час, коли викликаються ваші функції, а не платите за те, щоб ваша програма завжди працювала і чекала запитів на різних екземплярах.

Особливо слід розглянути можливість використання безсерверної архітектури, якщо у вас невелика кількість функцій. Якщо ваш додаток є більш складним, безсерверна архітектура все одно може бути корисною, але вам доведеться зосередити свою програму зовсім інакше. Це може бути неможливо, якщо у вас вже є існуюча програма. З часом може мати сенс перенести невеликі фрагменти програми на безсерверні функції.

Так як розроблюваний додаток є Telegram-ботом, передбачити навантаження та необхідні ресурси досить складно. Отже, замість того, щоб платити за роботу сервера 24 години на добу, буде використано безсерверне рішення, яке суттєво зменшить ціну обслуговування. Клієнт буде платити за кількість викликів сервера, а не за час його роботи. Конкретно у цьому випадку, буде використано AWS Lambda. Пропозиція по-попередньому буде працювати на серверах, а не керувати цим серверами AWS повністю бере на себе.

Однак однієї AWS Lambda недостатньо для функціонування бота. Обробка інформації, валідація, обчислення, взаємодія з базою даних буде відбуватися на стороні AWS Lambda, проте необхідний шлюз, який буде повідомляти лямбду про запроси. Для цього скористаємося Amazon API Gateway (рис.2.5).

Amazon API Gateway - це повністю керований сервіс для розробників, призначений для створення, публікації, обслуговування, моніторингу та забезпечення безпеки API в будь-яких масштабах. Через API додатка отримують доступ до даних, бізнес-логіки або функціональними можливостями ваших серверних сервісів. API Gateway дозволяє створювати API RESTful і WebSocket, які є головним компонентом додатків для двостороннього зв'язку в режимі реального часу. API Gateway підтримує робочі навантаження в контейнерах і бессерверной робочі навантаження, а також інтернет-додатки.

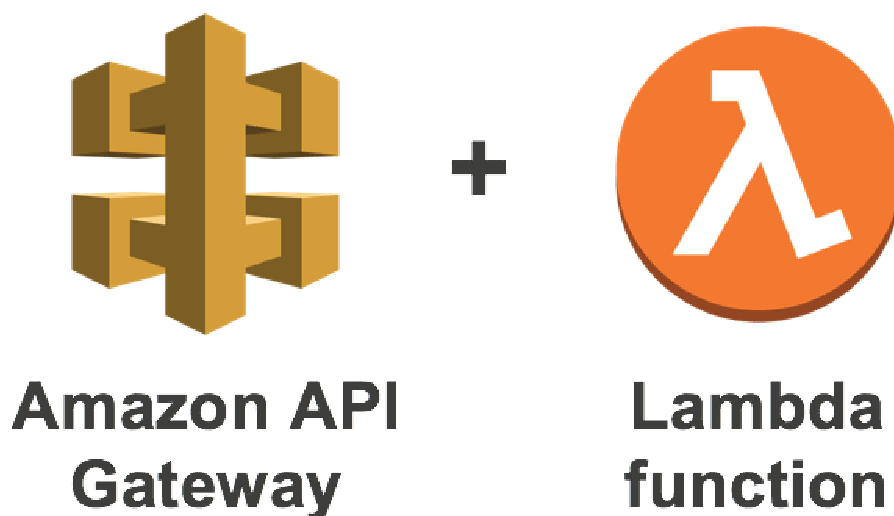

Рис. 2.5. Логотипи Amazon API Gateway та AWS Lambda

В моєму випадку будет використаний HTTP-протокол. Тепер коли ми маємо API та лямбду, необхідно відправляти запроси з Telegram на API. Зробити це можна за допомогою webhook.

Webhook – механізм оповіщення системи про події. Припустимо, є певний сервіс який повинен сповіщати про події, коли вони відбуваються. Як варіант, можна постійно запитувати нові дані на сервері, проте мінусом буде швидкість отримання даних, якщо на сервері встановлена затримка на отримання нових повідомлень, та кількість запитів. А ось webhook виправляє цей недолік: коли відбувається подія він запитує адресу сайту з параметрами, наприклад example.com/webhook/notification і передає в тілі POST запиту JSON (найчастіше) і там будуть свіжі дані.

Таким чином, схема передачі даних починаючи з користувацького введення, закінчуючи запитами до бази даних виглядає, як на рис. 2.6.

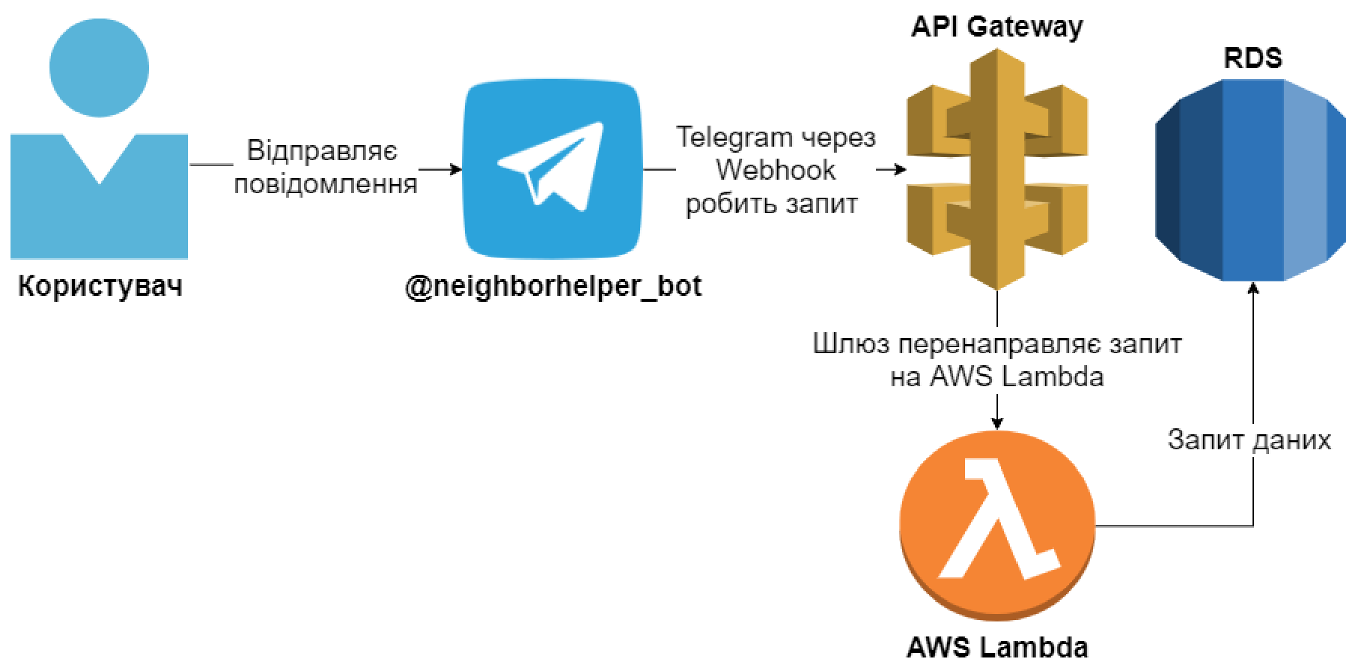

Рис. 2.6. Схема передачі даних починаючи з користувацького введення, закінчуючи запитами до бази даних

Також можна відмітити, що після отримання повідомлення від користувача, Telegram зберігає Webhook запит у черзі приблизно 4 години,

після чого повідомлення видаляється. Тобто якщо на серверній частині виникла несправність, то є 4 години для того, щоб отримати запит користувача.

2.4. Опис використаних технологій та мов програмування

Даний програмний продукт була розроблена з використанням таких технологій:

- Telegram bot API;
- JavaScript;
- Node.js;
- Claudia-bot-builder;
- PostgreSQL;
- PostGIS;
- Sequelize;
- AWS Lambda, Amazon API Gateway, Amazon RDS.

Telegram bot API – це інтерфейс на основі HTTP, створений для розробників, які прагнуть писати ботів для Telegram. По суті, Telegram-боти – це спеціальні облікові записи, для налаштування яких не потрібен додатковий номер телефону.

Користувачі можуть взаємодіяти з ботами двома способами:

- надіславши їм повідомлення та команди, відкриваючи чат або додаючи їх до груп;
- надіславши запити безпосередньо з поля введення, ввівши @username бота та запит. Це дозволяє надсилати вміст від вбудованих ботів безпосередньо в будь-який чат, групу або канал.

Повідомлення, команди та запити, надіслані користувачами, передаються програмному забезпеченню, що працює на серверах розробників. Проте перед цим відправлене користувачем повідомлення обробляється

посередницьким Telegram сервером, який виконує усі функції шифрування та зв'язку з API Telegram. Спілкування між цим та серверами розробників відбувається через простий HTTPS-інтерфейс.

Для того щоб почати розроблювати бота, його потрібно створити через іншого бота, який називається BotFather (рис. 2.7). Шукаємо бота з ім'ям @BotFather, вводимо бажане для нашого бота ім'я та натискаємо «Створити». Після цього батько ботів відправить нам унікальний токен, який ми будемо використовувати у розробці. Саме завдяки цьому токену посередницький Telegram сервер буде спілкуватися з нашим.

Рис. 2.7. Логотип батька ботів

JavaScript – прототипно-орієнтована мова програмування, яка спочатку була створена, щоб «зробити веб-сторінки живими».

Програми на цій мові називаються скриптами. Вони можуть вбудовуватися в HTML і виконуватися автоматично при завантаженні веб-сторінки.

Скрипти поширюються і виконуються, як простий текст. Їм не потрібна спеціальна підготовка або компіляція для запуску.

Сьогодні JavaScript може виконуватися не тільки в браузері, а й на сервері або на будь-якому іншому пристрої, який має спеціальну програму, що називається «движком» JavaScript.

У браузера є власний движок, який іноді називають «віртуальна машина JavaScript».

Різні движки мають різні «кодові імена», наприклад:

- V8 – в Chrome и Opera;
- SpiderMonkey – в Firefox;
- Також є «Trident» и «Chakra» для різних версій IE, «ChakraCore» для Microsoft Edge, «Nitro» і «SquirrelFish» для Safari і т.д.

Node.js представляє середу виконання коду на JavaScript, яка побудована на основі движка JavaScript Chrome V8, який дозволяє транслювати виклики на мові JavaScript в машинний код. Node.js перш за все призначений для створення серверних додатків на мові JavaScript. Хоча також існують проекти по написанню десктопних додатків (Electron) і навіть зі створення коду для мікроконтролерів. Але перш за все ми говоримо про Node.js, як про платформу для створення веб-додатків.

Claudia Bot Builder - це бібліотека, яка допомагає створювати ботів для Facebook Messenger, Telegram, Skype, Slack, Twilio, Kik та GroupMe. Ключова ідея проекту полягає у видаленні всіх шаблонних кодів та загальних завдань інфраструктури, щоб розробники могли зосередитися на написанні дійсно важливої частини бота - робочих процесів бізнесу.

Під час використання Bot Builder Клавдія автоматично встановить правильні веб-хуки для всіх підтримуваних за лічені хвилини.

Замість того, щоб вивчати окремі протоколи ботів і налаштовувати веб-хуки вручну, можна просто написати код для обробки запитів, а Claudia Bot Builder подбає про все інше.

Claudia Bot Builder спрощує робочі процеси обміну повідомленнями та перетворює вхідні повідомлення з усіх підтримуваних платформ у загальний формат, завдяки чому розробник може легко з ним працювати. Він також

автоматично упаковує текстові відповіді у потрібний формат для механізму ботів, що запитує, тому нам не доведеться турбуватися про форматування результатів для простих відповідей.

PostgreSQL - це потужна об'єктно-реляційна система баз даних з відкритим кодом, що займається більш ніж 30-річним активним розвитком, що забезпечило їй міцну репутацію надійності та продуктивності.

PostGIS - це просторовий розширювач бази даних для об’єктно-реляційної бази даних PostgreSQL. Він додає підтримку географічних об'єктів, що дозволяє виконувати запити про розташування в SQL. Також PostGIS пропонує багато можливостей, які рідко можна знайти в інших конкуруючих просторових базах даних, таких як Oracle Locator/Spatial та SQL Server. В моєму випадку PostGIS буде використовуватися для пошуку найближчих об’яв у заданому радіусі. Тобто замість того, щоб самостійно прописувати логіку, яка описана у розділі 2.2, можна взяти готове рішення.

Sequelize - це Node.js ORM (Object-Relational Mapping) на основі обіцянок для Postgres, MySQL, MariaDB, SQLite та Microsoft SQL Server. Він має надійну підтримку транзакцій, стосунки, прагнення та ліниве завантаження, реплікацію читання тощо. ORM – технологія програмування, яка зв'язує базу даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». Тобто замість написання SQL-запитів, взаємодія з базою буде відбуватися за допомогою класів, представляючих з себе таблиці в базі даних. Ці класи зазвичай називаються сутностями або моделями.

Усі перелічені технології э AWS (Amazon Web Service). Amazon Web Services (AWS) - це сама розповсюджена в світі хмарна платформа з широкими можливостями, що забезпечує більш ніж 200 повнофункціональних сервісів для центрів обробки даних на всій планеті. Мильони клієнтів, у тому числі стартапи, найбільші корпорації та передові державні установи, використовують AWS для зниження затрат, підвищення гнучкості та прискореного впровадження інновацій.

AWS представляє дуже багато сервісів, проте у цьому проекті використовується всього три – AWS Lambda, Amazon API Gateway та Amazon RDS. Про AWS Lambda, Amazon API Gateway було розказано у попередньому розділі, а служба реляційних баз даних Amazon (Amazon RDS) дозволяє просто налаштовувати, використовувати та масштабувати реляційні бази даних в хмарах. Сервіс забезпечує економічне та масштабоване використання ресурсів при одночасній автоматизації трудових завдань адміністрування, таких як вироблення апаратного забезпечення, налаштування бази даних, встановлення виправлень та резервне копіювання. Це дозволяє зосередити увагу на додатках, щоб забезпечити для них високу продуктивність, доступність, безпеку та сумісність. У якості RDS провайдеру я використав PostgreSQL.

2.5. Опис структури програми та алгоритмів її функціонування

Після того як відправлене користувачем повідомлення оброблюється посередницьким Telegram сервером, воно за допомогою вебхука потрапляє на наш сервер. З цього моменту починається робота з @neighborhelper_bot.

Основні типи повідомлень:

- текстове повідомлення;
- фото;
- геолокація;
- колбек-запит.

Кожний тип потрібно оброблювати певним чином. Визначення типу виконується в обробнику за шляхом *'src/features/index.js'*.

Текстовий обробник є найпростішим. Він відповідає за обробку звичайних текстових повідомлень. Наприклад, перша команда, з якої починається робота з ботом є /start. Ця команда і є першим текстовим повідомленням, яке потрапить у текстовий обробник. Натискаючи на кнопку, яка знаходиться замість клавіатури, користувач також відправляє текстове

повідомлення, просто не пишучи його вручну. Отже, усі команди та кнопки замість клавіатурі повинні бути оброблені у текстовому обробнику заздалегідь. На кожне текстове повідомлення може бути свій кейс. Тому всередині текстового обробника знаходиться switch-case с константами, які розміщені за шляхом *'src/constants/button.text.js'*. Якщо ж конструкція дійшла до default, відбувається перевірка на те, чи не запросив бот у попередньому кроці довільне текстове повідомлення, наприклад ім’я для створюваного повідомлення. Ця перевірка відбувається за допомогою стану, про який буде розказано далі.

Обробники фото і геолокацій схожі, тому що вони оброблюють лише один тип повідомлення – фото та геолокацію відповідно. Конструкція switch-case у них трохи відрізняється від текстового обробника, адже відправляючи команду у нього ми зазвичай знаємо, що користувач зробив конкретний запит і йому потрібна конкретна відповідь, окрім тих випадків коли відповідь базується на стані. У випадках з обробниками фото і геолокацій усі відповіді залежать від стану.

Колбек запити відбуваються тоді, коли користувач натискає на кнопку (рис. 2.8), яка знаходиться не на місці клавіатури, а у самому тексті повідомлення. Ці два типи кнопок відрізняються і називаються buttons та inline buttons.

Коли користувач натискає на inline кнопку (рис. 2.9), Telegram виконує колбек запит, відправляючи на сервер розробників callback_query. Глобальний обробник визначає, що callback_query не є пустим, отже потрібно передати контроль обробнику колбек-запитів, у якому вже і відбувається switch-case, проте в залежності від callback_data.

Звичайні кнопки знаходяться на місці клавіатури. Коли бот дає можливість взаємодіяти за допомогою кнопок, то праворуч від поля для вводу тексту є кнопка перемикання на клавіатуру. Це означає, що ми можем спілкуватися з ботом або за допомогою кнопок з уже написаним текстом, або вводячи команди самостійно. Таким чином при натисканні на кнопку на

сервер відправляється текстове повідомлення з її значенням. Тобто, якщо на кнопці написано «Знайти об’яву», то при натисканні відправиться текст «Знайти об’яву», який буде оброблений текстовим обробником.

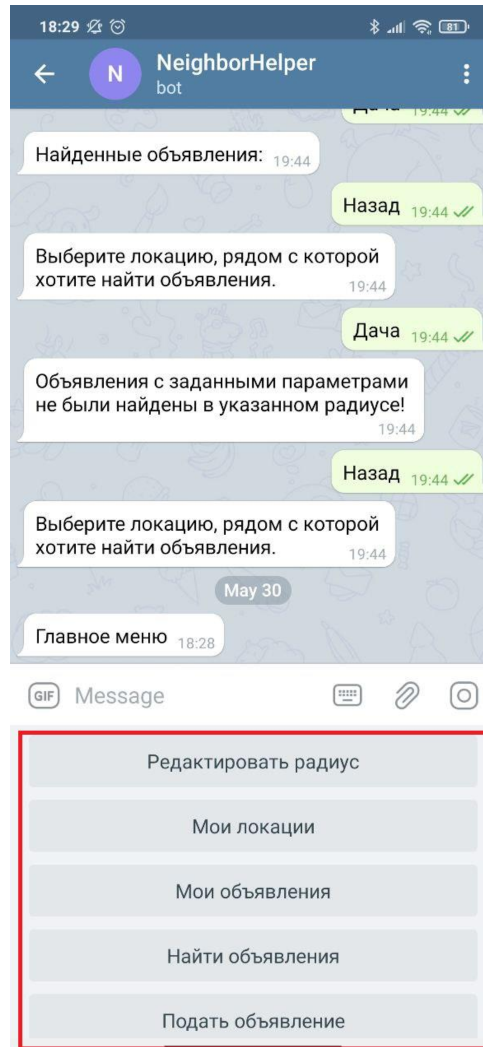

Рис. 2.8. Звичайні кнопки

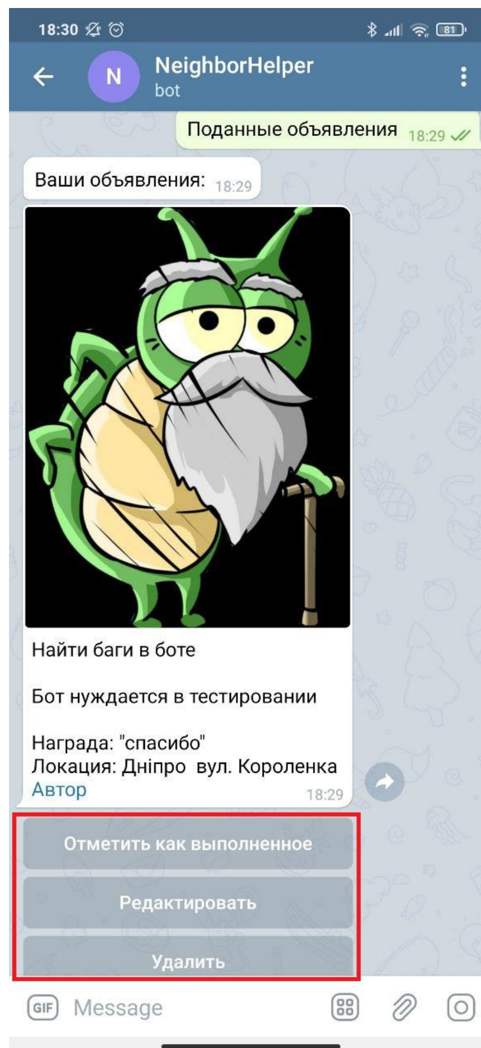

Рис 2.9. Inline кнопки

Inline кнопки дуже відрізняються від звичайних. По перше, вони розташовані не на місці клавіатури, а в самому повідомленні. Коли розробник конструює повідомлення, він додає до нього текст, редагує його, а також може додати спеціальні inline кнопки. Inline кнопка складається з двух полів: text - напис на самій кнопці та callback_data – інформація вшита у кнопку, яка буде відправлена назад на сервер у випадку її натискання.

Стан, як термін, згадувався декілька разів, та без нього робота даного бота була б неможливою, адже він використовується в усіх обробниках.

Для прикладу, візьмемо процес подачі об'яви. Користувачу потрібно вибрати категорію об'яви, її ім'я, опис, локацію, нагороду і фото. Тобто для створення об'яви необхідно виконати 6 кроків. Є декілька проблем.

Перша полягає у тому, що ми не знаємо, який крок був попереднім. Це важливо, коли користувач відправляє фото, адже невідомо при яких обставинах це фото було надіслане. Його могли відправити і з головного меню, тому нам потрібно знати, що даний користувач знаходиться у режимі подачі об'яви на кроці завантаження фотографії. Таким чином ми дізнаємося, що завантажене фото користувач хоче встановити як картинку своєї об'яви. Іншим прикладом є кнопка «Назад». Без зберігання поточного стану, неможливо визначити, який крок був попереднім.

Другою проблемою є те, що після вводу імені об'яви на стороні серверу спрацьовує текстовий обробник, клієнту відправляється наступне повідомлення, та саме ім'я нікуди не зберігається. Тобто разом з поточним режимом та кроком, потрібно кешувати дані, введенні користувачем. Після заповнення інформації про об'яву та натискання на кнопку «Опублікувати об'яву» усі дані з витягуються з кешу та їх основі створюється об'ява.

У рамках цього проекту кожен режим роботи, будь то подача, пошук об'яв, створення локації і т. д., названий flow, а кожен його крок flow_step, і розміщені вони за шляхом *'src/constants/flow.step.js'*.

```
28  exports.ADD_ASSIGNMENT_FLOW_STEPS = {
29    CHOOSE_CATEGORY: 'addAssignment.chooseCategory',
30    ADD_TITLE: 'addAssignment.addTitle',
31    ADD_DESCRIPTION: 'addAssignment.addDescription',
32    CHOOSE_LOCATION: 'addAssignment.chooseLocation',
33    ADD_REWARD: 'addAssignment.addReward',
34    SHOW_ASSIGNMENT: 'addAssignment.showAssignment',
35    PUBLISH_ASSIGNMENT: 'addAssignment.publishAssignment',
36  };
```

Рис. 2.10. flow_steps для подачі об'яви

Тут (рис. 2.10) можна побачити, що до крапки йде загальне ім’я flow (addAssignment), а після сам flow_step. В обробниках цей рядок розбивається на два, та в залежності від них викликаються відповідні функції.

У цьому проекті три сервіси – user.service, location.service, assignment.service. Ці сервіси відповідають за взаємодію між кодом та базою даних (рис. 2.11), виконуючи CRUD-операції. Прості функції використовують Sequelize для побудови SQL-запитів, складні, такі як getAllNearby (функція пошуку об’яв у заданому користувачем радіусі) записані у нативному SQL. Усі функції сервісів повертають відповідь у обгортці ServiceResponse, яка складається з чотирьох полів – succeeded (результат виконання), message (повідомлення у разі виникнення помилки), model (інформація, яка повертається), pagingData (допоміжне поле, коли потрібно повернути список записів). Сервіси викликаються методами, які називаються actions.

Після того, як обробник спрацював, потрібний стан встановлено, йде виклик проміжного коду. Ці проміжні функції називаються Actions, та вони відповідають за виклик сервісів, зберігання кешу та повернення необхідного шаблону повідомлення (templates).

Шаблони зберігаються за шляхом ‘src/templates’. Вони є тими повідомленнями, та набором кнопок, які повинні відправитися користувачу. У кінці функцій Actions повертаються шаблони самого повідомлення та звичайних кнопок, якщо вони потрібні (рис. 2.12).

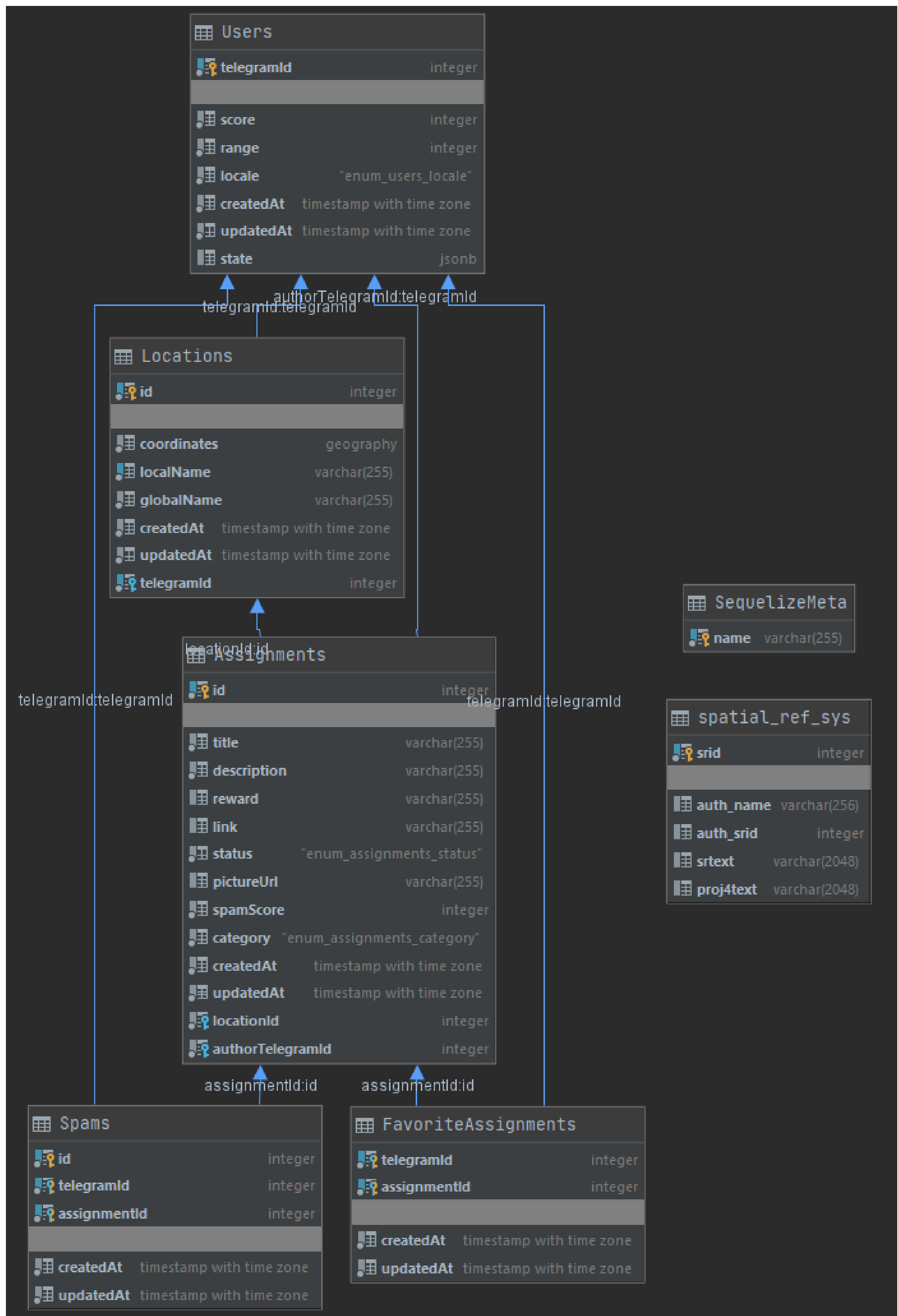

Рис. 2.11. Схема бази даних

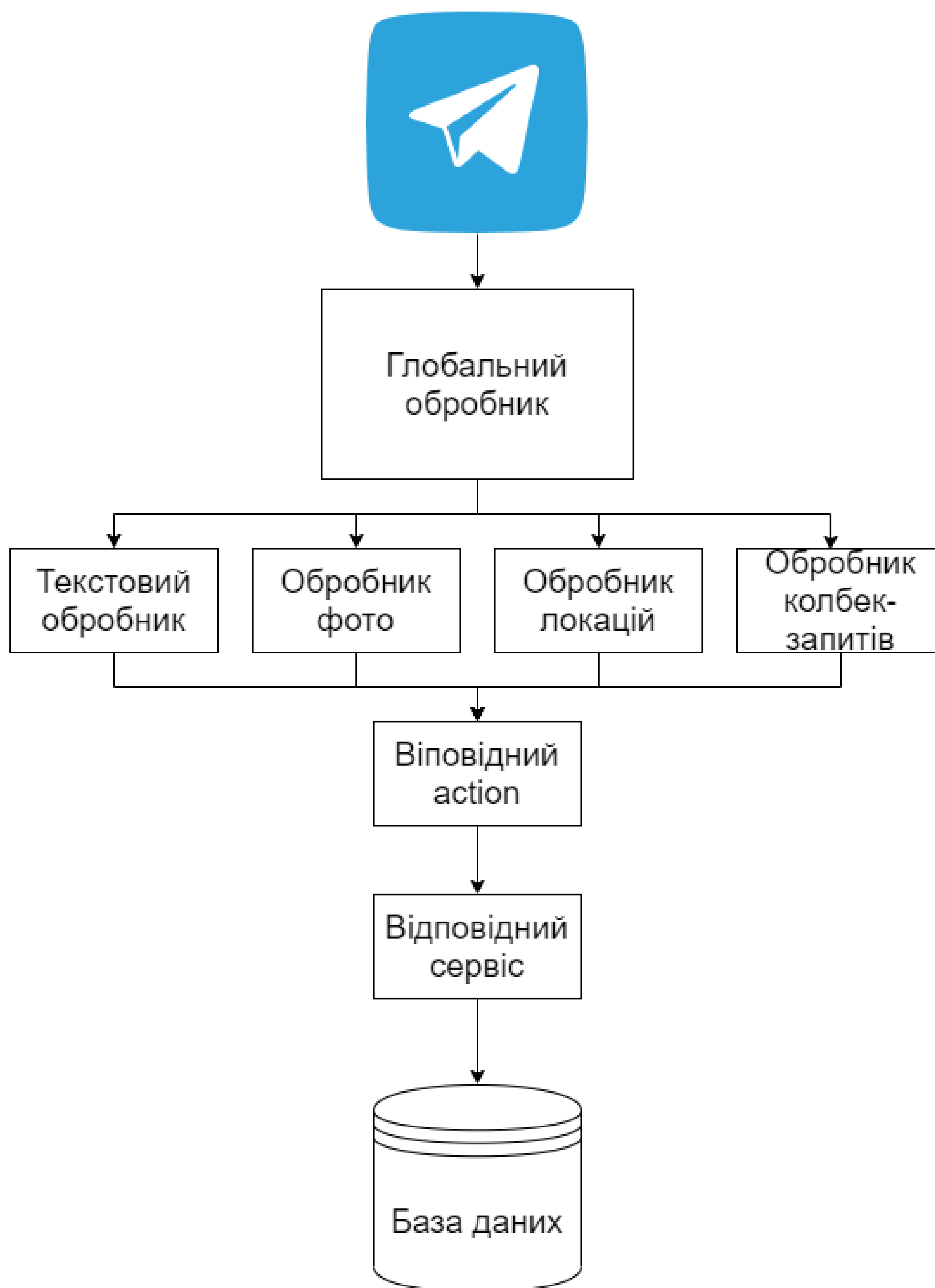

Рис. 2.12. Схема обробки запитів

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними та вихідними є дані, що безпосередньо стосуються об’яв, що через бота і публікуються. Об’ява має такі загальні атрибути як назва, опис, категорія, локація, винагорода, фото та автор. Якщо роздивлятись Telegram-бота як програмний продукт в глобальному сенсі, то вхідні та вихідні дані передаються у форматі JSON або form-data. Вхідні дані передаються через інтерфейс Telegram за допомогою кнопок, текстового поля фотографій та геолокацій. Вихідними даними буде інформація про вже створені об’яви у форматі JSON або form-data.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Бот розміщений на AWS Lambda – сервіс який автоматично розширюється, коли навантаження збільшується. Проте конкретні специфікації про обладнання не зазначені.

2.7.2. Використані програмні засоби

Під час розробки даного застосунку були використані такі програмні засоби:

– JetBrains WebStorm;
– JetBrains DataGrip;
– Git, Github;
– AWS CLI;
– Claudia CLI;

WebStorm - це інтегроване середовище розробки на JavaScript, CSS & HTML від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA.

WebStorm забезпечує автодоповнення, аналіз коду на льоту, навігацію по коду, рефакторинг, налагодження, і інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (в тому числі, рефакторинг коду JavaScript, що знаходиться в різних файлах і папках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладено Javascript) – тобто в таких конструкціях підтримується коректний рефакторинг.

Програмне забезпечення JetBrains DataGrip є IDE-інструмент для роботи з базами даних MySQL, PostgreSQL, Oracle, SQL Server, Sybase, DB2, SQLite, HyperSQL, Apache Derby і H2. Рішення JetBrains DataGrip включає потужний текстовий редактор, забезпечує синтаксичну виділення коду, підтримує інтеграцію з системами контролю версій Git, Subversion і т.д.

Git - розподілена система контролю версій, яка дає можливість розробникам відстежувати зміни в файлах і працювати над одним проектом спільно з колегами. Вона була розроблена в 2005 році Лінус Торвальдс, творцем Linux, щоб інші розробники могли вносити свій вклад в ядро Linux. Git відомий своєю швидкістю, простим дизайном, підтримкою нелінійної розробки, повної децентралізацією і можливістю ефективно працювати з великими проектами.

GitHub – це платформа для розміщення коду для контроля версій та співпраці. Він дозволяє спільно працювати над проектами з будь-якого місця та зберігати версії проекту на віддаленому сервері.

Інтерфейс командного рядка AWS – це єдиний інструмент для управління сервісами AWS. Завантаживши всього один засіб, можна контролювати безліч сервісів AWS з командного рядка і автоматизувати їх за

допомогою скриптів. Використовується для встановлення приватного IAM ключу, який використовує Claudia при автоматичному деплої.

Інтерфейс командного рядка Claudia дозволяє швидко створити, налаштувати або видалити AWS Lambda з написаним у проекті кодом.

2.7.3. Виклик та завантаження програми

Для того щоб розпочати роботу з ботом користувачу необхідно завантажити додаток Telegram через Google Play Market чи App Store та через пошук знайти користувача, ввівши @neighbor_helper. Після цього потрібно прописати команду /start чи натиснути кнопку Start.

2.7.4. Опис інтерфейсу користувача

Через головне меню користувач може змінити радіус пошуку об’яв.

Також через головне меню користувач може зайти в меню «Мої локації». Там він може створити нову локацію або видалити існуючу. Локації потрібні для того, щоб створювати або шукати об’яви. Під час створення локації користувач може відіслати свою поточну локацію або вибрати будь-яку на мапі. Також для вже створеної локації користувач може змінити координати.

Користувач може створювати об’яву, вказуючи її назву, опис, категорію, локацію, фото та нагороду. Таким чином інші користувачі зможуть знайти цю об’яву, якщо вони знаходяться поруч з нею.

Під час пошуку об’яви користувач вибирає категорію. В якості результату будуть показані об’яви, які знаходяться у заданому користувачем радіусі. Знайдені об’яви можна додати до вибраних або поскаржитися на спам. При 5 скаргах об’ява видаляється.

Через головне меню користувач може зайти у «Свої локації». Там можна подивитися створені локації та вибрані. Створену локацію можна редагувати,

видалити або відмітити як виконану. Виконані об’яви не будуть показуватися іншим користувачам під час пошуку. Це створено для того, щоб знаходженні виконувача об’яви можна було тимчасово викрити об’яву без її видалення. При необхідності статус об’яву можна змінити. Також присутня пагінація.

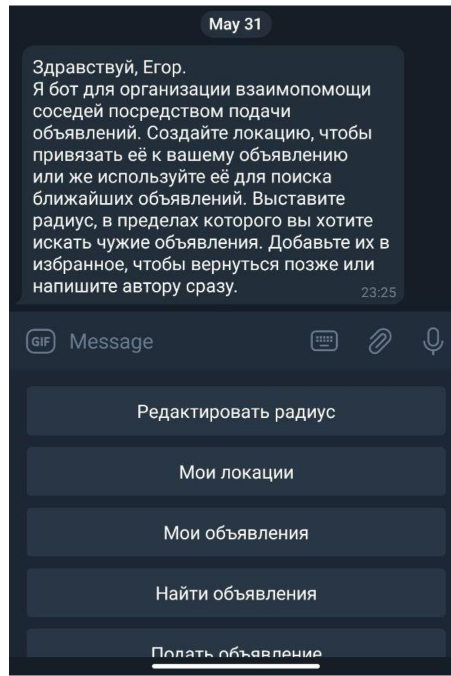

Рис. 2.13. Стартова сторінка бота

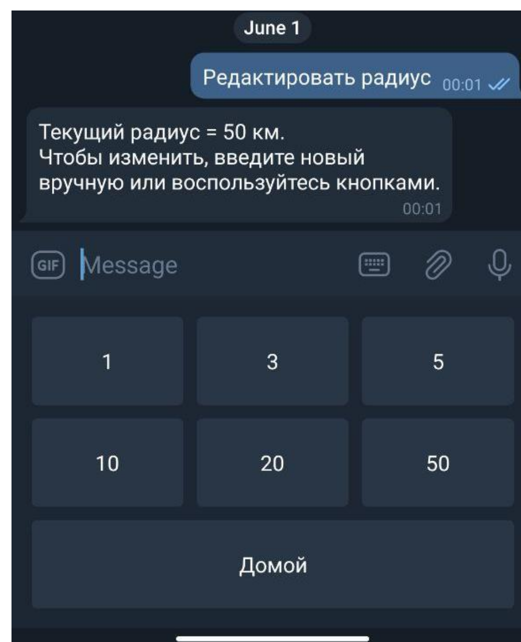

Рис. 2.14. Зміна радіусу, який буде використано для пошуку найближчих об’яв

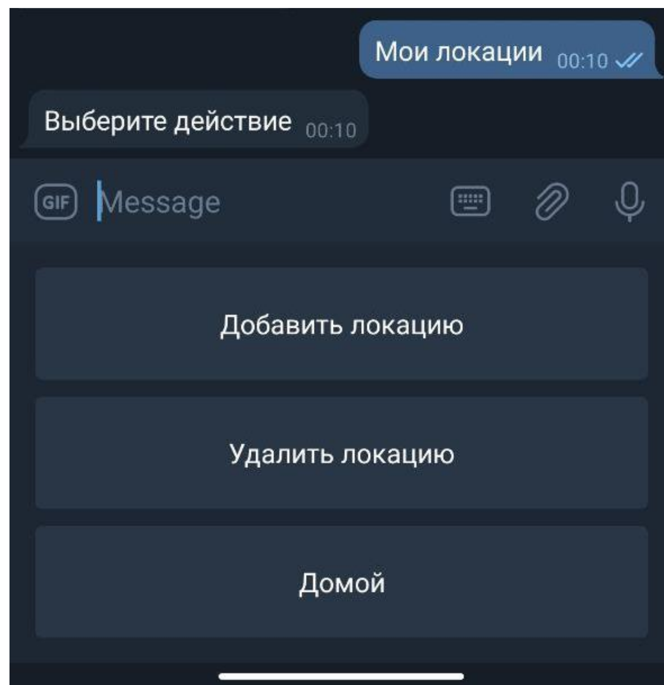

Рис. 2.15. Меню локацій користувача

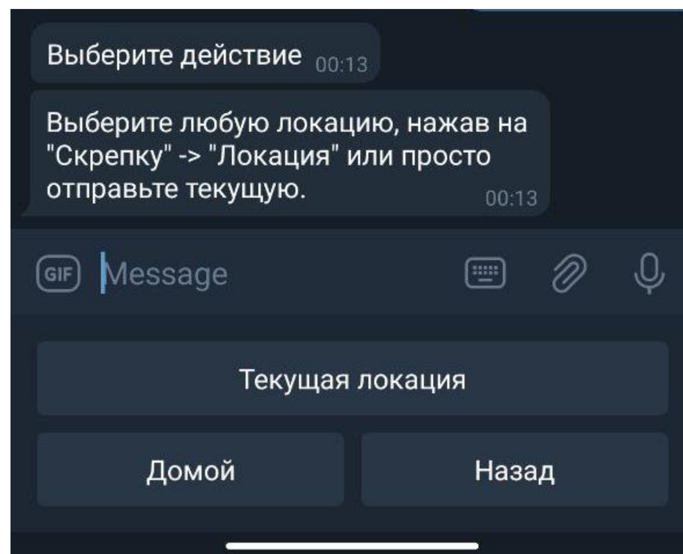

Рис. 2.16. Створення локації користувача, перший крок

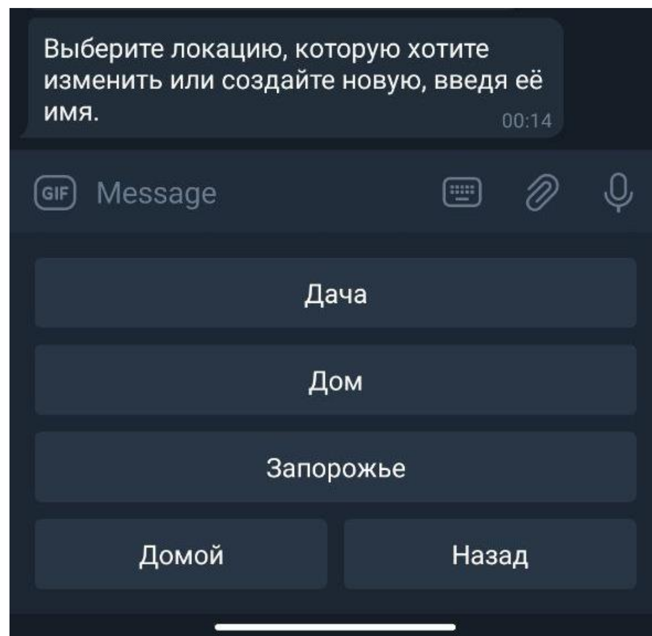

Рис. 2.17. Створення локації користувача, другий крок

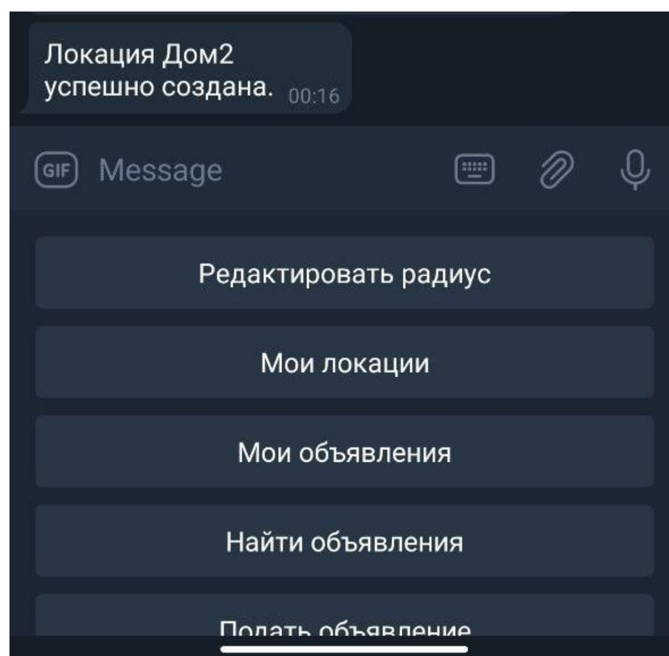

Рис. 2.18. Створення локації користувача, завершення

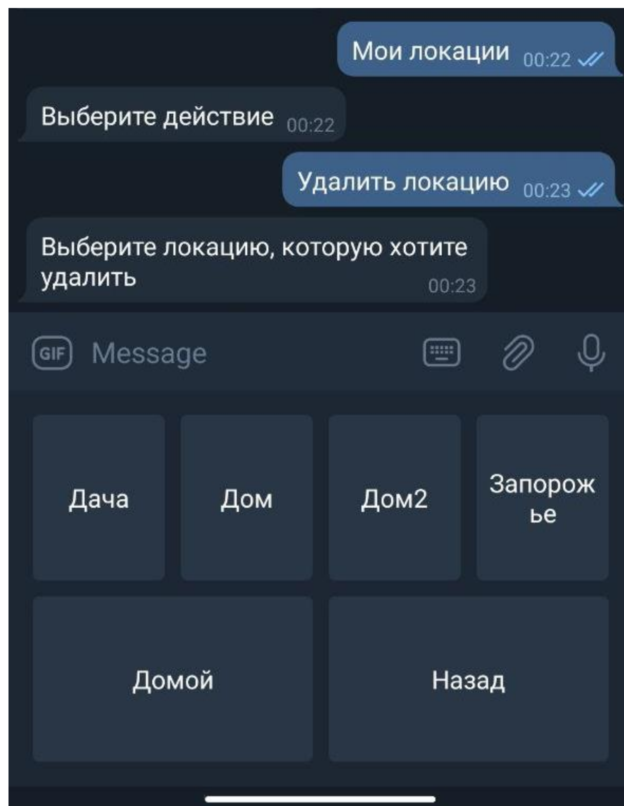

Рис. 2.19. Видалення локації користувача, перший крок

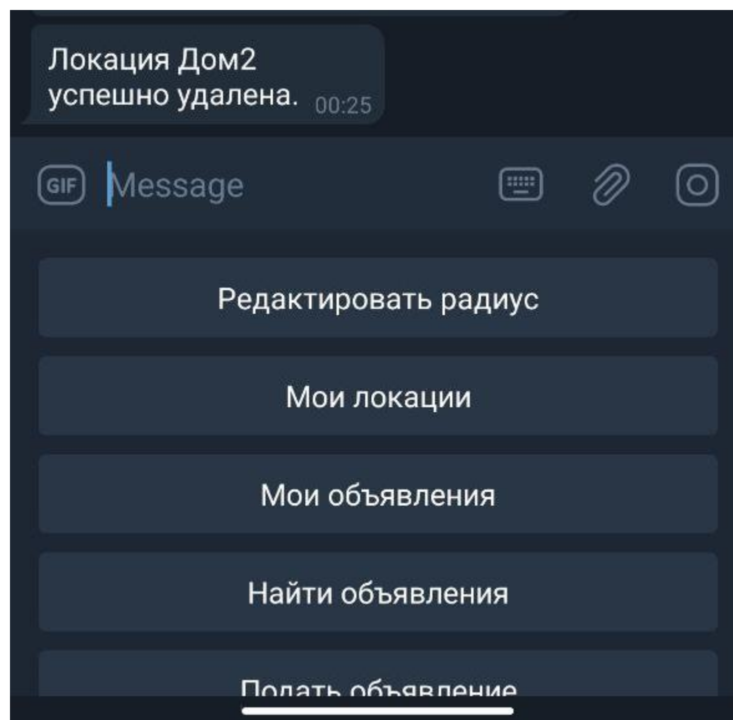

Рис. 2.20. Видалення локації користувача, завершення

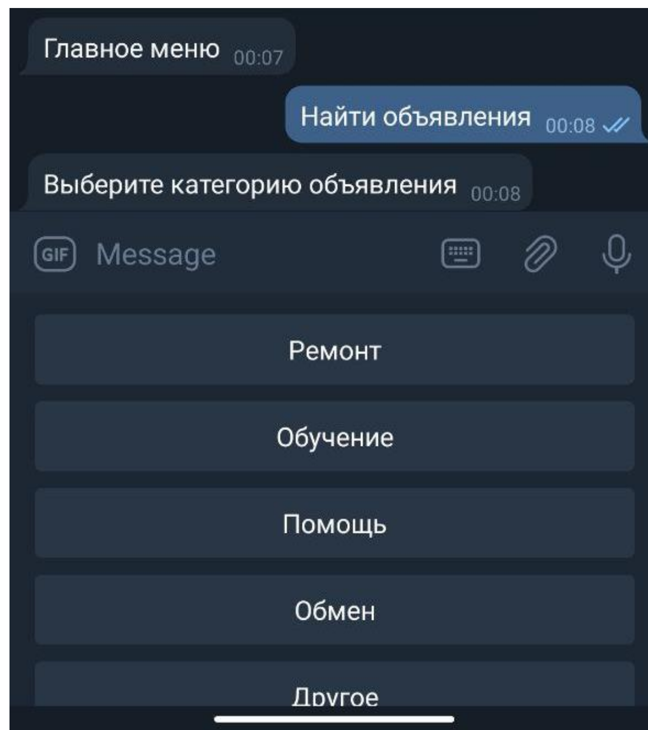

Рис. 2.21. Перший крок пошуку об’яв (вибір категорії)

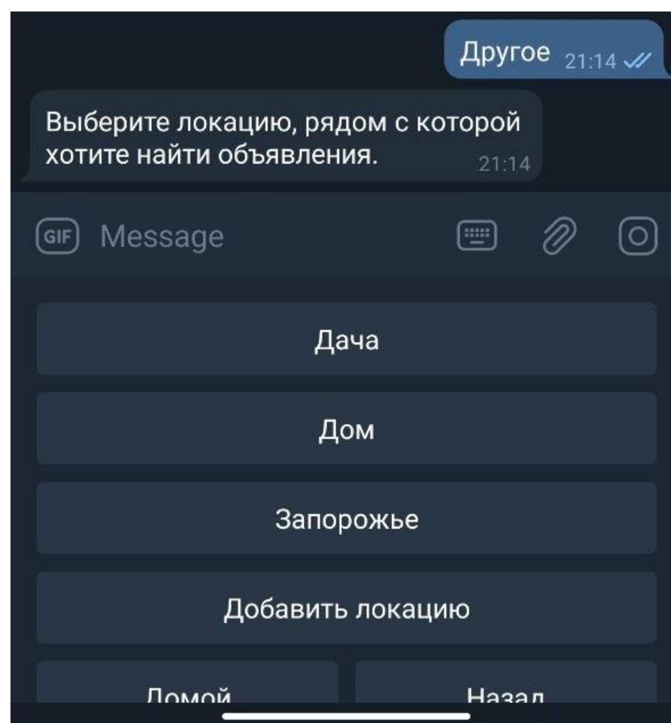

Рис. 2.22. Другий крок пошуку об’яв (вибір локаціїї)

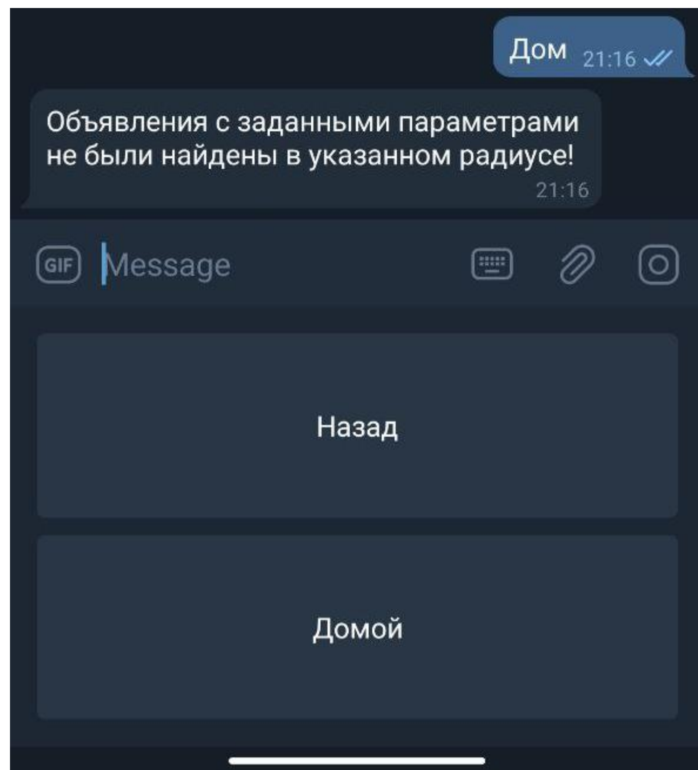

Рис. 2.23. Третій крок пошуку об’яв (вибір категорії)

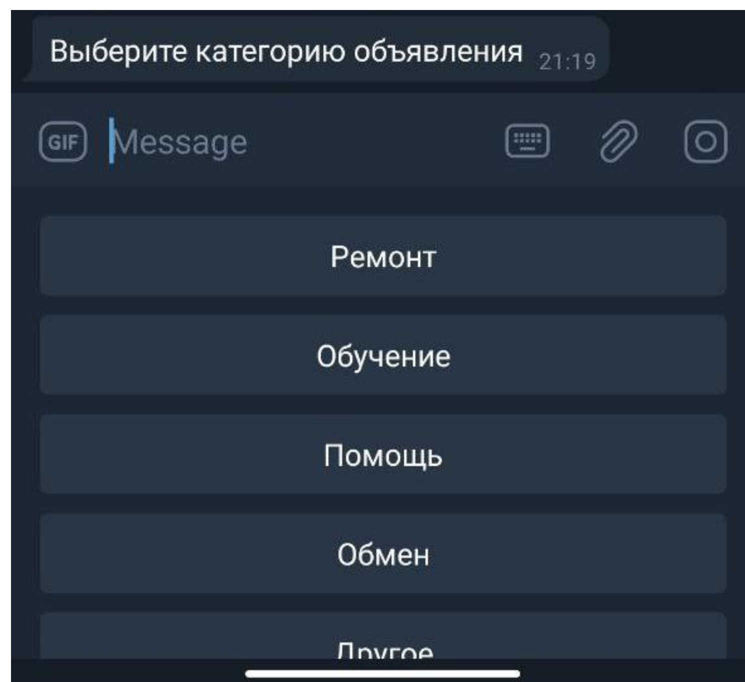

Рис. 2.24. Перший крок створення/редагування об’яв (вибір категорії)

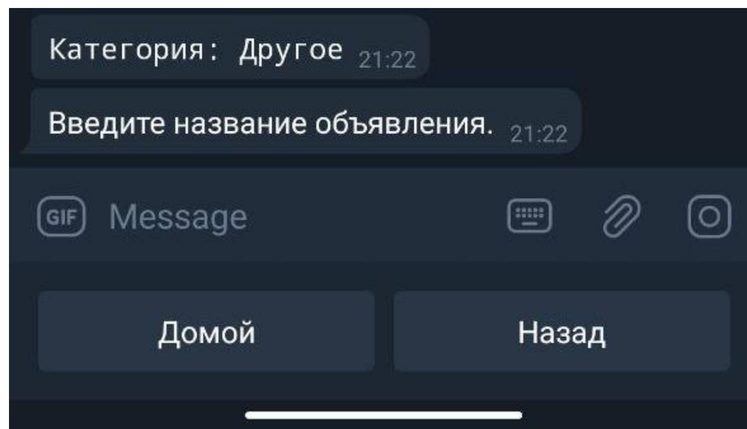

Рис. 2.25. Другий крок створення/редагування об’яв (введення назви об’яви)

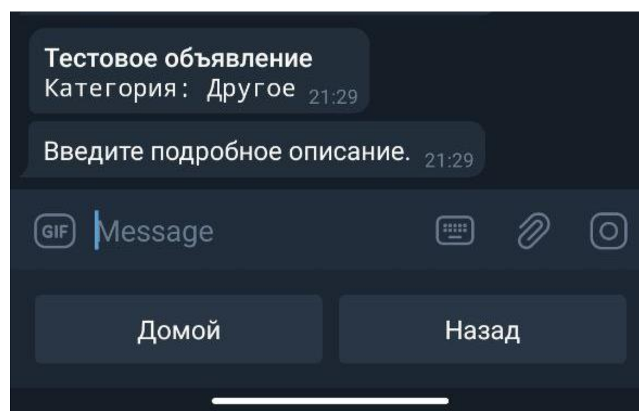

Рис. 2.26. Третій крок створення/редагування об’яв (введення опису об’яви)

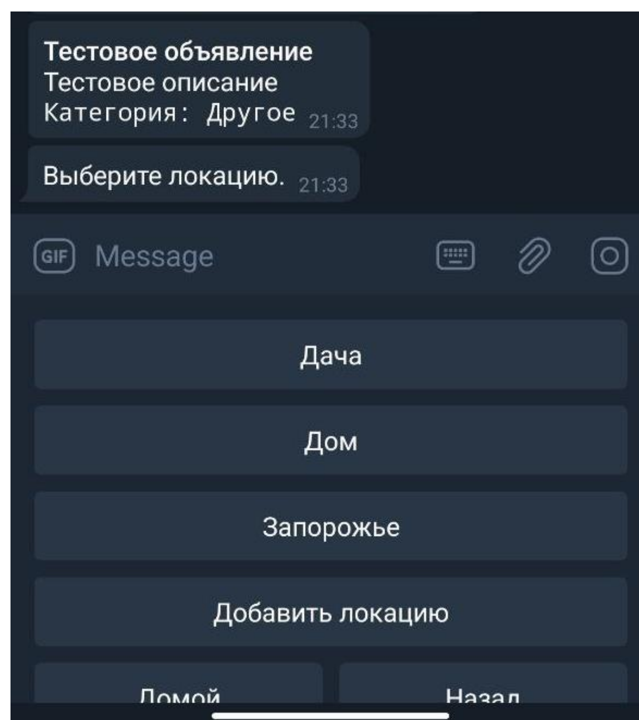

Рис. 2.27. Четвертий крок створення/редагування об’яв (вибір локації об’яви)

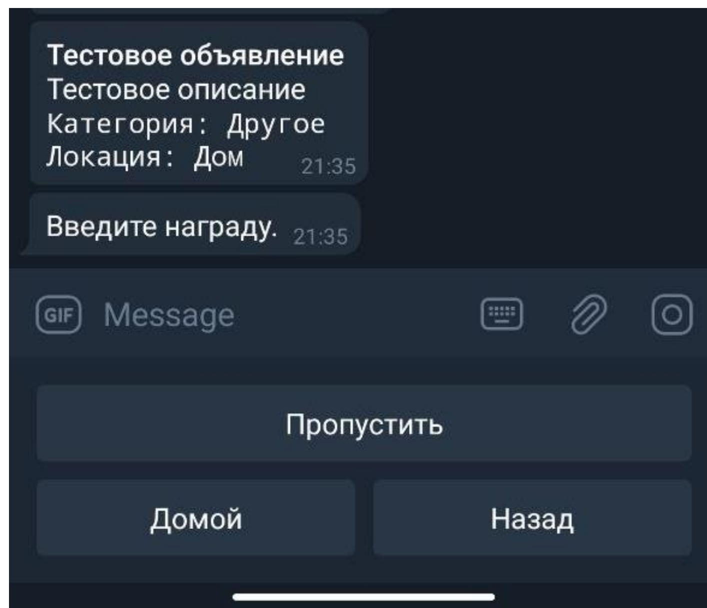

Рис. 2.28. П’ятий крок створення/редагування об’яв (введення нагороди)

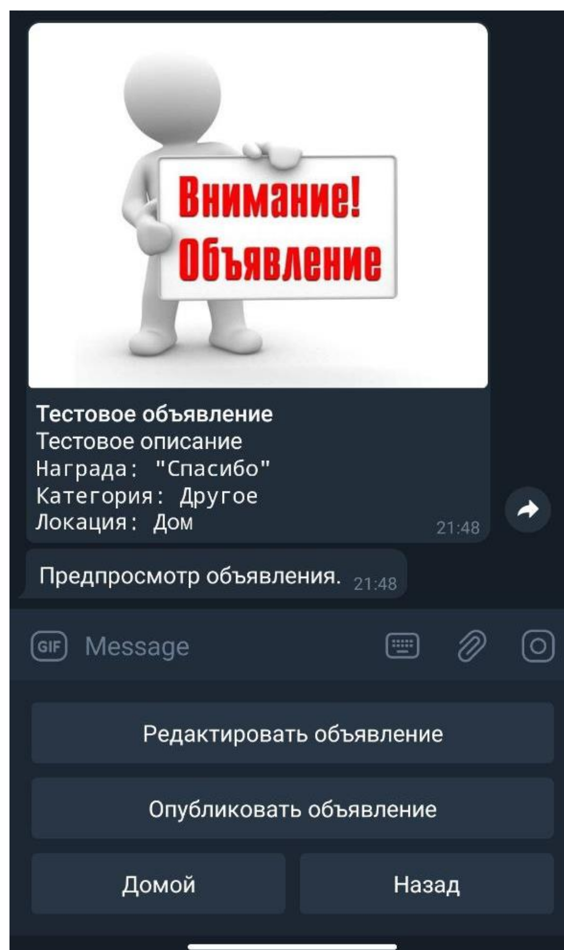

Рис. 2.29. Шостий крок створення/редагування об’яв (завершення)

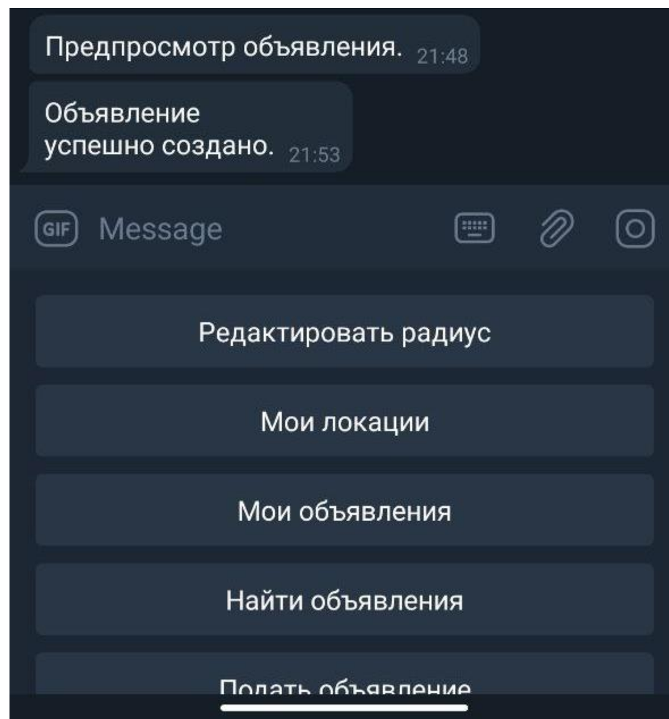

Рис. 2.30. Повернення до меню після створення/редагування об’яви

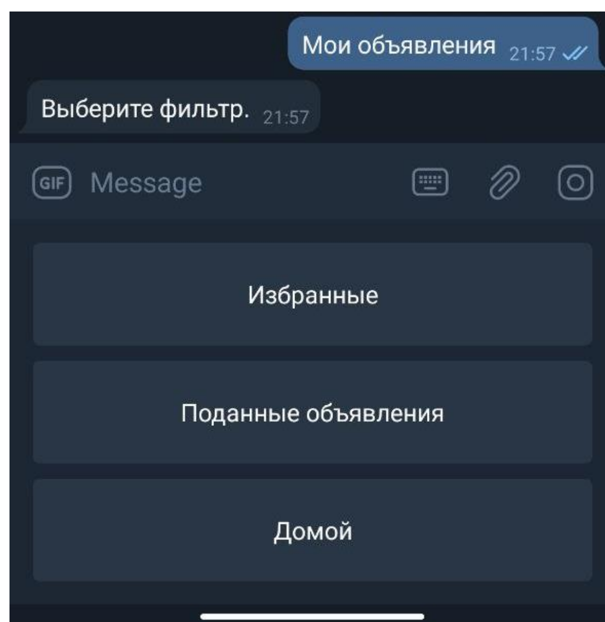

Рис. 2.31. Меню «Мої об’яви»

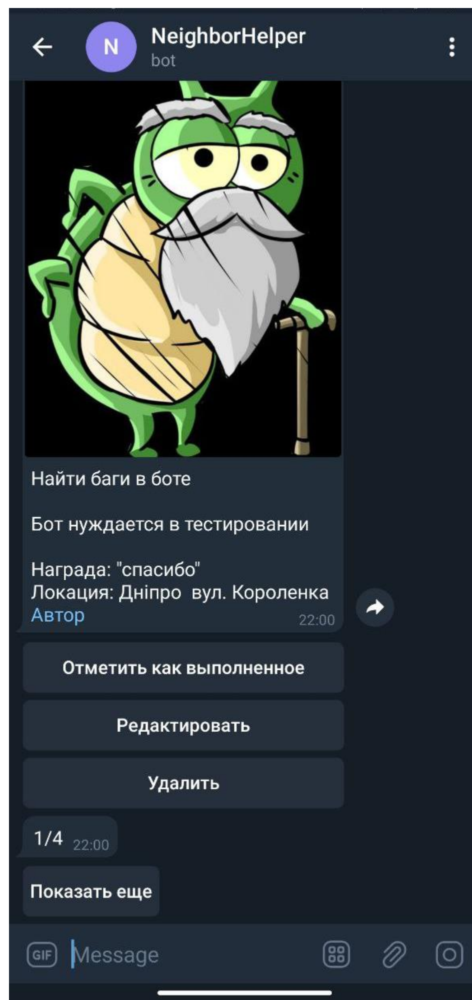

Рис. 2.32. Меню «Подані об’яви»

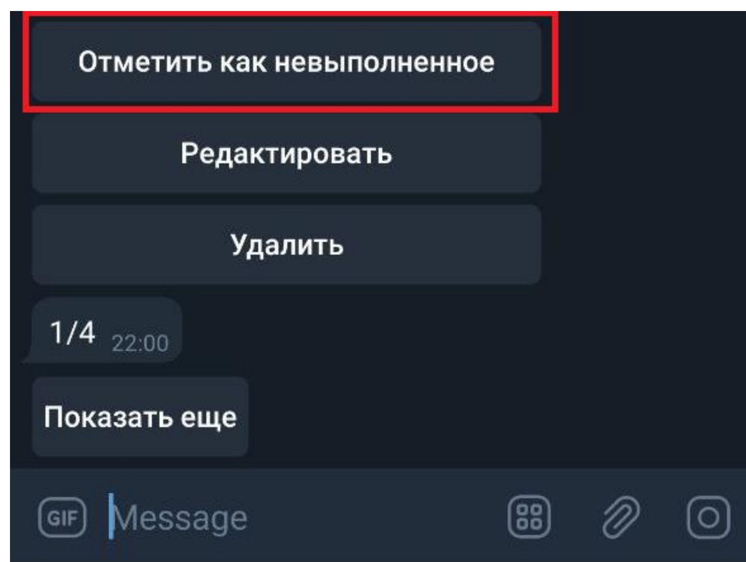

Рис. 2.33. Зміна статусу створеної об’яви

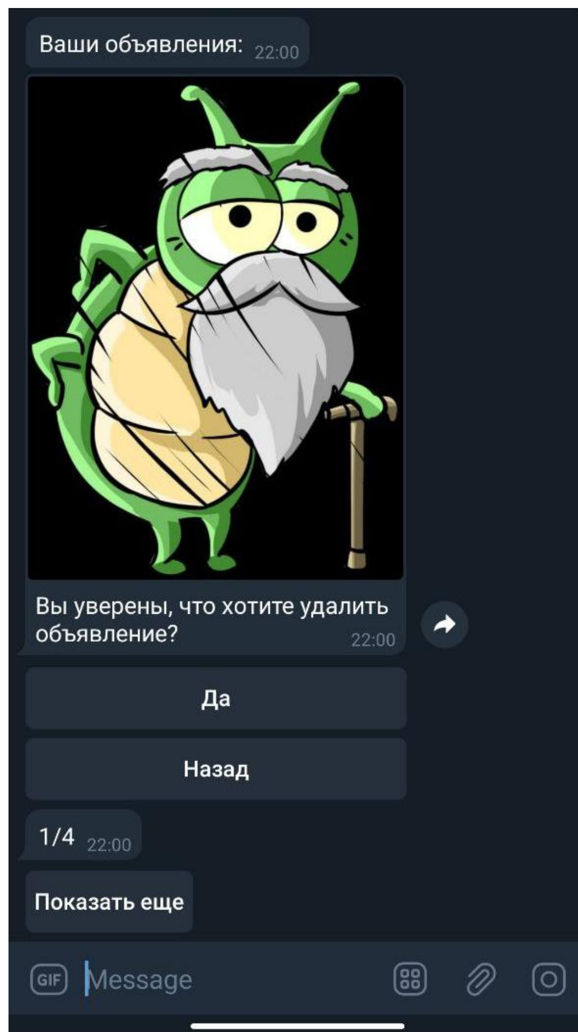

Рис. 2.34. Видалення створеної об’яви

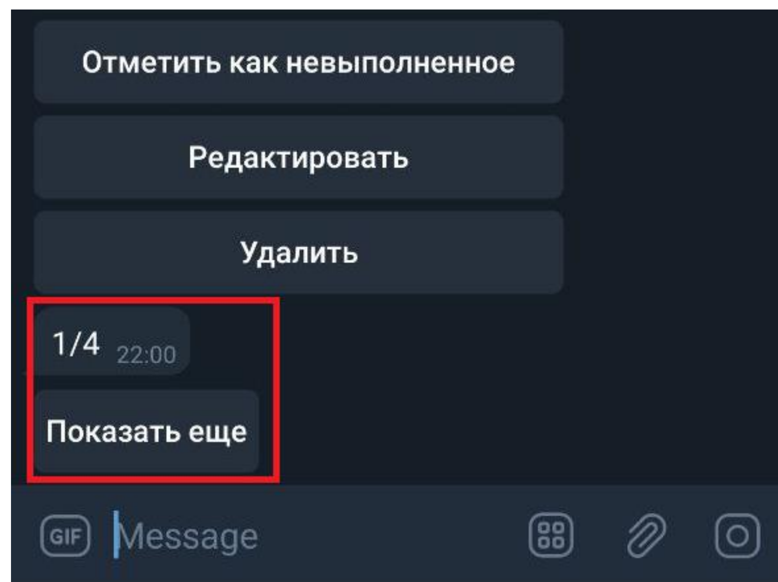

Рис. 2.35. Пагінація

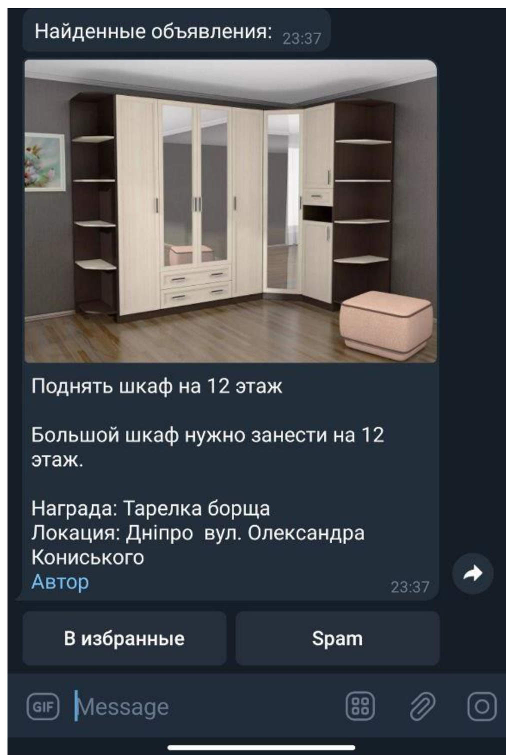

Рис. 2.36. Пошук об'яв

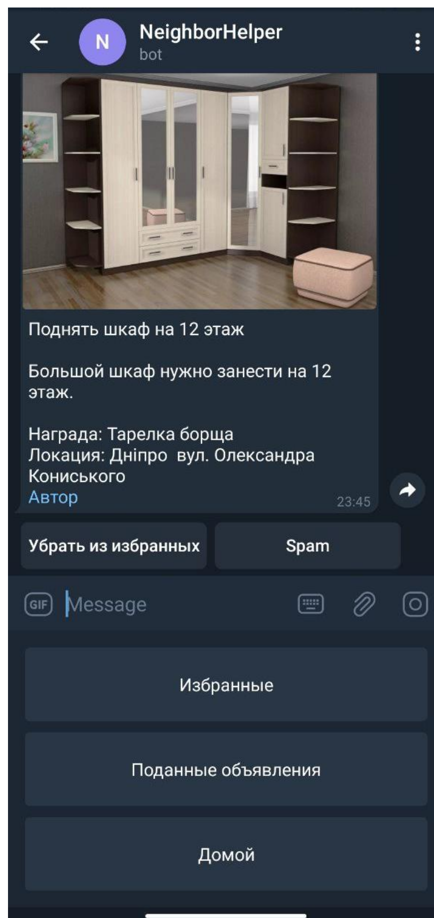

Рис. 2.37. Додані до вибраних об'яви

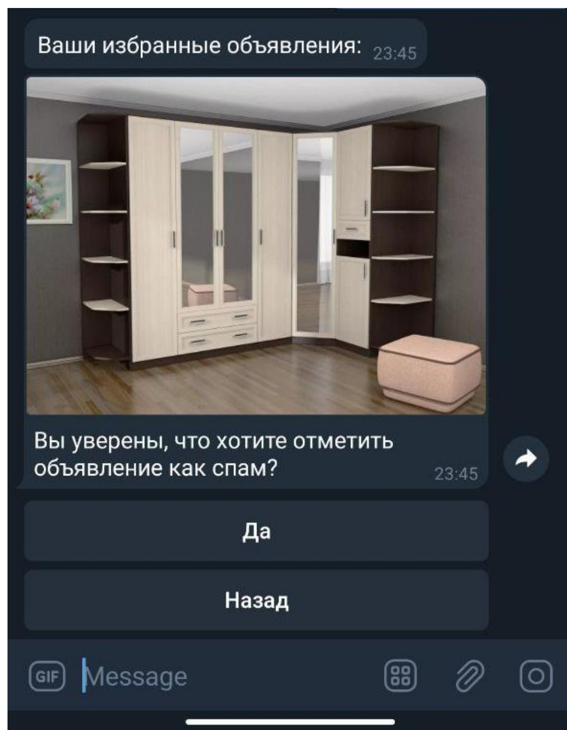

Рис. 2.38. Скарга на спам

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

Автоматизуючи суспільні процеси за допомогою ПЗ важливо визначити трудомісткість розробки програмного забезпечення і розрахувати витрати на створення програмного продукту.

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 1260;
2. коефіцієнт складності програми – 1,25;
3. коефіцієнт корекції програми в ході її розробки – 0,05 (ресурс: https://ain.ua/2021/01/11/zarplaty-ukrainskix-programmistov-zima-2021/);
4. годинна заробітна плата програміста – 130 грн/год;

Середня годинна зарплата програміста була вирахувати виходячи зі статики порталу Ain.ua. Середньоукраїнська заробітна плата junior-програміста, складає близько 800 доларів у місяць. При курсі валют НБУ на початок червня 2021 року один американський долар дорівнює 27,1 грн, тому середня зарплата в гривнях дорівнює 21683 грн. При восьмигодинному робочому дні (176 годин в місяць в середньому) середня зарплата за годину буде становити 130 грн.

Джерела:

- Веб-сайт «Українська спільнота програмістів», URL: https://jobs.dou.ua/salaries/#period=dec2020&city=all&title=Software%20Engineer&language=Java&spec=&exp1=0&exp2=1
- Веб-сайт НБУ, URL: https://bank.gov.ua/ua/markets/exchangerate-chart?cn%5B%5D=USD

– коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

– коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;

– вартість машино-години ЕОМ – 14 грн/год.

Звичайний комп'ютер споживає 180 Ватт / год, що дорівнює 0,18 кВт / год. Вартість одного кВт на годину від постачальника Yasno на стан середини 2021 року дорівнює 1,68 грн в незалежності від обсягом споживання. Отже вартість електроенергії споживаної комп'ютером за годину дорівнює 0,30 грн. Ціна довгострокової оренди комп'ютера дорівнює 2400 грн на місяць, з чого можна вирахувати, що ціна оренди на годину буде дорівнювати 13,63 грн. Остаточна вартість машино-години ЕОМ дорівнює 14 грн за годину.

Джерела:

- Веб-сайт постачальника електроенергії Yasno, URL: https://yasno.com.ua/b2c-tariffs
- Веб-сайт компанії орендатора комп'ютерів «ChipChip», URL: https://komputers.com.ua/arenda-kompyutera/

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1+p), \text{ де} \quad (3.2)$$

Q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1260 \cdot 1{,}25 \cdot (1+0{,}05)=1653{,}75;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u= \frac{Q \cdot B}{(75…85) \cdot K}, \text{ людино-годин,} \quad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{1653{,}75 \cdot 1{,}2}{85 \cdot 0{,}8}=29{,}183, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \ldots 25) \cdot K}; \quad (3.4)$$

$$t_a = \frac{1653{,}75}{20 \cdot 0{,}8} = 103{,}359,$$ людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \ldots 25) \cdot K}; \quad (3.5)$$

$$t_n = \frac{1653{,}75}{25 \cdot 0{,}8} = 82{,}6875,$$ людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4 \ldots 5) \cdot K}; \quad (3.6)$$

$$t_n = \frac{1653{,}75}{5 \cdot 0{,}8} = 413{,}4375,$$ людино-годин,

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1{,}5 \cdot t_{отл}; \quad (3.7)$$

$$t_{отл}^{к} = 1{,}5 \cdot 413{,}4375 = 620{,}16,$$ людино-годин

Витрати праці на підготовку документації:

$$t_{\partial} = t_{\partial p} + t_{\partial o}; \quad (3.8)$$

де $t_{\partial p}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{\partial} = \frac{Q}{(15\ldots20)\cdot K}; \quad (3.9)$$

$$t_{\partial} = \frac{1653,75}{20 \cdot 0,8} = 103,36 \text{ людино-годин.}$$

$t_{\partial o}$– трудомісткість редагування, печатки й оформлення документації

$$t_{\partial o} = 0,75 \cdot t_{\partial p}; \quad (3.10)$$

$$t_{\partial o} = 0,75 \cdot 103,359 = 77,5192, \text{ людино-годин.}$$

$$t_{\partial} = 77,5192 + 103,359 = 180,878, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

$$t = 77 + 29,1313 + 29,183 + 82,6875 + 620,16 + 180,878 = 1019,0398,$$
людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1084,5663 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ включають витрати на заробітну плату виконавця програми з/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн}, \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}\text{, грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 1019{,}0398 \cdot 140 = 142\ 665{,}572 \text{ грн.}$$

$З_{МВ}$ – Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М}\text{, грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 620{,}16 \cdot 14 = 8682{,}24 \text{ грн.}$$

$$К_{ПО} = 142\ 665{,}572 + 8682{,}24 = 151\ 347{,}812 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}\text{, мес.} \qquad (3.14)$$

де B_k - число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{1019{,}0398}{1 \cdot 176} = 5{,}7 \text{ міс.}$$

Висновки. На розробку даного програмного забезпечення піде 1019,0398 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 6,1 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 151 347,812 грн.

ВИСНОВКИ

В даній кваліфікаційній роботі був розроблений Telegram-бот, через який можна створювати та знаходити оголошення.

Це програмне забезпечення призначене для надання можливості швидкого пошуку людей, які можуть допомогти у різних ситуаціях. Під час подачі та пошуку об’яви користувач вибирає локацію. Таким чином, найпрактичнішим цей додаток буде для сусідів або людей, які близько живуть. Проте при бажанні можна збільшити радіус пошуку та шукати об’яви по всьому місту!

Практичне значення полягає у розробці платформи для подачі об’яв з прив’язаними до них геолокаціями. Сервіс дає можливість надати детальну інформацію про об’яву, взаємодіяти з ними (додати в вибране, поскаржитися, зв’язатися з автором) та встановити радіус пошуку.

Під час виконання даної кваліфікаційної роботи були виконані наступні задачі:

- проаналізовано предметну галузь задачі, що розв’язується;
- проведено порівняння з можливостей існуючих подібних застосунків;
- обрано раціональну структуру і параметри програми;
- написано програмний код додатку;
- розроблено рекомендації щодо використання програми.

Програма реалізована на мові програмування JavaScript з використанням Node.js.

Також у кваліфікаційній роботі було визначено трудомісткість розробленого програмного продукту 1019,0398 люд-год, проведений підрахунок вартості роботи по створенню програми 151 347,812 грн. та розраховано час на його створення 5,7 міс.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Modrzyk Nicolas. Building Telegram Bots: Develop Bots in 12 Programming Languages using the Telegram Bot API, 2019. — 277 с.

2. Сріні Джанарсанам. Розробка чат-ботів і розмовних інтерфейсів, 2016 – 340 с.

3. David Flanagan. JavaScript: The Definitive Guide Seventh Edition, 2021 – 722 с.

4. Marijn Haverbeke. Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming. O'Reilly Media, 2019 – 474 с.

5. Robert Martin. Clean code, 2019 – 368 с.

6. Ірина Бородкіна, Георгій Бородкін. Інженерія програмного забезпечення. Посібник для студентів вищих навчальних закладів, 2018 – 204 с.

7. Юрій Грицюк. Аналіз вимог до програмного забезпечення, 2018 – 456 с.

8. Calculate distance, bearing and more between Latitude/Longitude points. Movable Type Scripts. Online: https://www.movable-type.co.uk/scripts/latlong.html (переглянуто 01/06/2021)

9. What is a Geoid? Why do we use it and where does its shape come from? U.S. Geological Survey. Online: https://www.usgs.gov/faqs/what-a-geoid-why-do-we-use-it-and-where-does-its-shape-come?qt-news_science_products=0#qt-news_science_products (переглянуто 01/06/2021)

10. AWS Lambda in Action: Event-driven serverless applications, 2016 – 384 с.

11. Amazon Relational Database Service: User Guide, 2602 с.

12. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith, 2019 – 284 с.

13. PostgreSQL Query Optimization The Ultimate Guide to Building Efficient Queries, 2021 – 310 с.

14. PostGIS in Action, Third Edition, 2019 – 615 c.

15. Relational Databases using Node.js. O'Reilly. Online: https://www.oreilly.com/library/view/node-up-and/9781449332235/ch06s02.html (переглянуто 18/05/2021)

16. Володимир Гайдаржи, Ігор Изварин. Бази даних в інформаційних системах, 2018 – 418 с.

17. Євген Моргунов. PostgreSQL. Основи мови SQL. Навчальний посібник, 2019 – 336 с.

18. Node.js: The Ultimate Beginner's Guide to Learn node.js Step by Step - 2021 (3nd edition) – 2021, 129 c.

19. Functional JS #3: State. Medium. Online: https://medium.com/dailyjs/functional-js-3-state-89d8cc9ebc9e (переглянуто 27/05/2021)

20. Brent Laster. Professional Git. 1st Edition. Wrox, 2016. 433 c.

21. Telegram Bot API documentation. Офіційний сайт Telegram. Online: https://core.telegram.org/bots/api (переглянуто 25/05/2021)

22. PostGIS documentation. Офіційний сайт PostGIS. Online: https://postgis.net/docs/ (переглянуто 17/05/2021)

23. User Experience Design Process. UX Planet. Online: https://uxplanet.org/user-experience-design-process-d91df1a45916 (переглянуто 16/05/2021)

24. AWS Command Line Interface Documentation. Офіційний сайт AWS. Online: https://docs.aws.amazon.com/cli/index.html (переглянуто 27/05/2021)

25. Claudia Bot Builder documentation. Офіційний сайт Claudia. Online: https://claudiajs.com/documentation.html (переглянуто 18/05/2021)

ДОДАТОК А

КОД ПРОГРАМИ

Лістинг Telegram боту

bot.js

```
const botBuilder = require('claudia-bot-builder');
const messageHandler = require('./features');

const bot = botBuilder(messageHandler, {
  platforms: ['telegram'],
});

module.exports = bot;
```

user.js

```
const {
  Model,
} = require('sequelize');

const { USER_MODEL_NAME } =
require('../../constants').enums.databaseModel;

module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    static associate(models) {
      User.hasMany(models.Location, { foreignKey: 'telegramId' });
      User.hasMany(models.Assignment, { foreignKey: 'authorTelegramId', as:
'createdAssignments' });

      User.belongsToMany(models.Assignment, { through: 'FavoriteAssignments',
foreignKey: 'telegramId', as: 'favoriteAssignments' });

      User.belongsToMany(models.Assignment, { through: models.Spam,
foreignKey: 'telegramId', as: 'spamAssignments' });
    }
```

```
  }
  User.init({
    telegramId: {
      allowNull: false,
      primaryKey: true,
      type: DataTypes.INTEGER,
    },
    score: {
      allowNull: false,
      defaultValue: 0,
      type: DataTypes.INTEGER,
    },
    range: {
      allowNull: false,
      defaultValue: 3,
      type: DataTypes.INTEGER,
    },
    locale: {
      allowNull: false,
      type: DataTypes.ENUM,
      values: ['ua', 'ru', 'en'],
      defaultValue: 'en',
    },
    state: {
      type: DataTypes.JSONB,
    },
  }, {
    sequelize,
    modelName: USER_MODEL_NAME,
  });
  return User;
};
```

location.js

```
const {
  Model,
} = require('sequelize');

const { LOCATION_MODEL_NAME } =
require('../../constants').enums.databaseModel;

module.exports = (sequelize, DataTypes) => {
```

```
  class Location extends Model {
    static associate(models) {
      Location.belongsTo(models.User, { foreignKey: 'telegramId' });
      Location.hasOne(models.Assignment, { foreignKey: 'locationId' });
    }
  }
  Location.init({
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: DataTypes.INTEGER,
    },
    telegramId: {
      allowNull: false,
      type: DataTypes.INTEGER,
      unique: 'telegramIdAndLocalnameUnique',
      references: {
        model: 'Users',
        key: 'telegramId',
      },
    },
    coordinates: {
      allowNull: false,
      type: DataTypes.GEOGRAPHY,
    },
    localName: {
      allowNull: false,
      type: DataTypes.STRING,
      unique: 'telegramIdAndLocalnameUnique',
    },
    globalName: {
      allowNull: false,
      type: DataTypes.STRING,
    },
  }, {
    sequelize,
    modelName: LOCATION_MODEL_NAME,
  });
  return Location;
};
```

assignment.js

```
const {
  Model,
} = require('sequelize');

const { ASSIGNMENT_MODEL_NAME } =
require('../../constants').enums.databaseModel;

module.exports = (sequelize, DataTypes) => {
  class Assignment extends Model {
    static associate(models) {
      Assignment.belongsTo(models.Location, { foreignKey: 'locationId', onDelete:
'cascade' });
      Assignment.belongsTo(models.User, { foreignKey: 'authorTelegramId', as:
'author' });

      Assignment.belongsToMany(models.User, { through: 'FavoriteAssignments',
foreignKey: 'assignmentId', as: 'favoriteUsers' });

      Assignment.belongsToMany(models.User, { through: models.Spam,
foreignKey: 'assignmentId', as: 'spamUsers' });
    }
  }
  Assignment.init({
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: DataTypes.INTEGER,
    },
    authorTelegramId: {
      allowNull: false,
      type: DataTypes.INTEGER,
      references: {
        model: 'Users',
        key: 'telegramId',
      },
    },
    locationId: {
      allowNull: false,
      type: DataTypes.INTEGER,
      references: {
        model: 'Locations',
```

```
      key: 'id',
    },
  },
  title: {
    allowNull: false,
    type: DataTypes.STRING,
  },
  description: {
    allowNull: false,
    type: DataTypes.STRING,
  },
  reward: {
    allowNull: true,
    type: DataTypes.STRING,
  },
  link: {
    allowNull: true,
    type: DataTypes.STRING,
  },
  status: {
    allowNull: false,
    type: DataTypes.ENUM,
    values: ['notDone', 'done'],
    defaultValue: 'notDone',
  },
  pictureUrl: {
    allowNull: true,
    type: DataTypes.STRING,
  },
  spamScore: {
    allowNull: false,
    defaultValue: 0,
    type: DataTypes.INTEGER,
  },
  category: {
    allowNull: false,
    type: DataTypes.ENUM,
    values: ['help', 'barter', 'repair', 'education', 'other'],
    defaultValue: 'other',
  },
}, {
  sequelize,
  modelName: ASSIGNMENT_MODEL_NAME,
});
```

```
  return Assignment;
};
```

favoriteassignment.js

```
const {
  Model,
} = require('sequelize');

const { FAVORITE_ASSIGNMENT_MODEL_NAME } =
require('../../constants').enums.databaseModel;

module.exports = (sequelize, DataTypes) => {
  class FavoriteAssignment extends Model {
    static associate(models) {
      FavoriteAssignment.belongsTo(models.User, { foreignKey: 'telegramId' });
      FavoriteAssignment.belongsTo(models.Assignment, { foreignKey:
'assignmentId', onDelete: 'CASCADE' });
    }
  }
  FavoriteAssignment.init({
    telegramId: {
      allowNull: false,
      primaryKey: true,
      type: DataTypes.INTEGER,
      references: {
        model: 'Users',
        key: 'telegramId',
      },
    },
    assignmentId: {
      allowNull: false,
      primaryKey: true,
      type: DataTypes.INTEGER,
      references: {
        model: 'Assignments',
        key: 'id',
      },
    },
  }, {
    sequelize,
    modelName: FAVORITE_ASSIGNMENT_MODEL_NAME,
  });
  return FavoriteAssignment;
```

```
};
```

spam.js

```
const {
  Model,
} = require('sequelize');

const { SPAM_MODEL_NAME } =
require('../../constants').enums.databaseModel;

module.exports = (sequelize, DataTypes) => {
  class Spam extends Model {
    static associate(models) {
      Spam.belongsTo(models.User, { foreignKey: 'telegramId' });
      Spam.belongsTo(models.Assignment, { foreignKey: 'assignmentId' });
    }
  }
  Spam.init({
    id: {
      allowNull: false,
      autoIncrement: true,
      primaryKey: true,
      type: DataTypes.INTEGER,
    },
    telegramId: {
      allowNull: false,
      primaryKey: true,
      type: DataTypes.INTEGER,
      references: {
        model: 'Users',
        key: 'telegramId',
      },
    },
    assignmentId: {
      allowNull: false,
      primaryKey: true,
      type: DataTypes.INTEGER,
      references: {
        model: 'Assignments',
        key: 'id',
      },
    },
```

```
  }, {
    sequelize,
    modelName: SPAM_MODEL_NAME,
  });
  return Spam;
};
```

actions.js

```
exports.ASSIGNMENT_ACTIONS = {
  ADD_TO_FAVORITES: 'addToFavoritesAction',
  REMOVE_FROM_FAVORITES: 'removeFromFavoritesAction',

  MARK_ASSIGNMENT_AS_NOT_COMPLETED:
'markAssignmentAsNotCompletedAction',
  MARK_ASSIGNMENT_AS_COMPLETED:
'markAssignmentAsCompletedAction',

  MARK_ASSIGNMENT_AS_SPAM: 'markAssignmentAsSpamAction',
  CONFIRM_ASSIGNMENT_AS_SPAM: 'confirmAssignmentAsSpamAction',
  BACK_FROM_CONFIRM_ASSIGNMENT_AS_SPAM:
'backFromConfirmAssignmentAsSpamAction',

  REMOVE_ASSIGNMENT: 'removeAssignmentAction',
  CONFIRM_ASSIGNMENT_REMOVE: 'confirmAssignmentRemoveAction',
  BACK_FROM_CONFIRM_ASSIGNMENT_REMOVE:
'backFromConfirmAssignmentRemoveAction',

  EDIT_ASSIGNMENT: 'editAssignmentAction',
};

exports.RANGE_ACTIONS = {
  CHANGE_RANGE: 'changeRangeAction',
};

exports.COMMON_ACTIONS = {
  PAGINATION: 'paginationAction',
};
```

button.text.js

```
exports.MAIN_MENU_BUTTONS = {
  ABOUT_US: 'О нас',
  CHANGE_RANGE: 'Редактировать радиус',
  MY_ASSIGNMENT: 'Мои объявления',
  MY_LOCATIONS: 'Мои локации',
  FIND_ASSIGNMENTS: 'Найти объявления',
  ADD_ASSIGNMENT: 'Подать объявление',
};

exports.LOCATION_BUTTONS = {
  ADD_LOCATION: 'Добавить локацию',
  DELETE_LOCATION: 'Удалить локацию',
};

exports.COMMON_BUTTONS = {
  BACK: 'Назад',
  HOME: 'Домой',
  CONFIRM: 'Да',
  SKIP: 'Пропустить',
  EDIT_ASSIGNMENT: 'Редактировать объявление',
  DELETE: 'Удалить',
  ADD_LOCATION: 'Добавить локацию',
};

exports.CATEGORY_BUTTONS = {
  BARTER: 'Обмен',
  EDUCATION: 'Обучение',
  HELP: 'Помощь',
  REPAIR: 'Ремонт',
  OTHER: 'Другое',
};

exports.MY_ASSIGNMENTS_MENU_BUTTONS = {
  FAVORITE_ASSIGNMENTS: 'Избранные',
  CREATED_ASSIGNMENTS: 'Поданные объявления',
};

exports.ADD_LOCATIONS_MENU_BUTTONS = {
  ADD_CURRENT_LOCATION: 'Текущая локация',
};

exports.ADD_ASSIGNMENT_BUTTONS = {
```

```
  PUBLISH_ASSIGNMENT: 'Опубликовать объявление',
};
```

flow.step.js

```
exports.ADD_LOCATION_FLOW_STEPS = {
  ADD_LOCATION: 'addLocation.addLocation',
  ADD_LOCATION_NAME: 'addLocation.addLocationName',
};

exports.LOCATION_FLOW_STEPS = {
  CHOOSE_ACTION: 'myLocations.chooseAction',
  ADD_LOCATION: 'myLocations.addLocation',
  DELETE_LOCATION: 'myLocations.deleteLocation',
  SHOW_LOCATIONS: 'myLocations.showLocation',
};

exports.FIND_ASSIGNMENTS_FLOW_STEPS = {
  CHOOSE_CATEGORY: 'findAssignment.chooseCategory',
  CHOOSE_LOCATION: 'findAssignment.chooseLocation',
  GET_ASSIGNMENTS: 'findAssignment.getAssignments',
};

exports.MY_ASSIGNMENTS_FLOW_STEPS = {
  GET_CREATED_ASSIGNMENTS: 'myAssignment.getCreatedAssignments',
  GET_FAVORITE_ASSIGNMENTS: 'myAssignment.getFavoriteAssignments',
};

exports.CHANGE_RANGE_FLOW_STEPS = {
  CHANGE_RANGE: 'changeRange.ChangeRange',
};

exports.ADD_ASSIGNMENT_FLOW_STEPS = {
  CHOOSE_CATEGORY: 'addAssignment.chooseCategory',
  ADD_TITLE: 'addAssignment.addTitle',
  ADD_DESCRIPTION: 'addAssignment.addDescription',
  CHOOSE_LOCATION: 'addAssignment.chooseLocation',
  ADD_REWARD: 'addAssignment.addReward',
  SHOW_ASSIGNMENT: 'addAssignment.showAssignment',
  PUBLISH_ASSIGNMENT: 'addAssignment.publishAssignment',
};

exports.EDIT_ASSIGNMENT_FLOW_STEPS = {
```

```
  CHOOSE_CATEGORY: 'editAssignment.chooseCategory',
  EDIT_TITLE: 'editAssignment.addTitle',
  EDIT_DESCRIPTION: 'editAssignment.addDescription',
  CHOOSE_LOCATION: 'editAssignment.chooseLocation',
  EDIT_REWARD: 'editAssignment.addReward',
  SHOW_ASSIGNMENT: 'editAssignment.showAssignment',
  PUBLISH_ASSIGNMENT: 'editAssignment.publishAssignment',
};
```

user.service.js

```
const {
  models: {
    User,
  },
} = require('../database');

const ServiceResponse = require('../helpers/ServiceResponse');

module.exports = {
  async create({ telegramId }) {
    try {
      const [, created] = await User.findOrCreate({
        where: {
          telegramId,
        },
        defaults: {
          telegramId,
        },
      });

      if (created) { return new ServiceResponse({ succeeded: true }); }

      return new ServiceResponse({
        succeeded: true,
        message: `User with such telegramId=${telegramId} already exists.`,
      });
    } catch (e) {
      return new ServiceResponse({
        succeeded: false,
        message: `Error occurred while creating user with
telegramId=${telegramId}.`
          + `${e}.`,
      });
```

```
    }
  },

  async update({
    telegramId = null,
    newRange = null,
    newLocale = null,
    updatedState = null,
  }) {
    try {
      const foundUser = await User.findOne({
        where: {
          telegramId,
        },
        attributes: ['telegramId', 'range', 'locale', 'state'],
      });

      if (foundUser) {
        await foundUser.update({
          range: newRange || foundUser.range,
          locale: newLocale || foundUser.locale,
          state: updatedState ? { ...foundUser.state, ...updatedState } : foundUser.state,
        });
        return new ServiceResponse({ succeeded: true });
      }
      return new ServiceResponse({
        succeeded: true,
        message: `User with such telegramId=${telegramId} was not found.`,
      });
    } catch {
      return new ServiceResponse({
        succeeded: false,
        message: `Error occurred while updating user with
telegramId=${telegramId}`
          + `with values newRange=${newRange} and newLocale=${newLocale}`,
      });
    }
  },

  async getOne({ telegramId, params }) { // params: ['locale', 'range', 'state'] -
example
    try {
      const foundUser = await User.findOne({
        where: {
```

```
        telegramId,
      },
      attributes: params,
    });

    if (foundUser) {
      return new ServiceResponse({
        succeeded: true,
        model: foundUser.dataValues,
      });
    }
    return new ServiceResponse({
      succeeded: true,
      message: `User with such telegramId=${telegramId} was not found`,
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: `Error occurred while getting user with telegramId=${telegramId}`
        + `${e}.`,
    });
  }
 },
};
```

location.service.js

```
const {
 models: {
  Location,
  User,
 },
} = require('../database');

const ServiceResponse = require('../helpers/ServiceResponse');

module.exports = {
 async create({
  telegramId, coordinates, localName, globalName, // coordinates: [longitude,
latitude]
 }) {
  try {
   const foundUser = await User.findOne({
    where: {
```

```
        telegramId,
      },
      attributes: ['telegramId'],
    });

    if (foundUser) {
      const foundLocation = await Location.findOne({
        where: {
          telegramId,
          localName,
        },
        attributes: ['id'],
      });

      if (!foundLocation) {
        await Location.create({
          telegramId,
          coordinates: { type: 'Point', coordinates },
          localName,
          globalName,
        });

        return new ServiceResponse({ succeeded: true });
      }

      return new ServiceResponse({
        succeeded: true,
        message: `Location name ${localName} is already taken!`,
      });
    }
    return new ServiceResponse({
      succeeded: true,
      message: `User with such telegramId=${telegramId} was not found.`,
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: `Error occurred while creating location with
telegramId=${telegramId}
          + coordinates=${coordinates}, localName=${localName},
globalName=${globalName}.
          + ${e}.`,
    });
  }
```

```
  },

  async delete({ telegramId, localName }) {
    try {
      const foundLocation = await Location.findOne({
        where: {
          localName,
          telegramId,
        },
      });

      if (foundLocation) {
        await foundLocation.destroy();
        return new ServiceResponse({ succeeded: true });
      }

      return new ServiceResponse({
        succeeded: true,
        message: `Location with localName=${localName} wasn't`
          + `found on user with telegramId=${telegramId}.`,
      });
    } catch (e) {
      return new ServiceResponse({
        succeeded: false,
        message: `Error occurred while deleting location with
telegramId=${telegramId}
          + localName=${localName}.
          + ${e}.`,
      });
    }
  },

  async getAllByTelegramId({ telegramId }) {
    try {
      const foundLocations = await User.findOne({
        where: {
          telegramId,
        },
        attributes: [],
        include: [{
          model: Location,
          attributes: ['localName', 'id'],
        }],
      });
```

```
    if (foundLocations) {
      return new ServiceResponse({
        succeeded: true,
        model: foundLocations.Locations,
      });
    }

    return new ServiceResponse({
      succeeded: true,
      message: `User with such telegramId=${telegramId} was not found.`,
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: `Error occurred while getting user's locations with
telegramId=${telegramId}. `
        + `${e}.`,
    });
  }
},

// eslint-disable-next-line consistent-return
async updateLocation({ id, coordinates, ...data }) {
  try {
    const result = await Location.update(
      {
        ...data,
        coordinates: { type: 'Point', coordinates },
      },
      {
        where: { id },
        returning: true,
      },
    );
    if (result[0]) return new ServiceResponse({ succeeded: true });
  } catch {
    return new ServiceResponse({
      succeeded: false,
      message: `Error occurred while updating location with id=${id}`,
    });
  }
},
```

```
};
```

assignment.service.js

```
const {
  models: {
    Assignment,
    User,
    Location,
    FavoriteAssignment,
    Spam,
    sequelize,
  },
} = require('../database');

const { getPagingData } = require('../helpers/pagination');
const { REQUIRED_SPAM_SCORE_TO_DELETE } =
require('../configs/global.config');

const ServiceResponse = require('../helpers/ServiceResponse');

module.exports = {
  async create(
    {
      title,
      description,
      reward = null,
      link = null,
      pictureUrl = null,
      category,
      authorTelegramId,
      localLocationName,
    },
  ) {
    try {
      const result = await User.findOne({
        where: {
          telegramId: authorTelegramId,
        },
        include: [{
          model: Location,
          attributes: ['id'],
          where: {
            localName: localLocationName,
          },
```

```
      }],
      attributes: ['telegramId'],
    });

    if (result) {
      if (result.Locations.length) {
        await Assignment.create({
          title,
          description,
          reward,
          link,
          pictureUrl,
          category,
          authorTelegramId: result.telegramId,
          locationId: result.Locations[0].id,
        });
      }
      return new ServiceResponse({ succeeded: true });
    }

    return new ServiceResponse({
      succeeded: true,
      message: `No locations were found using user's
telegramId=${authorTelegramId} `
        + `and location's localName=${localLocationName}`,
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while creating assignment with '
        + `title=${title}, telegramId=${authorTelegramId},
description=${description}, `
        + `reward=${reward}, link=${link}, pictureUrl=${pictureUrl}, `
        + `category=${category},  localLocationName=${localLocationName}. `
        + `${e}`,
    });
  }
},

async getAllNearby({
  telegramId,
  category = null,
  locationId,
  pagination: { limit, offset },
```

```
  page,
 }) {
  try {
    const foundRangeAndCoordinates = await User.findOne({
      where: {
        telegramId,
      },
      attributes: ['range'],
      include: [{
        model: Location,
        where: {
          id: locationId,
        },
        attributes: ['coordinates'],
      }],
    });

    const categoryCondition = category ? `AND A.category = '${category}'` : '';

    const nearbyAssignmentsQuery = `SELECT A.id, A.title, A.description,
A.reward,
    A."pictureUrl", L."globalName", A."authorTelegramId",
    EXISTS (
    SELECT 1 FROM "FavoriteAssignments" FA
    WHERE FA."assignmentId" = A.id
    AND FA."telegramId" = ${telegramId}) AS "isFavorite"
    FROM "Assignments" A
    INNER JOIN "Locations" L ON A."locationId" = L.id
    LEFT JOIN "Spams" S ON A.id = S."assignmentId"
    AND S."telegramId" = ${telegramId}
    WHERE ST_DWithin(L.coordinates,

ST_MakePoint(${foundRangeAndCoordinates.Locations[0].coordinates.coordinat
es[0]},
    ${foundRangeAndCoordinates.Locations[0].coordinates.coordinates[1]}),
    ${foundRangeAndCoordinates.range} * 1000)
    ${categoryCondition}
    AND A."authorTelegramId" <> ${telegramId}
    AND A.status <> 'done'
    AND S."assignmentId" IS NULL
    ORDER BY L.coordinates <-> L.coordinates
    LIMIT ${limit}
    OFFSET ${offset}`;
```

```
    const nearbyAssignmentsCountQuery = `SELECT COUNT(A.id)
    FROM "Assignments" A
    INNER JOIN "Locations" L ON A."locationId" = L.id
    LEFT JOIN "Spams" S ON A.id = S."assignmentId"
    AND S."telegramId" = ${telegramId}
    WHERE ST_DWithin(L.coordinates,

ST_MakePoint(${foundRangeAndCoordinates.Locations[0].coordinates.coordinat
es[0]},
    ${foundRangeAndCoordinates.Locations[0].coordinates.coordinates[1]}),
    ${foundRangeAndCoordinates.range} * 1000)
    ${categoryCondition}
    AND A."authorTelegramId" <> ${telegramId}
    AND A.status <> 'done'
    AND S."assignmentId" IS NULL`;

    const [[nearbyAssignments], [[nearbyAssignmentsCount]]] = await
Promise.all([
      sequelize.query(nearbyAssignmentsQuery),
      sequelize.query(nearbyAssignmentsCountQuery),
    ]);

    return new ServiceResponse({
      succeeded: true,
      pagingData: getPagingData(nearbyAssignmentsCount, page, limit),
      model: nearbyAssignments,
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while getting all nearby assignments with '
        + `telegramId=${telegramId}, category=${category}. `
        + `${e}`,
    });
  }
},

async getAllFavorites({
  telegramId,
  pagination: { limit, offset },
  page,
}) {
  try {
```

```
    const queryRecords = 'SELECT A."id", A."title", A."description", A."reward",
A."authorTelegramId",'
      + `A."pictureUrl", L."globalName"
    FROM "Assignments" A
    INNER JOIN "FavoriteAssignments" FA ON A.id = FA."assignmentId"
    AND FA."telegramId" = ${telegramId}
    LEFT JOIN "Spams" S ON A.id = S."assignmentId"
    AND S."telegramId" = ${telegramId}
    INNER JOIN "Locations" L on A."locationId" = L.id
    WHERE S."telegramId" IS NULL
    AND A.status <> 'done'
    LIMIT ${limit}
    OFFSET ${offset}`;

    const queryCount = `SELECT COUNT(A.id) as count
    FROM "Assignments" A
    INNER JOIN "FavoriteAssignments" FA ON A.id = FA."assignmentId"
    AND FA."telegramId" = ${telegramId}
    LEFT JOIN "Spams" S ON A.id = S."assignmentId"
    AND S."telegramId" = ${telegramId}
    WHERE S."telegramId" IS NULL
    AND A.status <> 'done'`;

    const [[favoriteAssignments], [[countResult]]] = await Promise.all([
      sequelize.query(queryRecords),
      sequelize.query(queryCount),
    ]);

    favoriteAssignments.count = countResult.count;

    return new ServiceResponse({
      succeeded: true,
      pagingData: getPagingData(favoriteAssignments, page, limit),
      model: favoriteAssignments.map((elem) => ({
        id: elem.id,
        title: elem.title,
        description: elem.description,
        reward: elem.reward,
        authorTelegramId: elem.authorTelegramId,
        authorUsername: elem.authorUsername,
        pictureUrl: elem.pictureUrl,
        locationName: elem.globalName,
      })),
    });
```

```
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while getting all favorite assignments with '
        + `telegramId=${telegramId}. `
        + `${e}`,
    });
  }
},

async getCreated({
  telegramId,
  pagination: { limit, offset },
  page,
}) {
  try {
    const createdAssignments = await Assignment.findAndCountAll({
      where: {
        authorTelegramId: telegramId,
      },
      order: [
        ['id', 'ASC'],
      ],
      include:
        [
          {
            model: Location,
          },
          {
            model: User,
            as: 'author',
            attributes: [],
          },
        ],
      limit,
      offset,
    });

    return new ServiceResponse({
      succeeded: true,
      pagingData: getPagingData(createdAssignments, page, limit),
      model: createdAssignments.rows.map((elem) => ({
        id: elem.dataValues.id,
        title: elem.dataValues.title,
```

```
        description: elem.dataValues.description,
        status: elem.dataValues.status,
        reward: elem.dataValues.reward,
        authorTelegramId: elem.dataValues.authorTelegramId,
        pictureUrl: elem.dataValues.pictureUrl,
        locationName: elem.dataValues.Location.dataValues.globalName,
        localLocationName: elem.dataValues.Location.dataValues.localName,
      })),
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while getting created assignments with '
        + `telegramId=${telegramId}. `
        + `${e}`,
    });
  }
},

async delete({ telegramId, assignmentId }) {
  try {
    const foundAssignment = await Assignment.findOne({
      where: {
        id: assignmentId,
        authorTelegramId: telegramId,
      },
      attributes: ['id'],
    });

    if (foundAssignment) {
      await foundAssignment.destroy();
      return new ServiceResponse({ succeeded: true });
    }

    return new ServiceResponse({
      succeeded: true,
      message: `Assigment with id=${assignmentId} wasn't`
        + `found on user with telegramId=${telegramId}.`,
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while deleting assignment with '
        + `telegramId=${telegramId}, assignmentId=${assignmentId}. `
```

```
      + `${e}.`,
    });
  }
},

async update({ telegramId, assignmentId, status = null }) {
  try {
    const foundAssignment = await Assignment.findOne({
      where: {
        id: assignmentId,
        authorTelegramId: telegramId,
      },
      attributes: ['id'],
    });

    if (foundAssignment) {
      await foundAssignment.update({
        status: status || foundAssignment.status,
      });
      return new ServiceResponse({ succeeded: true });
    }

    return new ServiceResponse({
      succeeded: true,
      message: `Assigment with id=${assignmentId} wasn't`
        + `found on user with telegramId=${telegramId}.`,
    });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while updating assignment with '
        + `authorTelegramId=${telegramId}, id=${assignmentId}, status=${status}.
`
        + `${e}.`,
    });
  }
},

async addToFavorites({ telegramId, assignmentId }) {
  try {
    await FavoriteAssignment.findOrCreate({
      where: {
        telegramId,
        assignmentId,
```

```
      },
      defaults: {
        telegramId,
        assignmentId,
      },
    });

    return new ServiceResponse({ succeeded: true });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while adding assignment to favorites with '
        + `telegramId=${telegramId}, id=${assignmentId}. `
        + `${e}.`,
    });
  }
},

async removeFromFavorites({ telegramId, assignmentId }) {
  try {
    await FavoriteAssignment.destroy({
      where: {
        telegramId,
        assignmentId,
      },
    });

    return new ServiceResponse({ succeeded: true });
  } catch (e) {
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while adding assignment to favorites with '
        + `telegramId=${telegramId}, id=${assignmentId}. `
        + `${e}.`,
    });
  }
},

async markAsSpam({ telegramId, assignmentId }) {
  const t = await sequelize.transaction();

  try {
    const [[assignment]] = await sequelize.query(`UPDATE "Assignments" A
    SET "spamScore" = "spamScore" + 1, "updatedAt" = NOW()
```

```
    WHERE "id" = ${assignmentId} RETURNING A."spamScore"`);

    if (assignment.spamScore >= REQUIRED_SPAM_SCORE_TO_DELETE) {
      await Promise.all([
        Assignment.destroy({
          where: {
            id: assignmentId,
          },
        },
        { transaction: t }),

        Spam.destroy(
          {
            where: {
              assignmentId,
            },
          },
          { transaction: t },
        )]);
    } else {
      await Spam.findOrCreate({
        where: {
          telegramId,
          assignmentId,
        },
        defaults: {
          telegramId,
          assignmentId,
        },
        attributes: ['id'],
      });
    }

    await t.commit();
    return new ServiceResponse({ succeeded: true });
  } catch (e) {
    await t.rollback();
    return new ServiceResponse({
      succeeded: false,
      message: 'Error occurred while marking assignment as spam with '
        + `telegramId=${telegramId}, assignmentId=${assignmentId}. `
        + `${e}.`,
    });
  }
```

```
  },

  async get({ telegramId, assignmentId }) {
    try {
      const query = `SELECT A.id, A.title, A.reward, A.description,
      A."pictureUrl", L."globalName" as "locationName",
      A."authorTelegramId", A.status, EXISTS (
      SELECT 1 FROM "FavoriteAssignments" FA
      WHERE FA."assignmentId" = ${assignmentId}
      AND FA."telegramId" = ${telegramId}) AS "isFavorite"
      FROM "Assignments" A
      INNER JOIN "Locations" L
      ON A."locationId" = L.id
      WHERE A.id = ${assignmentId}
      LIMIT 1`;

      const [[foundAssignment]] = await sequelize.query(query);

      if (foundAssignment) {
        return new ServiceResponse({ succeeded: true, model: foundAssignment });
      }
      return new ServiceResponse({ succeeded: true });
    } catch (e) {
      return new ServiceResponse({
        succeeded: false,
        message: 'Error occurred while getting assignment by id with '
          + `assignmentId=${assignmentId}. `
          + `${e}.`,
      });
    }
  },

  async assignmentGetById(id) {
    return Assignment.findOne({
      where: { id },
      attributes: ['id', 'authorTelegramId', 'title', 'description', 'reward', 'category',
'pictureUrl'],
      include: [{
        model: Location,
        attributes: ['localName'],
      }],
    });
  },
```

```
  // eslint-disable-next-line consistent-return
  async assignmentUpdateById(id, data) {
    const result = await Assignment.update(data, {
      where: { id },
      returning: true,
      plain: true,
    });

    if (result[1]) return new ServiceResponse({ succeeded: true });
    return new ServiceResponse({
      succeeded: false,
      message: `Error occurred while updatinng assignment with
assignmentId=${id}.`,
    });
  },
};
```

features/index.js

```
/* eslint-disable consistent-return */
const locationHandler = require('./locationHandler');
const textHandler = require('./textHandler');
const callbackQueryHandler = require('./callbackQueryHandler');
const { messageDefaultAction } = require('../actions/commonActions');
const pictureHandler = require('./pictureHandler');
const { stateLoad } = require('../helpers/state');

const { ADD_LOCATION_FLOW_STEPS } = require('../constants/flow.step');

const handlers = async ({ originalRequest }) => {
  try {
    let state = {
      data: '',
      step: '',
      cache: '',
    };
    if (originalRequest.message) {
      const { message } = originalRequest;
      // eslint-disable-next-line no-use-before-define
      state = await stateLoad(message);
      if (message.location && state.step ===
ADD_LOCATION_FLOW_STEPS.ADD_LOCATION) {
        return await locationHandler(message, state);
```

```
      }

      if (message.photo) {
        return await pictureHandler(message, state);
      }

      if (message.text) {
        return await textHandler(message, state);
      }
    }

    if (originalRequest.callback_query) {
      // eslint-disable-next-line no-use-before-define
      state = await stateLoad(originalRequest.callback_query);
      return await callbackQueryHandler(originalRequest.callback_query, state);
    }

    return messageDefaultAction();
  } catch (err) {
    console.error(err.name);
    console.error(err.message);
    console.error(err.stack);
    return messageDefaultAction();
  }
};

module.exports = handlers;
```

textHandler.js

```
const {
  MAIN_MENU_BUTTONS,
  MY_ASSIGNMENTS_MENU_BUTTONS,
  COMMON_BUTTONS,
  CATEGORY_BUTTONS,
  ADD_ASSIGNMENT_BUTTONS,
  LOCATION_BUTTONS,
} = require('../constants/button.text');

const {
  buttonBackHandler,
  categoryHandler,
} = require('./buttonHandlers');
```

```
const {
  stateDefaultHandler,
} = require('./stateDefaultHandler');

const {
  startAction,
  mainMenuAction,
  aboutUsAction,
  showRangeAction,
  myAssignmentAction,
  findAssignmentsAction,
  addMenuSelectCategoryForCreatedAssignmentAction,
  myLocationsAction,
} = require('../actions/mainActions');

const { addMenuAddLocationAction } = require('../actions/commonActions');

const {
  createdAssignmentsAction,
  favoriteAssignmentsAction,
} = require('../actions/assignmentActions');

const {
  buttonEditAssignmentHandler,
  buttonPublishAssignmentHandler,
  myLocationsHandler,
} = require('./buttonHandlers');

const textHandlers = async (request, state) => {
  switch (request.text) {
    case '/start':
      return startAction(request);

    case MAIN_MENU_BUTTONS.ABOUT_US:
      return aboutUsAction(request);

    case MAIN_MENU_BUTTONS.CHANGE_RANGE:
      return showRangeAction(request);

    case MAIN_MENU_BUTTONS.MY_ASSIGNMENT:
      return myAssignmentAction();

    case MAIN_MENU_BUTTONS.FIND_ASSIGNMENTS:
      return findAssignmentsAction(request, state);
```

```
    case MAIN_MENU_BUTTONS.MY_LOCATIONS:
      return myLocationsAction(request);

    case
MY_ASSIGNMENTS_MENU_BUTTONS.FAVORITE_ASSIGNMENTS:
      return favoriteAssignmentsAction(request);

    case MY_ASSIGNMENTS_MENU_BUTTONS.CREATED_ASSIGNMENTS:
      return createdAssignmentsAction(request);

    case COMMON_BUTTONS.HOME:
      return mainMenuAction(request);

    case COMMON_BUTTONS.ADD_LOCATION:
      return addMenuAddLocationAction(request, state);

    case COMMON_BUTTONS.BACK:
      return buttonBackHandler(request, state);

    // Выбор категории
    // TODO: заменить на константы, когда будет локализация
    case CATEGORY_BUTTONS.HELP:
      request.text = 'help';
      return categoryHandler(request, state);

    case CATEGORY_BUTTONS.BARTER:
      request.text = 'barter';
      return categoryHandler(request, state);

    case CATEGORY_BUTTONS.REPAIR:
      request.text = 'repair';
      return categoryHandler(request, state);

    case CATEGORY_BUTTONS.EDUCATION:
      request.text = 'education';
      return categoryHandler(request, state);

    case CATEGORY_BUTTONS.OTHER:
      request.text = 'other';
      return categoryHandler(request, state);

    case MAIN_MENU_BUTTONS.ADD_ASSIGNMENT:
      return addMenuSelectCategoryForCreatedAssignmentAction(request, state);
```

```
    case ADD_ASSIGNMENT_BUTTONS.PUBLISH_ASSIGNMENT:
      return buttonPublishAssignmentHandler(request, state);

    case COMMON_BUTTONS.EDIT_ASSIGNMENT:
      return buttonEditAssignmentHandler(request, state);

    case LOCATION_BUTTONS.ADD_LOCATION:
    case LOCATION_BUTTONS.DELETE_LOCATION:
      return myLocationsHandler(request, state);

    default:
      // eslint-disable-next-line no-case-declarations
      const response = await stateDefaultHandler(request, state);
      if (response) return response;
      return mainMenuAction(request);
  }
};

module.exports = textHandlers;
```

pictureHandler.js

```
const {
  ADD_ASSIGNMENT_FLOW_STEPS,
  EDIT_ASSIGNMENT_FLOW_STEPS,
} = require('../constants/flow.step');

const { addPictureForAddAssignmentAction } =
require('../actions/addAssignmentAction');
const { editPictureForEditAssignmentAction } =
require('../actions/editAssignmentAction');

const pictureHandler = (message, state) => {
  const { step } = state;

  switch (step) {
    case ADD_ASSIGNMENT_FLOW_STEPS.SHOW_ASSIGNMENT:
      return addPictureForAddAssignmentAction(message, state);

    case EDIT_ASSIGNMENT_FLOW_STEPS.SHOW_ASSIGNMENT:
      return editPictureForEditAssignmentAction(message, state);

    default:
```

```
    throw Error('Unplanned image.');
  }
};

module.exports = pictureHandler;
```

stateDefaultHandler.js

```
const {
  addLocalNameLocationAction,
} = require('../actions/locationAction');

const {
  addFoundAssignmentLocationAction,
} = require('../actions/assignmentActions');

const {
  changeRangeAction,
} = require('../actions/mainActions');

const {
  chooseCategoryAssignmentAction,
  addTitleForAddAssignmentAction,
  addDescriptionForAddAssignmentAction,
  chooseLocationForAddAssignmentAction,
  addRewardForAddAssignmentAction,
  addPictureForAddAssignmentAction,
} = require('../actions/addAssignmentAction');

const {
  chooseCategoryForEditAssignmentAction,
  editTitleForEditAssignmentAction,
  editDescriptionForEditAssignmentAction,
  chooseLocationForEditAssignmentAction,
  editRewardForEditAssignmentAction,
  editPictureForEditAssignmentAction,
} = require('../actions/editAssignmentAction');

const {
  myLocationDeleteAction,
} = require('../actions/myLocationAction');

const {
  ADD_ASSIGNMENT_FLOW_STEPS,
```

```
  EDIT_ASSIGNMENT_FLOW_STEPS,
  FIND_ASSIGNMENTS_FLOW_STEPS,
  ADD_LOCATION_FLOW_STEPS,
  CHANGE_RANGE_FLOW_STEPS,
  LOCATION_FLOW_STEPS,
} = require('../constants/flow.step');

// eslint-disable-next-line consistent-return
const stateDefaultHandler = (request, state) => {
  // eslint-disable-next-line default-case
  switch (state.step) {
    case ADD_LOCATION_FLOW_STEPS.ADD_LOCATION_NAME:
      return addLocalNameLocationAction(request, state);

    case FIND_ASSIGNMENTS_FLOW_STEPS.CHOOSE_LOCATION:
      return addFoundAssignmentLocationAction({ request, state });

    case CHANGE_RANGE_FLOW_STEPS.CHANGE_RANGE:
      return changeRangeAction(request);

    // Add assignment
    case ADD_ASSIGNMENT_FLOW_STEPS.CHOOSE_CATEGORY:
      return chooseCategoryAssignmentAction(request, state, false);

    case ADD_ASSIGNMENT_FLOW_STEPS.ADD_TITLE:
      return addTitleForAddAssignmentAction(request, state);

    case ADD_ASSIGNMENT_FLOW_STEPS.ADD_DESCRIPTION:
      return addDescriptionForAddAssignmentAction(request, state);

    case ADD_ASSIGNMENT_FLOW_STEPS.CHOOSE_LOCATION:
      return chooseLocationForAddAssignmentAction(request, state);

    case ADD_ASSIGNMENT_FLOW_STEPS.ADD_REWARD:
      return addRewardForAddAssignmentAction(request, state);

    case ADD_ASSIGNMENT_FLOW_STEPS.SHOW_ASSIGNMENT:
      return addPictureForAddAssignmentAction(request, state);

    // Edit assignment
    case EDIT_ASSIGNMENT_FLOW_STEPS.CHOOSE_CATEGORY:
      return chooseCategoryForEditAssignmentAction(request, state, false);

    case EDIT_ASSIGNMENT_FLOW_STEPS.EDIT_TITLE:
```

```
      return editTitleForEditAssignmentAction(request, state);

    case EDIT_ASSIGNMENT_FLOW_STEPS.EDIT_DESCRIPTION:
      return editDescriptionForEditAssignmentAction(request, state);

    case EDIT_ASSIGNMENT_FLOW_STEPS.CHOOSE_LOCATION:
      return chooseLocationForEditAssignmentAction(request, state);

    case EDIT_ASSIGNMENT_FLOW_STEPS.EDIT_REWARD:
      return editRewardForEditAssignmentAction(request, state);

    case EDIT_ASSIGNMENT_FLOW_STEPS.SHOW_ASSIGNMENT:
      return editPictureForEditAssignmentAction(request, state);

    case LOCATION_FLOW_STEPS.SHOW_LOCATIONS:
      return myLocationDeleteAction(request, state);
  }
};

exports.stateDefaultHandler = stateDefaultHandler;
```

callbackQueryHandler.js

```
/* eslint-disable no-return-await */
const {
  changeRangeAction,
  mainMenuAction,
} = require('../actions/mainActions');

const { paginationAction } = require('../actions/commonActions');
const {
  removeFromFavoritesAction,
  addToFavoritesAction,
  markAssignmentAsSpamAction,
  confirmAssignmentAsSpamAction,
  backFromConfirmAssignmentAsSpamAction,
  removeAssignmentAction,
  markAssignmentAsNotCompletedAction,
  markAssignmentAsCompletedAction,
  backFromConfirmAssignmentRemoveAction,
  confirmAssignmentRemoveAction,
} = require('../actions/assignmentActions');
```

```
const { addMenuSelectCategoryForEditAssignmentAction } =
require('../actions/editAssignmentAction');

const {
  ASSIGNMENT_ACTIONS,
  COMMON_ACTIONS,
  RANGE_ACTIONS,
} = require('../constants/actions');

const { deleteMessage } = require('../helpers/telegram');

const callbackQueryHandler = async (callbackQuery, state) => {
  let response;
  const [action, index, fromFavorites] = callbackQuery.data.split('.');
  switch (action) {
    case RANGE_ACTIONS.CHANGE_RANGE:
      return changeRangeAction(callbackQuery);

    case COMMON_ACTIONS.PAGINATION:
      response = await paginationAction(callbackQuery, +index, state);
      return response.concat(deleteMessage(callbackQuery));

    case ASSIGNMENT_ACTIONS.REMOVE_FROM_FAVORITES:
      return removeFromFavoritesAction(
        {
          request: callbackQuery,
          assignmentId: index,
          fromFavorites,
        },
      );

    case ASSIGNMENT_ACTIONS.ADD_TO_FAVORITES:
      return addToFavoritesAction(callbackQuery, index);

    case
ASSIGNMENT_ACTIONS.MARK_ASSIGNMENT_AS_NOT_COMPLETED:
      return markAssignmentAsNotCompletedAction({
        request: callbackQuery,
        assignmentId: index,
      });

    case ASSIGNMENT_ACTIONS.MARK_ASSIGNMENT_AS_COMPLETED:
      return markAssignmentAsCompletedAction({
        request: callbackQuery,
```

```
      assignmentId: index,
    });

  case ASSIGNMENT_ACTIONS.MARK_ASSIGNMENT_AS_SPAM:
    return markAssignmentAsSpamAction(callbackQuery, index);

  case ASSIGNMENT_ACTIONS.CONFIRM_ASSIGNMENT_AS_SPAM:
    return confirmAssignmentAsSpamAction(callbackQuery, index);

  case
ASSIGNMENT_ACTIONS.BACK_FROM_CONFIRM_ASSIGNMENT_AS_SP
AM:
    return backFromConfirmAssignmentAsSpamAction(callbackQuery, index);

  case ASSIGNMENT_ACTIONS.REMOVE_ASSIGNMENT:
    return removeAssignmentAction(callbackQuery, index);

  case ASSIGNMENT_ACTIONS.CONFIRM_ASSIGNMENT_REMOVE:
    return confirmAssignmentRemoveAction(callbackQuery, index);

  case
ASSIGNMENT_ACTIONS.BACK_FROM_CONFIRM_ASSIGNMENT_REMO
VE:
    return backFromConfirmAssignmentRemoveAction(callbackQuery, index);

  case ASSIGNMENT_ACTIONS.EDIT_ASSIGNMENT:
    // eslint-disable-next-line no-case-declarations
    const { message } = callbackQuery;
    return addMenuSelectCategoryForEditAssignmentAction(
      message,
      null,
      index,
    );

  default:
    return mainMenuAction();
  }
};

module.exports = callbackQueryHandler;
```

myLocationHandler.js

```
const {
  LOCATION_FLOW_STEPS,
} = require('../../constants/flow.step');

const {
  myLocationsChooseAction,
  myLocationsListAction,
} = require('../../actions/myLocationAction');

// eslint-disable-next-line consistent-return
const myLocationsHandler = (request, state) => {
  // eslint-disable-next-line default-case
  switch (state.step) {
    case LOCATION_FLOW_STEPS.CHOOSE_ACTION:
      return myLocationsChooseAction(request, state);
    case LOCATION_FLOW_STEPS.SHOW_LOCATIONS:
      return myLocationsListAction(request, state);
  }
};

module.exports = myLocationsHandler;
```

state.js

```
const { update } = require('../../services').userService;

const setState = async (
  telegramId,
  step = '',
  data = '',
  cache = '',
) => {
  const updatedState = {
    step,
    cache,
    data,
  };
  const result = await update(
    {
      telegramId,
      updatedState,
    },
  );
```

```
  if (!result.succeeded) throw Error(result.message);
};

module.exports = setState;
```

stateLoad.js

```
const { getOne, create } = require('../../services').userService;


const stateLoad = async (message) => {
  if (message.text === '/start') return { data: '', step: '', cache: '' };
  const result = await getOne({ telegramId: message.from.id, params: ['state'] });
  if (!result.succeeded) throw Error(result.message);
  if (!result.model) {
    const resultCreate = await create({
      telegramId: message.from.id,
    });

    if (!resultCreate.succeeded) throw Error(resultCreate.message);
    throw Error('State was not found.');
  }
  const response = {
    step: result.model.state.step || '',
    data: result.model.state.data || '',
    cache: result.model.state.cache || '',
  };

  return response;
};

module.exports = stateLoad;
```

mainAction.js

```
const { telegramTemplate } = require('claudia-bot-builder');
const { MAX_AVAILABLE_RANGE } = require('../configs/global.config');

const {
  ADD_ASSIGNMENT_FLOW_STEPS,
  FIND_ASSIGNMENTS_FLOW_STEPS,
  LOCATION_FLOW_STEPS,
  CHANGE_RANGE_FLOW_STEPS,
} = require('../constants/flow.step');

const { aboutUsMessageTemplate } = require('../templates/aboutUsTemplates');

const {
  mainMenuKeyboardTemplate,
  mainMenuMessageTemplate,
  greetingsMessageTemplate,
} = require('../templates/mainMenuTemplates');

const {
  rangeKeyboardTemplate,
  rangeMessageTemplate,
  rangeResponseMessageTemplate,
} = require('../templates/rangeTemplates');

const {
  myAssignmentsKeyboardTemplate,
  chooseFilterMessageTemplate,
} = require('../templates/assignmentTemplates');

const {
  chooseCategoryKeyboardTemplate,
  chooseCategoryMessageTemplate,
} = require('../templates/categoryTemplates');

const {
  myLocationsMessageTemplate,
  myLocationsKeyboardTemplate,
} = require('../templates/locationTemplates');

const responseMessage = require('../helpers/responseMessage');
```

```
const { setState } = require('../helpers/state');
const { userService } = require('../services');

const startAction = async (message) => {
  const result = await userService.create(
    {
      telegramId: message.from.id,
    },
  );

  if (!result.succeeded) throw Error(result.message);

  const messageStart =
`${greetingsMessageTemplate(message.from.first_name)}\n${aboutUsMessageTe
mplate}`;

  return responseMessage(
    message,
    messageStart,
    mainMenuKeyboardTemplate,
  );
};

const mainMenuAction = async (message) => {
  // eslint-disable-next-line no-use-before-define
  await setState(message.from.id);
  return responseMessage(
    message,
    mainMenuMessageTemplate,
    mainMenuKeyboardTemplate,
  );
};

const aboutUsAction = (message) => responseMessage(
  message,
  aboutUsMessageTemplate,
  mainMenuKeyboardTemplate,
);

const showRangeAction = async (message) => {
  const result = await userService.getOne({ telegramId: message.from.id, params:
['range'] });
```

```
  if (!result.succeeded) throw Error(result.message);

  await setState(message.from.id,
CHANGE_RANGE_FLOW_STEPS.CHANGE_RANGE);

  return new telegramTemplate.Text(rangeMessageTemplate(result.model.range))
    .addReplyKeyboard(rangeKeyboardTemplate)
    .get();
};

const changeRangeAction = async (message) => {
  if (Math.sign(message.text) === 1 && message.text <=
MAX_AVAILABLE_RANGE) {
    const result = await userService.update({
      telegramId: message.from.id,
      newRange: message.text,
    });

    if (!result.succeeded) throw Error(result.message);

    return new telegramTemplate

.Text(rangeResponseMessageTemplate.rangeSuccessResponseMessageTemplate)
      .addReplyKeyboard(
        mainMenuKeyboardTemplate,
        { one_time_keyboard: true },
      )
      .get();
  }
  return new telegramTemplate
    .Text(rangeResponseMessageTemplate.rangeErrorResponseMessageTemplate)
    .get();
};

const myAssignmentAction = () => new telegramTemplate

.Text(chooseFilterMessageTemplate).addReplyKeyboard(myAssignmentsKeyboar
dTemplate).get();

const myLocationsAction = async (message) => {
  await setState(message.from.id,
LOCATION_FLOW_STEPS.CHOOSE_ACTION);
  return new telegramTemplate
    .Text(myLocationsMessageTemplate)
```

```
    .addReplyKeyboard(myLocationsKeyboardTemplate)
    .get();
};

const findAssignmentsAction = async (message) => {
  await setState(message.from.id,
FIND_ASSIGNMENTS_FLOW_STEPS.CHOOSE_CATEGORY);
  return (
    new telegramTemplate.Text(chooseCategoryMessageTemplate)
      .addReplyKeyboard(
        chooseCategoryKeyboardTemplate,
      )
      .get()
  );
};

const addMenuSelectCategoryForCreatedAssignmentAction = async (message) =>
{
  await setState(message.from.id,
ADD_ASSIGNMENT_FLOW_STEPS.CHOOSE_CATEGORY);
  return responseMessage(
    message,
    chooseCategoryMessageTemplate,
    chooseCategoryKeyboardTemplate,
  );
};

module.exports = {
  startAction,
  mainMenuAction,
  aboutUsAction,
  showRangeAction,
  changeRangeAction,
  myAssignmentAction,
  myLocationsAction,
  findAssignmentsAction,
  addMenuSelectCategoryForCreatedAssignmentAction,
};
```

assignmentAction.js

```
const { telegramTemplate } = require('claudia-bot-builder');
```

```
const { locationService, assignmentService } = require('../services');
const addLocationNamesToKeyboard =
require('../helpers/locationNamesKeyboard');

const {
  assignmentMessageTemplate,
  ownAssignmentInlineKeyboardTemplate,
  publicAssignmentInlineKeyboardTemplate,
  findAssignmentsMessageTemplate,
  findAssignmentsKeyboardTemplate,
  favoriteAssignmentsMessageTemplate,
  markAssignmentAsSpamMessageTemplate,
  markAssignmentAsSpamKeyboardTemplate,
  markAssignmentAsSpamAlertTemplate,
  createdAssignmentsMessageTemplate,
  removeAssignmentMessageTemplate,
  removeAssignmentAlertTemplate,
  removeAssignmentKeyboardTemplate,
} = require('../templates/assignmentTemplates');

const {
  addAssignmentLocationMessageTemplate,
} = require('../templates/locationTemplates');

const { COMMON_BUTTONS } = require('../constants/button.text');

const {
  FIND_ASSIGNMENTS_FLOW_STEPS,
  MY_ASSIGNMENTS_FLOW_STEPS,
} = require('../constants/flow.step');

const { setState } = require('../helpers/state');

const {
  deleteMessage,
  editMessageReplyMarkup,
  sendPhoto,
  sendMessage,
  editMessageCaption,
  editMessageText,
  answerCallbackQuery,
} = require('../helpers/telegram');
```

```
const { paginationKeyboardTemplate, paginationMessageTemplate } =
require('../templates/paginationTemplates');
const { PAGE_SIZE } = require('../constants/pageSize');
const { getPagination } = require('../helpers/pagination');

const favoriteAssignmentsAction = async (request, page = 0) => {
  const result = await assignmentService.getAllFavorites({
    telegramId: request.from.id,
    pagination: getPagination(page, PAGE_SIZE),
    page,
  });

  if (!result.succeeded) throw Error(result.message);

  await setState(request.from.id,
MY_ASSIGNMENTS_FLOW_STEPS.GET_FAVORITE_ASSIGNMENTS);

  const { pagingData } = result;
  const assignments = result.model;

  if (!assignments.length) {
    return [
      new telegramTemplate.Text(

favoriteAssignmentsMessageTemplate.noFavoriteAssignmentsMessageTemplate,
      )
        .get(),
    ];
  }

  const basicResponse = [
    new telegramTemplate.Text(
      favoriteAssignmentsMessageTemplate.favoriteAssignmentsMessageTemplate,
    )
      .get(),
  ];

  const paginationResponse = [
    new telegramTemplate.Text(paginationMessageTemplate(pagingData))
      .addInlineKeyboard(paginationKeyboardTemplate(pagingData))
      .get(),
  ];

  const assignmentsResponse = assignments.map((assignment) => {
```

```
    if (assignment.pictureUrl) {
      return sendPhoto(
        request,
        {
          inline_keyboard: publicAssignmentInlineKeyboardTemplate(
            {
              isFavorite: true,
              assignmentId: assignment.id,
              fromFavorites: true,
            },
          ),
        },
        assignmentMessageTemplate(assignment),
        assignment.pictureUrl,
        'Markdown',
      );
    }

    return sendMessage(
      request,
      {
        inline_keyboard: publicAssignmentInlineKeyboardTemplate(
          {
            isFavorite: true,
            assignmentId: assignment.id,
            fromFavorites: true,
          },
        ),
      },
      assignmentMessageTemplate(assignment),
      'Markdown',
    );
  });

  if (pagingData.totalPages === 1) return
basicResponse.concat(assignmentsResponse);

  return basicResponse.concat(assignmentsResponse, paginationResponse);
};

const createdAssignmentsAction = async (request, page = 0) => {
  const result = await assignmentService.getCreated({
    telegramId: request.from.id,
    pagination: getPagination(page, PAGE_SIZE),
```

```
    page,
  });

  if (!result.succeeded) throw Error(result.message);

  await setState(request.from.id,
MY_ASSIGNMENTS_FLOW_STEPS.GET_CREATED_ASSIGNMENTS);

  const { pagingData } = result;
  const assignments = result.model;

  if (!assignments.length) {
    return [
      new telegramTemplate.Text(

createdAssignmentsMessageTemplate.noCreatedAssignmentsMessageTemplate,
      )
        .get(),
    ];
  }

  const basicResponse = [
    new telegramTemplate.Text(
      createdAssignmentsMessageTemplate.createdAssignmentsMessageTemplate,
    )
      .get(),
  ];

  const paginationResponse = [
    new telegramTemplate.Text(paginationMessageTemplate(pagingData))
      .addInlineKeyboard(paginationKeyboardTemplate(pagingData))
      .get(),
  ];

  const assignmentsResponse = assignments.map((assignment) => {
    if (assignment.pictureUrl) {
      return sendPhoto(
        request,
        {
          inline_keyboard: ownAssignmentInlineKeyboardTemplate({
            status: assignment.status,
            assignmentId: assignment.id,
          }),
        },
```

```
      assignmentMessageTemplate(assignment),
      assignment.pictureUrl,
      'Markdown',
    );
  }

  return sendMessage(
    request,
    {
      inline_keyboard: ownAssignmentInlineKeyboardTemplate({
        status: assignment.status,
        assignmentId: assignment.id,
      }),
    },
    assignmentMessageTemplate(assignment),
    'Markdown',
  );
 });

 if (pagingData.totalPages === 1) return
basicResponse.concat(assignmentsResponse);

 return basicResponse.concat(assignmentsResponse, paginationResponse);
};

const addFoundAssignmentCategoryAction = async (message, state) => {
 const result = await locationService.getAllByTelegramId({ telegramId:
message.from.id });

 if (!result.succeeded) throw Error(result.message);

 const { keyboard } = await addLocationNamesToKeyboard(message.from.id);
 await setState(
  message.from.id,
  FIND_ASSIGNMENTS_FLOW_STEPS.CHOOSE_LOCATION,
  {
   ...state.data,
   telegramId: message.from.id,
   // TODO: заменить, когда будет локализация
   foundAssignmentChosenCategory: message.text !==
COMMON_BUTTONS.BACK
    ? message.text : state.data.foundAssignmentChosenCategory,
  },
  result.model,
```

```
  );

  return (
    new telegramTemplate.Text(addAssignmentLocationMessageTemplate)
      .addReplyKeyboard(
        keyboard,
      )
      .get());
};

const addFoundAssignmentLocationAction = async ({ request, state, page = 0 })
=> {
  const locationId = state.data.locationId
    ? state.data.locationId
    : state.cache.find((elem) => elem.localName === request.text).id;

  const result = await assignmentService.getAllNearby({
    telegramId: request.from.id,
    category: state.data.foundAssignmentChosenCategory,
    locationId,
    pagination: getPagination(page, PAGE_SIZE),
    page,
  });

  if (!result.succeeded) throw Error(result.request);

  const { pagingData } = result;
  const assignments = result.model;

  await setState(
    request.from.id,
    FIND_ASSIGNMENTS_FLOW_STEPS.GET_ASSIGNMENTS,
    {
      ...state.data,
      locationId,
    },
    [
      state.cache,
    ],
  );

  if (!assignments.length) {
    return [
      new telegramTemplate
```

```
.Text(findAssignmentsMessageTemplate.notFoundAssignmentsMessageTemplate)
        .addReplyKeyboard(findAssignmentsKeyboardTemplate)
        .get(),
    ];
  }

  const basicResponse = [
    new
telegramTemplate.Text(findAssignmentsMessageTemplate.foundAssignmentsMes
sageTemplate)
      .addReplyKeyboard(findAssignmentsKeyboardTemplate)
      .get(),
  ];

  const paginationResponse = [
    new telegramTemplate.Text(paginationMessageTemplate(pagingData))
      .addInlineKeyboard(paginationKeyboardTemplate(pagingData))
      .get(),
  ];

  const assignmentsResponse = assignments.map((assignment) => {
    if (assignment.pictureUrl) {
      return sendPhoto(
        request,
        {
          inline_keyboard: publicAssignmentInlineKeyboardTemplate(
            {
              isFavorite: assignment.isFavorite,
              assignmentId: assignment.id,
            },
          ),
        },
        assignmentMessageTemplate({
          ...assignment,
          locationName: assignment.globalName,
        }),
        assignment.pictureUrl,
        'Markdown',
      );
    }

    return sendMessage(
      request,
```

```
      {
        inline_keyboard: publicAssignmentInlineKeyboardTemplate(
          {
            isFavorite: assignment.isFavorite,
            assignmentId: assignment.id,
          },
        ),
      },
      assignmentMessageTemplate({ ...assignment, locationName:
assignment.globalName }),
      'Markdown',
    );
  });

  if (pagingData.totalPages === 1) return
basicResponse.concat(assignmentsResponse);

  return basicResponse.concat(assignmentsResponse, paginationResponse);
};

const removeFromFavoritesAction = async ({ request, assignmentId,
fromFavorites }) => {
  const result = await assignmentService.removeFromFavorites({
    telegramId: request.from.id,
    assignmentId,
  });

  if (!result.succeeded) throw Error(result.message);

  if (fromFavorites === 'false') {
    if (!request.message.photo) {
      return editMessageReplyMarkup(request, {
        inline_keyboard: publicAssignmentInlineKeyboardTemplate(
          {
            isFavorite: false,
            assignmentId,
          },
        ),
      });
    }

    return editMessageReplyMarkup(
      request,
      {
```

```
      inline_keyboard: publicAssignmentInlineKeyboardTemplate(
        {
          isFavorite: false,
          assignmentId,
        },
      ),
    },
  );
}

const response = await favoriteAssignmentsAction(request);

return [
  ...response,
  deleteMessage(request, 1),
  deleteMessage(request),
  deleteMessage(request, -1),
];
};

const addToFavoritesAction = async (request, assignmentId) => {
  const result = await assignmentService.addToFavorites({
    telegramId: request.from.id,
    assignmentId,
  });

  if (!result.succeeded) throw Error(result.message);

  if (!request.message.photo) {
    return editMessageReplyMarkup(request, {
      inline_keyboard: publicAssignmentInlineKeyboardTemplate(
        {
          isFavorite: true,
          assignmentId,
        },
      ),
    });
  }

  return editMessageReplyMarkup(
    request,
    {
      inline_keyboard: publicAssignmentInlineKeyboardTemplate(
        {
```

```
        isFavorite: true,
        assignmentId,
      },
    ),
  },
 );
};

const removeAssignmentAction = (request, assignmentId) => {
 if (!request.message.photo) {
  return editMessageText(
   request,
   {
    inline_keyboard: removeAssignmentKeyboardTemplate(assignmentId),
   },
   removeAssignmentMessageTemplate,
  );
 }

 return editMessageCaption(
  request,
  {
   inline_keyboard: removeAssignmentKeyboardTemplate(assignmentId),
  },
  removeAssignmentMessageTemplate,
 );
};

const confirmAssignmentRemoveAction = async (request, assignmentId) => {
 const result = await assignmentService.delete({
  telegramId: request.from.id,
  assignmentId,
 });
 if (!result.succeeded) throw Error(result.message);

 const response = await createdAssignmentsAction(request);
 const alertResponse = answerCallbackQuery(request,
removeAssignmentAlertTemplate, true);

 return [
  ...response,
  deleteMessage(request, 1),
  deleteMessage(request),
  deleteMessage(request, -1),
```

```
    alertResponse,
  ];
};

const backFromConfirmAssignmentRemoveAction = async (request,
assignmentId) => {
  const result = await assignmentService.get({
    telegramId: request.from.id,
    assignmentId,
  });

  if (!result.succeeded) throw Error(result.message);

  if (result.model) {
    const assignment = result.model;

    if (!request.message.photo) {
      return editMessageText(
        request,
        {
          inline_keyboard: ownAssignmentInlineKeyboardTemplate({
            status: assignment.status,
            assignmentId: assignment.id,
          }),
        },
        assignmentMessageTemplate({
          ...assignment,
          locationName: assignment.locationName,
        }),
        'Markdown',
      );
    }

    return editMessageCaption(
      request,
      {
        inline_keyboard: ownAssignmentInlineKeyboardTemplate({
          status: assignment.status,
          assignmentId: assignment.id,
        }),
      },
      assignmentMessageTemplate({
        ...assignment,
        locationName: assignment.locationName,
```

```
    }),
    'Markdown',
  );
 }

 return deleteMessage(request);
};

const markAssignmentAsCompletedAction = async ({ request, assignmentId }) =>
{
 const result = await assignmentService.update({
  telegramId: request.from.id,
  assignmentId,
  status: 'done',
 });

 if (!result.succeeded) throw Error(result.message);

 return editMessageReplyMarkup(request, {
  inline_keyboard: ownAssignmentInlineKeyboardTemplate(
   {
    assignmentId,
    status: 'done',
   },
  ),
 });
};

const markAssignmentAsNotCompletedAction = async ({ request, assignmentId })
=> {
 const result = await assignmentService.update({
  telegramId: request.from.id,
  assignmentId,
  status: 'notDone',
 });

 if (!result.succeeded) throw Error(result.message);

 return editMessageReplyMarkup(request, {
  inline_keyboard: ownAssignmentInlineKeyboardTemplate(
   {
    assignmentId,
    status: 'notDone',
   },
```

```
    ),
  });
};

const markAssignmentAsSpamAction = (request, assignmentId) => {
  if (!request.message.photo) {
    return editMessageText(
      request,
      {
        inline_keyboard: markAssignmentAsSpamKeyboardTemplate(assignmentId),
      },
      markAssignmentAsSpamMessageTemplate,
    );
  }

  return editMessageCaption(
    request,
    {
      inline_keyboard: markAssignmentAsSpamKeyboardTemplate(assignmentId),
    },
    markAssignmentAsSpamMessageTemplate,
  );
};

const confirmAssignmentAsSpamAction = async (request, assignmentId) => {
  const result = await assignmentService.markAsSpam({ telegramId:
request.from.id, assignmentId });

  if (!result.succeeded) throw Error(result.message);

  const deleteResponse = deleteMessage(request);
  const alertResponse = answerCallbackQuery(request,
markAssignmentAsSpamAlertTemplate, true);

  return [deleteResponse, alertResponse];
};

const backFromConfirmAssignmentAsSpamAction = async (request,
assignmentId) => {
  const result = await assignmentService.get({
    telegramId: request.from.id,
    assignmentId,
  });
```

```
  if (!result.succeeded) throw Error(result.message);

  if (result.model) {
    const assignment = result.model;

    if (!request.message.photo) {
      return editMessageText(
        request,
        {
          inline_keyboard: publicAssignmentInlineKeyboardTemplate(
            {
              isFavorite: assignment.isFavorite,
              assignmentId: assignment.id,
            },
          ),
        },
        assignmentMessageTemplate({
          ...assignment,
          locationName: assignment.locationName,
        }),
        'Markdown',
      );
    }

    return editMessageCaption(
      request,
      {
        inline_keyboard: publicAssignmentInlineKeyboardTemplate(
          {
            isFavorite: assignment.isFavorite,
            assignmentId: assignment.id,
          },
        ),
      },
      assignmentMessageTemplate({
        ...assignment,
        locationName: assignment.locationName,
      }),
      'Markdown',
    );
  }

  return deleteMessage(request);
};
```

```
module.exports = {
  favoriteAssignmentsAction,
  createdAssignmentsAction,
  addFoundAssignmentCategoryAction,
  addFoundAssignmentLocationAction,
  removeFromFavoritesAction,
  addToFavoritesAction,
  markAssignmentAsSpamAction,
  confirmAssignmentAsSpamAction,
  backFromConfirmAssignmentAsSpamAction,
  removeAssignmentAction,
  markAssignmentAsNotCompletedAction,
  markAssignmentAsCompletedAction,
  confirmAssignmentRemoveAction,
  backFromConfirmAssignmentRemoveAction,
};
```

locationAction.js

```
/* eslint-disable default-case */
/* eslint-disable no-else-return */
/* eslint-disable consistent-return */
/* eslint-disable no-use-before-define */
const openGeocoder = require('node-open-geocoder');

const {
  ADD_LOCATION_FLOW_STEPS,
  ADD_ASSIGNMENT_FLOW_STEPS,
  EDIT_ASSIGNMENT_FLOW_STEPS,
  FIND_ASSIGNMENTS_FLOW_STEPS,
} = require('../constants/flow.step');

const {
  addLocationNameMessageTemplate,
  returnMainMenuMessageAfterCreateLocationTemplate,
  returnMainMenuMessageAfterUpdateLocationTemplate,
} = require('../templates/locationTemplates');

const { mainMenuKeyboardTemplate } =
require('../templates/mainMenuTemplates');
```

```
const addLocationNamesToKeyboard =
require('../helpers/locationNamesKeyboard');
const responseMessage = require('../helpers/responseMessage');

const { buttonBackHandler } = require('../features/buttonHandlers');
const { BACK } = require('../constants/button.text').COMMON_BUTTONS;

const { locationService } = require('../services');
const { setState } = require('../helpers/state');

const addLocationAction = async (message, state) => {
  const { location } = message;
  const globalName = await addGlobalName(location);
  const locationNameKeyboard = await
addLocationNamesToKeyboard(message.from.id, false);

  await setState(
    message.from.id,
    ADD_LOCATION_FLOW_STEPS.ADD_LOCATION_NAME,
    {
      telegramId: message.from.id,
      globalName,
      coordinates: [
        location.longitude,
        location.latitude,
      ],
    },
    {
      ...state.cache,
      locationNameKeyboard: locationNameKeyboard.cache,
    },
  );

  return responseMessage(
    message,
    addLocationNameMessageTemplate,
    locationNameKeyboard.keyboard,
    null,
    null,
    2,
  );
};

const addLocalNameLocationAction = async (message, state) => {
```

```
  let messageResponse = '';
  if (state.cache.locationNameKeyboard[message.text]) {
    // eslint-disable-next-line no-const-assign
    await updateOldLocation(message, state);
    messageResponse = returnMainMenuMessageAfterUpdateLocationTemplate;
  } else {
    // eslint-disable-next-line no-const-assign
    await createNewLocation(message, state);
    messageResponse = returnMainMenuMessageAfterCreateLocationTemplate;
  }

  if (state.cache.stepToReturn) {
    return await buttonBackHandler(
      { ...message, text: BACK },
      {
        step: stepToReturn(state.cache.stepToReturn),
        data: state.cache.data,
        cache: { ...state.cache.cache, deleteMessage: 2 },
      },
    );
  } else {
    messageResponse = messageResponse.replace(/Локация/, `Локация
${message.text}`);
    await setState(message.from.id, state.cache.stepToReturn);

    return responseMessage(
      message,
      messageResponse,
      mainMenuKeyboardTemplate,
      null,
      null,
      2,
    );
  }
};

module.exports = {
  addLocationAction,
  addLocalNameLocationAction,
};

async function addGlobalName(location) {
  return new Promise((resolve, reject) => (
    openGeocoder()
```

```
      .reverse(location.longitude, location.latitude)
      .end((err, { address }) => {
        // eslint-disable-next-line prefer-promise-reject-errors
        if (err) reject(Error('Global name hasn\'t created.'));
        const response = (
          address.city
          || address.town
          || address.town
          || address.village
          || address.hallmet
          || address.state
          || ''
        );
        if (!address.road) return resolve(response);

        let { road } = address;
        const str = (road.match(/(^| )(пер|пров|наб|бульв|в?ул|п(р|л))/i) || [''])[0];
        road = road.replace(/(^|
)((пере|пров)улок|набережная?|бульвар|в?улиц(а|я)|проспект|площа(дь)?)/i, '');
        resolve(`${response} ${str || ''}. ${road || ''}`);
      })
  ));
}

async function createNewLocation(message, state) {
  const result = await locationService.create({
    ...state.data,
    localName: message.text,
  });
  if (!result.succeeded) throw Error(result.message);
}

async function updateOldLocation(message, state) {
  const result = await locationService.updateLocation({
    ...state.data,
    id: state.cache.locationNameKeyboard[message.text],
  });
  if (!result.succeeded) throw Error(result.message);
}

function stepToReturn(step) {
  switch (step) {
    case ADD_ASSIGNMENT_FLOW_STEPS.CHOOSE_LOCATION:
      return ADD_ASSIGNMENT_FLOW_STEPS.ADD_REWARD;
```

```
    case EDIT_ASSIGNMENT_FLOW_STEPS.CHOOSE_LOCATION:
      return EDIT_ASSIGNMENT_FLOW_STEPS.EDIT_REWARD;

    case FIND_ASSIGNMENTS_FLOW_STEPS.CHOOSE_LOCATION:
      return FIND_ASSIGNMENTS_FLOW_STEPS.GET_ASSIGNMENTS;
  }
}
```

commonAction.js

```
const { telegramTemplate } = require('claudia-bot-builder');
const { defaultMessageTemplate } =
require('../templates/defaultMessageTemplates');
const { mainMenuKeyboardTemplate } =
require('../templates/mainMenuTemplates');
const {
  addLocationMessageTemplate,
  addLocationKeyboardTemplate,
} = require('../templates/locationTemplates');

const responseMessage = require('../helpers/responseMessage');

const { setState } = require('../helpers/state');

const {
  ADD_LOCATION_FLOW_STEPS,
  ADD_ASSIGNMENT_FLOW_STEPS,
  EDIT_ASSIGNMENT_FLOW_STEPS,
  FIND_ASSIGNMENTS_FLOW_STEPS,
  MY_ASSIGNMENTS_FLOW_STEPS,
  LOCATION_FLOW_STEPS,
} = require('../constants/flow.step');

const { COMMON_BUTTONS } = require('../constants/button.text');

const {
  favoriteAssignmentsAction,
  createdAssignmentsAction,
  addFoundAssignmentLocationAction,
} = require('./assignmentActions');

const messageDefaultAction = () => (
  new telegramTemplate
```

```
    .Text(defaultMessageTemplate)
    .addReplyKeyboard(
      mainMenuKeyboardTemplate,
      { one_time_keyboard: true },
    )
    .get());

const paginationAction = async (request, page, state) => {
  let response;
  switch (state.step) {
    case
MY_ASSIGNMENTS_FLOW_STEPS.GET_CREATED_ASSIGNMENTS:
      response = await createdAssignmentsAction(request, page, state);

      break;
    case
MY_ASSIGNMENTS_FLOW_STEPS.GET_FAVORITE_ASSIGNMENTS:
      response = await favoriteAssignmentsAction(request, page, state);

      break;
    case FIND_ASSIGNMENTS_FLOW_STEPS.GET_ASSIGNMENTS:
      response = await addFoundAssignmentLocationAction({ request, page, state
});

      break;
    default:
      response = messageDefaultAction();
  }
  return response;
};

const addMenuAddLocationAction = async (message, state) => {
  let cache;
  if (message.text === COMMON_BUTTONS.BACK) {
    cache = state.cache;
  } else {
    cache = {
      stepToReturn: state.step,
      data: state.data,
      cache: state.cache,
    };
  }
  if (state.step === LOCATION_FLOW_STEPS.CHOOSE_ACTION) {
    await setState(
```

```
      message.from.id,
      // LOCATION_FLOW_STEPS.ADD_LOCATION,
      ADD_LOCATION_FLOW_STEPS.ADD_LOCATION,
      null,
      { stepToReturn: '', data: state.data, cache: state.cache },
      // cache,
    );
  } else {
    await setState(
      message.from.id,
      ADD_LOCATION_FLOW_STEPS.ADD_LOCATION,
      null,
      cache,
    );
  }

  return responseMessage(
    message,
    addLocationMessageTemplate,
    addLocationKeyboardTemplate,
    null,
    null,
    // eslint-disable-next-line no-use-before-define
    countMessagesDelete(state),
  );
};

module.exports = {
  messageDefaultAction,
  paginationAction,
  addMenuAddLocationAction,
};

function countMessagesDelete(state) {
  switch (state.step) {
    case ADD_LOCATION_FLOW_STEPS.ADD_LOCATION_NAME:
    case FIND_ASSIGNMENTS_FLOW_STEPS.CHOOSE_LOCATION:
      return 2;

    case ADD_ASSIGNMENT_FLOW_STEPS.CHOOSE_LOCATION:
    case EDIT_ASSIGNMENT_FLOW_STEPS.CHOOSE_LOCATION:
      return 3;

    default:
```

```
    return 1;
  }
}
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

«Розробка програмного забезпечення для подачі оголошень та зв'язку між сусідами на прикладі Telegram бота»

студента групи 121-17-1 Юдіна Єгора Валерійовича

Керівник економічного розділу **О. Г. Вагонова**

Зав. каф. ПЕП та ПУ, д.е.н.

ДОДАТОК В

Перелік файлів на диску

Перелік документів на магнітному носії

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Юдін.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Юдіна.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Юдін.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Юдін.ppt	Презентація кваліфікаційної роботи