МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна система розміщення об'яв про загублені речі»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Берест О.Б.**

Студент гр. ІН-71 **Котелевський К.В.**

СУМИ 2021

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Зав. секцією

_____ А. С. Довбиш
«___» _____ 2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-71 спеціальності «Інформатика» денної форми навчання Котелевський Кирило Володимирович.

Тема: «Інформаційна система розміщення об'яв про загублені речі»

Затверджена наказом по СумДУ
№_____ от _____2021 г.

Зміст пояснювальної записки: 1) аналіз предметної області; 2) постановка завдання й загальне формулювання задач інформаційної системи; 3) огляд технологій для реалізації завдання; 4) розробка телеграм-бота; 5) аналіз виконаної роботи.

Дата видачі завдання "_____"_____2020 г.
Керівник випускної роботи _____ Берест О.Б.
Завдання прийняв до виконання _____ Котелевський К.В.

РЕФЕРАТ

Записка: 53 стр., 21 рис., 4 табл., 1 додаток, 20 джерел.

Об’єкт дослідження — інформаційні процеси автоматичного розміщення об'яв.

Мета роботи — розробка інформаційної система для розміщення об'яв про загублені речі.

Методи дослідження — метод аналітично-статистичний.

Результати — розроблено інформаційна система розміщення об'яв про загублені речі.

Ключові слова: REST API, JAVA, ТЕЛЕГРАМ-БОТ, TELEGRAM, ОГОЛОШЕННЯ, ПОШУК, ELASTICSEARCH, HEROKU.

ЗМІСТ

ВСТУП

Як відомо, кожного року сотні й тисячі людей України можуть ненароком загубити важливі речі, чи то документи, чи гаманець. За зібраною статистикою, лише за минулий 2020 рік поліція зареєстровала приблизно 55 тисяч заяв від громадян про втрату або крадіжку внутрішнього паспорта[1].

Частина громадян, які знайшли загублені речі, намагаються знайти власника, але стикаються з тим що існує кілька сервісів для розміщення оголошень в Україні. У зв’язку з цим пошук потрібного оголошення ускладняється, як і процедура розміщення оголошення. Також проблемою подібних сервісів є те - що про них ніхто не знає, тож було вирішено збирати інформацію згідно оголошень з інших ресурсів до бази даних чат-боту. Таким чином бот зможе бути корисним раніше ніж користувачі почнуть додавати велику кількість оголошень.

Вибір саме чат-боту як засобу взаємодії з користувачем зумовлений тим, що така реалізація не вимагає від користувача реєстрацій на сайті або завантаження мобільних додатків. Для використання потрібен лише встановлений месенджер, який на разі користуються високою популярністю. Також це дозволяє розробляти лише серверну частину системи, оскільки взаємодія з користувачем проходить через інтерфейс месенджера для якого створюється чат-бот.

Мета проекту – розробка телеграм-бота для розміщення об'яв про загублені речі:

- ознайомлення з сферами використання телеграм-ботів;
- аналіз використання сервісів;
- обрання інструментарію для реалізації проектної роботи;
- розробка схеми та sql-скрипта реалізації бази даних;
- розробка телеграм-бота;

– тестування розробленого телеграм-бота на прикладах, схожими із реальними даними та інформацією.

Також буде реалізований парсинг вже реалізованих та існуючих ресурсів із подібними оголошеннями з метою збереження їх до бази даних чат-боту. Це допоможе виконати розширення інформації та даних для використання. Для налаштування пошукової системи буде використано пошукову систему ElasticSearch.

Телеграм-бот для розміщення об'яв про загублені речі повинен реалізовувати такі функціонуючи блоки як:

– створення оголошення;
– пошук за містом;
– перегляд інших заяв;
– видалення власних оголошень.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

Як відомо, телеграм-боти це частіше всього додаткові сторонні програми, які працюють та використовуються всередині додатку «Telegram». Будь-який користувач може безпосередньо взаємодіяти із створеними ботами. Дана взаємодія відбувається за допомогою надсилання повідомлень, команд чи використовуючи вбудовані запити у додаток.

Розглянемо популярні сфери та напрямки використання телеграм-ботів.

Таблиця 1.1 – Сфери використання телеграм-ботів

№	Напрямок	Детальний опис
1	Отримання індивідуальних сповіщень та новин	У сучасних реаліях телеграм-бот може виступати аналогом «розумної газети». Адже бот надсилає користувачеві відповідний вміст новин чи інших публікацій відразу після його оновлення.
2	Якісна інтеграція з різними службами	У телеграм-бот можна додати інформацію через сторонні ресурси, тим самим доповнити базу даних.
3	Приймання платежі від інших користувачів Telegram	У даному ресурсі можна запропонувати додаткові платні послуги чи працювати як онлайн-магазин.
4	Розробка та використання власних інструментів	Бот може надавати вам попередження, прогнози погоди, переклади, форматування чи інші послуги.

Продовження таблиці 1.1.

№	Напрямок	Детальний опис
5	Розробка ігор	У телеграм-бот можна додати розроблену власну гру. Це може бути як проста аркада чи головоломок, так й 3D-шутер.
6	Робота із соціальними послугами	Даний інструмент можна також використовувати як інструмент для налагодження комунікації, знайомство з новими людьми й інше.

По суті, Telegram-боти у сучасних реаліях – своєрідні облікові записи. Головна відмінність в тому, що для цього налаштування не потрібен додатковий номер телефону.

Розглянемо два головних та найактуальніших способи взаємодії з телеграм-ботами:

1. Надсилання повідомлень чи використання вбудованих команди;
2. Надсилання запитів безпосередньо з поля введення повідомлення;

Повідомлення, команди та запити, надіслані користувачами, передаються до розробленого та заімплементованого програмного забезпечення, що працює на реалізованих розробником серверах.

1.2 Аналіз аналогів для пошуку загублених речей

У світі сучасних технологій ми звичайно маємо безліч можливостей та велику кількість реалізацій схожих за функціонуванням задач. Не дивлячись на це, не кожна з даних можливостей може реалізувати весь запропонований потенціал задачі або представляє з себе не повну та неякісну реалізацію.

Для пошуку загублених речей на ринку реалізовано декілька веб-систем обраного напрямку. Розглянемо обрані приклади. Дані веб-додатки реалізовують

модулі та певний функціонал, що буде виконаний в телеграм-боті по пошуку загублених речей.

Обрані приклади:

– «verni.com.ua»;

– «luckfind.me»;

Розглянемо першу веб сторінку, а саме «verni.com.ua» (рис.1.1). Даний веб-додаток представляє з себе дошку оголошень, що має приблизно 2000 об’яви про пропажу чи умисну крадіжку та більше 500 об’ява про знайдені речі.

Рисунок 1.1 – Головна сторінка додатку веб-додатку «verni.com.ua»

На головній сторінці додатку представлена коротка само-реклама та швидкий пошук за критерієм та введеним словом.

Самі нахідки мають блочний формат відображення (рис.1.2). Реалізований функціонал – реєстрація, розміщення та пошук оголошень

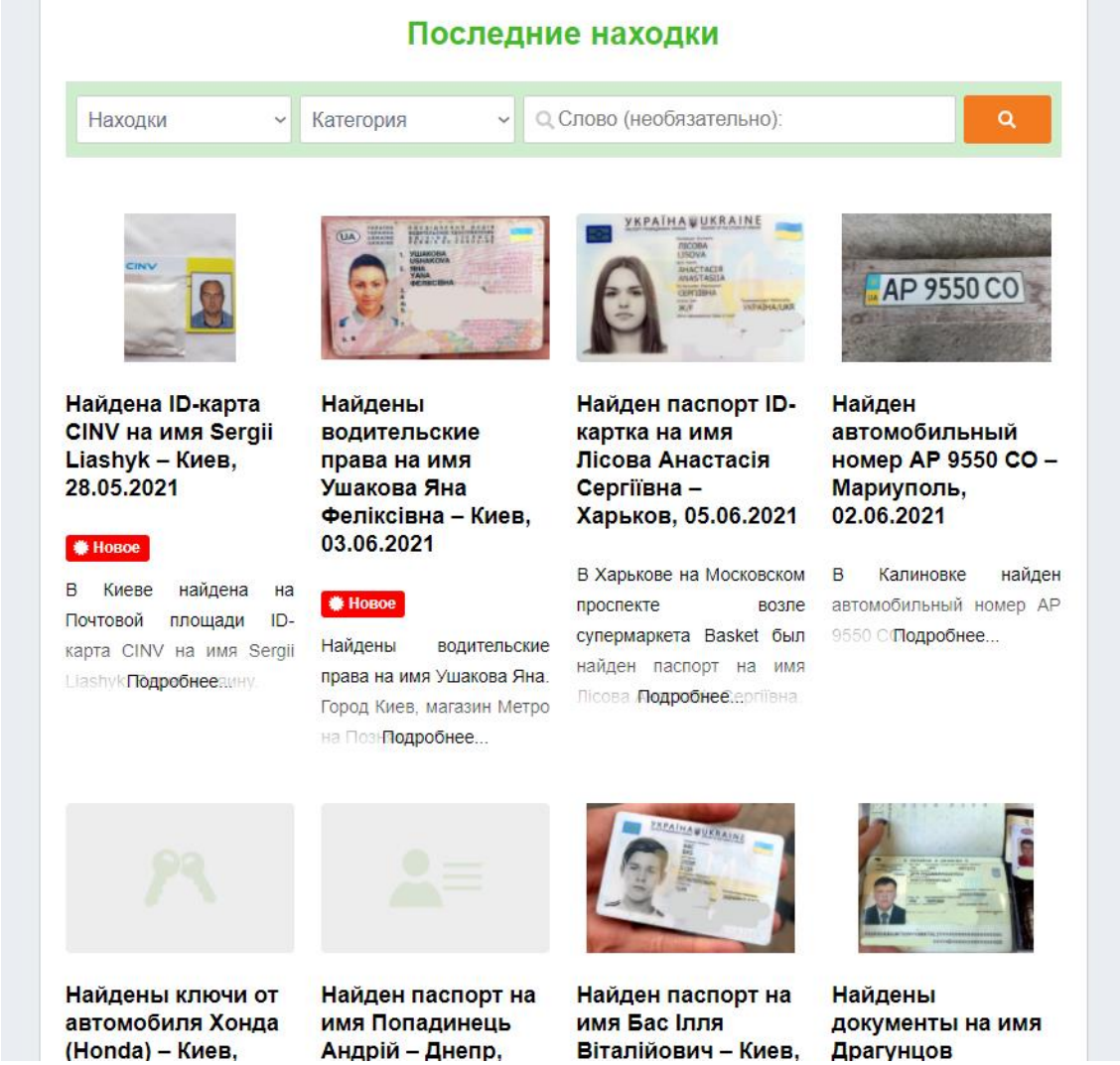

Рисунок 1.2 – Блоки із нахідками

Переходячи за певним посиланням натискаючи на обраний блок, користувач може переглянути більш детальну інформацію про знахідку (рис.1.3), а саме опис місця знаходження документу.

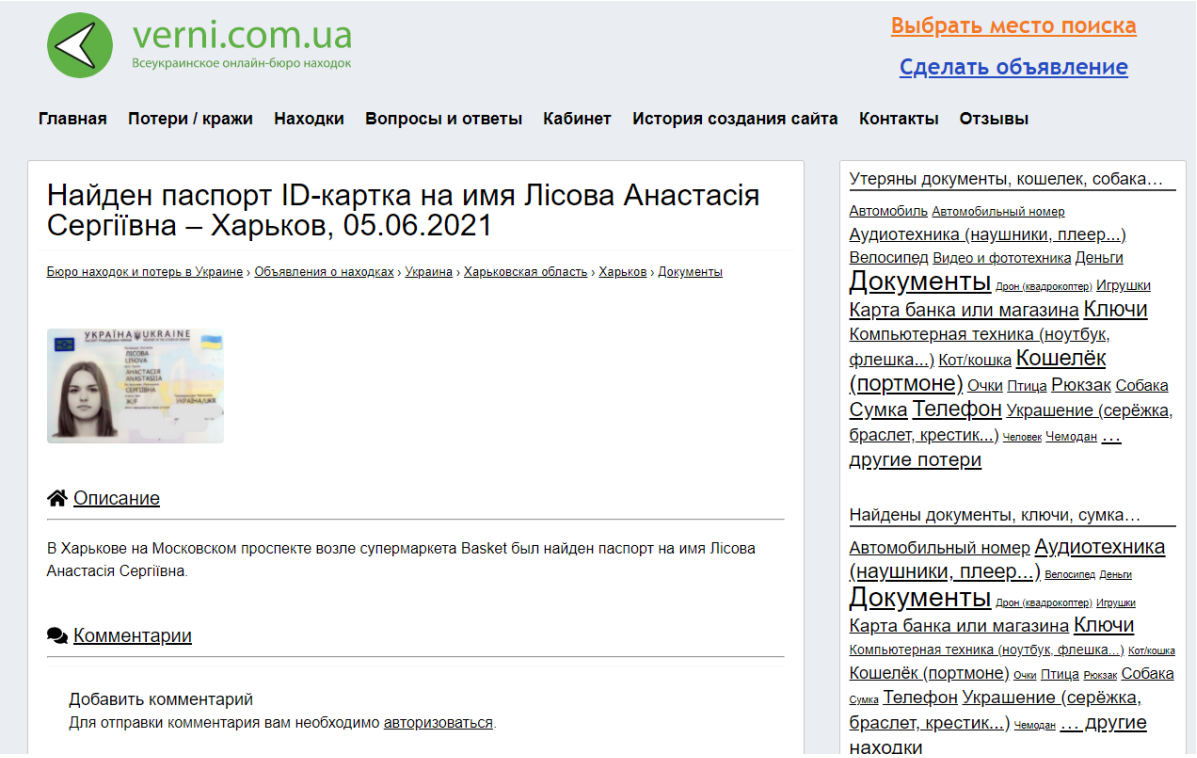

Рисунок 1.3 – Детальна інформація

Не дивлячись на реалізацію та досить значну популярність веб-додатку, «verni.com.ua» має й недоліки. Перш за все, це те що для розміщення оголошення потрібно виконати реєстрацію або авторизацію за допомогою Facebook-аккаунту чи Google- аккаунту.

Крім того, використовуючи прямий пошук, виникають проблеми, якщо є потреба ввести та виконати пошук за декількома словами.

Наступний веб-додаток аналог власного проекту по пошуку загублених речей – «luckfind.me». Даний додаток має цікавий дизайн та власну особливість, а саме мапа із відміченими оголошеннями (рис.1.4).

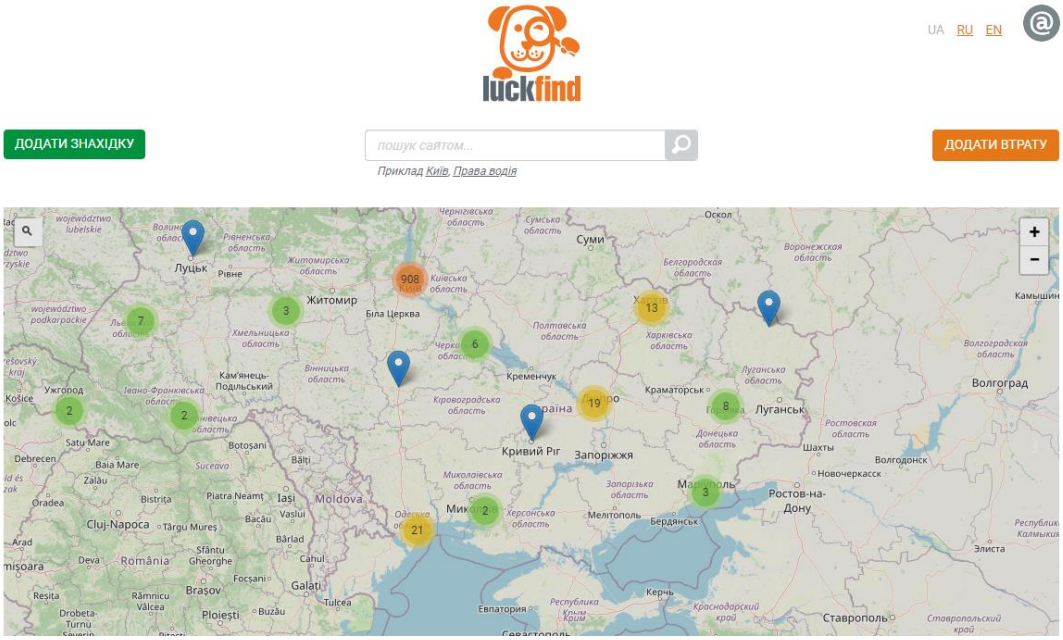

Рисунок 1.4 – Головна сторінка додатку веб-додатку «luckfind.me»

Даний веб-додаток є більш популярний ніж «verni.com.ua», так як має більше 8000 об’яв про пропажі, й приблизно 1800 об’ява про знахідки речей за різними категоріями. Головна ідея та як результат реалізований функціонал – розміщення оголошень.

Розглянемо приклад оголошення на рис.1.5.

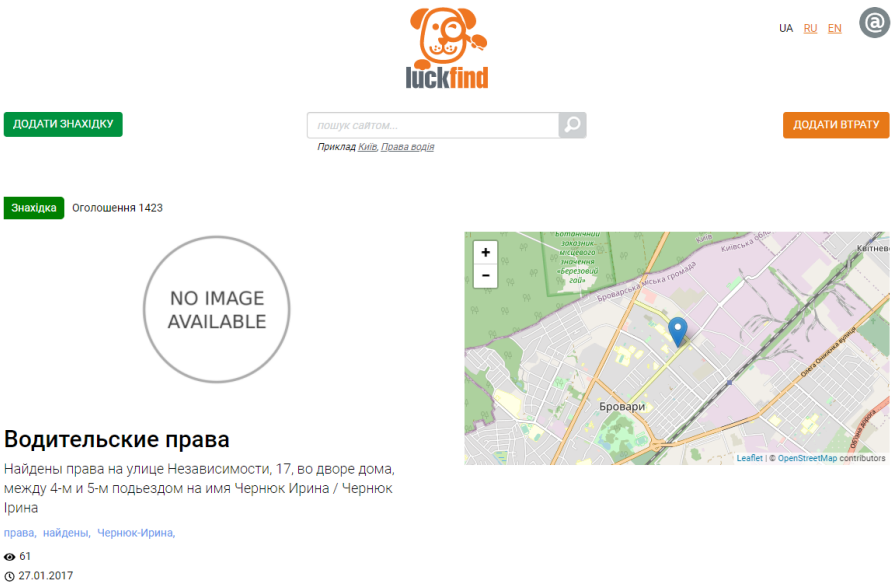

Рисунок 1.5 – Оголошення

До головних недоліків веб-додатку можна віднести відсутність точного пошук за містом знахідки та не врахування пошуком однокорінних слів.

У результаті аналізу вед-додатків конкурентів було сформоване загальне уявлення про функціональні запити користувачів та їх потреби.

1.3 Постановка задачі проекту

Актуальність бакалаврської роботи зумовлена дуже високою популярністю месенджерів й таких відомих засобів автоматизації, як чат-боти серед звичайних користувачів мережі Інтернет. Чат-боти дозволяють спростити щоденні рутинні завдання, такі як отримання інформації про погоду, пробки, останні новини та інші. Головним достоїнством щодо класичних додатків є можливість заміщення всіх можливостей на платформі одного месенджера.

Мета проекту – розробка телеграм-бота для розміщення об'яв про загублені речі:

- ознайомлення з сферами використання телеграм-ботів;
- аналіз використання сервісів;
- обрання інструментарію для реалізації проектної роботи;
- розробка схеми та sql-скрипта реалізації бази даних;
- безпосередня розробка телеграм-бота;
- тестування розробленого телеграм-бота на прикладах, схожими із реальними даними та інформацією.

Телеграм-бот для розміщення об'яв про загублені речі повинен реалізовувати такі функціонуючи блоки як:

- створення оголошення;
- пошук за містом;
- перегляд інших заяв;
- видалення власних оголошень.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ТЕЛЕГРАМ-БОТА

2.1 Вибір засобів реалізації

Перш за все потрібно обрати мову програмування та реалізації функціональної частини телеграм-бота. Головні претенденти та найпопулярніші рішення задачі – Java та Python.

Розглянемо переваги та недоліки кожного з них (табл.2.1-3).

Таблиця 2.1 – Переваги використання Java

№	Назва	Опис
1	Проста	Java просто використовувати, виконувати написання коду, компілювати, реалізовувати налагоджувати код, вивчати та вдосконалювати власні навички.
2	Об’єктно-орієнтований підхід	Дозволяє сформувати та використовувати загально-прийняті стандарти по написанню та використанню коду.
3	Незалежна від платформи	Буде працювати на будь-якій машині чи ПК без встановлення спеціального програмного забезпечення, лише при встановлені JRE.
4	Захищений	Java має додатково встановлені менеджер безпеки, що визначає та контролює доступ до розроблених девелопером класів .
5	Розподіл пам'яті	Легко зберігає інформацію, відновлює її та виконує різні спрощення.
7	Багатопоточність	Присутня можливість виконувати реалізацію декількох завдань одночасно.

Таблиця 2.2 – Недоліки використання Java

№	Назва	Опис
1	Продуктивність	Займає багато пам’яті та через власні налаштування та бібліотеки працює значно повільніше.
2	Робота із грфічним інтерфейсом	Відсутність якісно-реалізованих інструментів для розробки проектів з графічним інтерфейсом.
3	Управління пам'яттю	Незначний доступ розробника при роботі із пам’ятю.

Таблиця 2.3 – Переваги та недоліки використання Python

№	Переваги	Недоліки
1	Відкритий код із яскравою спільнотою.	Обмеження швидкодії роботи додатку.
2	Універсальний, простий та швидкий у розробці	Немає якісних інструментів для роботи із мобільним середовищем.
3	Має всі необхідні бібліотеки.	Значне споживання пам'яті.
4	Відмінно підходить для прототипів – адже розробник може реалізувати більше функціоналу, загалом використовуючи менше коду	Примітивний та не розвинені способи роботи із базами даних.
5	Висока продуктивність у невеликих пооектів.	Слабкий рівень для виконання мобільних обчислень.

Отже, у результаті аналізу та детального порівняння мов програмування, для розробки телеграм-бота для пошуку загублених речей було обрано Java.

Розглянемо також детальніше додатковий інструментарій для виконання проекту та усіх поставлених цілей.

По перше – Bot API. Він представляє собою HTTP-інтерфейс для розробки різного типу ботів у додатку Telegram. Кожен бот – це спеціально створений акаунт для автоматичного оброблення та відправлення повідомлень.

Документація Telegram Bot API виділяє два максимально протилежних способи отримання оновлень:

– періодичні запити;
– установлення веб-хуків.

Перший і найбільш простий варіант полягає в періодичному опитуванні серверів Telegram на наявність нової інформації. Відкривається з'єднання на нетривалий час і всі оновлення відразу відправляються боту, все це здійснюється через зв’язок Long Polling. Цей спосіб є простим, але не дуже надійним.

Веб-хуки працюють трохи інакше. Якщо в чат приходить повідомлення, то Telegram сам говорить про це, в цьому і полягає робота вебхука.

Elasticsearch відома як проста у використанні система для реалізації повнотекстового пошуку Використовується також для виконання аналітичних запитів у реальному часі. Elasticsearch дуже простий у використанні та реалізації.

Розглянемо Heroku. Даний інструмент представляє з себе платформу, що безпосередньо використовується для керування та використання інформаційної системи. Для цього виконується інтеграція із службами передачі даних, для розгортання й запуску сучасних додатків.

2.2. Проектування бази даних

Так як робота присвячена розробці чат-боту для розміщення оголошень про загублені речі. Також буде реалізований парсинг вже реалізованих та існуючих ресурсів із подібними оголошеннями з метою збереження їх до бази даних чат-боту.

Це допоможе виконати розширення інформації та даних для використання. Отже, потрібно розробити якісну структуру бази даних для реалізації телеграм-бота. Розглянемо вимоги до БД:

1) В системі немає потреби в складних запитах, робота частіше за все буде вестися с однією сутністю.

2) Оскільки ми маємо отримувати об’яви з інших ресурсів к-ть полів може періодично змінюватися, з додаванням нових джерел даних.

3) Оскільки інтерфейс програми досить мінімалістичний для пошуку об’яв зручно було б використовувати повнотекстовий пошук по документам, для цього можна використати пошукову систему.

Розглянемо спроектовану ER-діаграму телеграм-бота на рис.2.1.

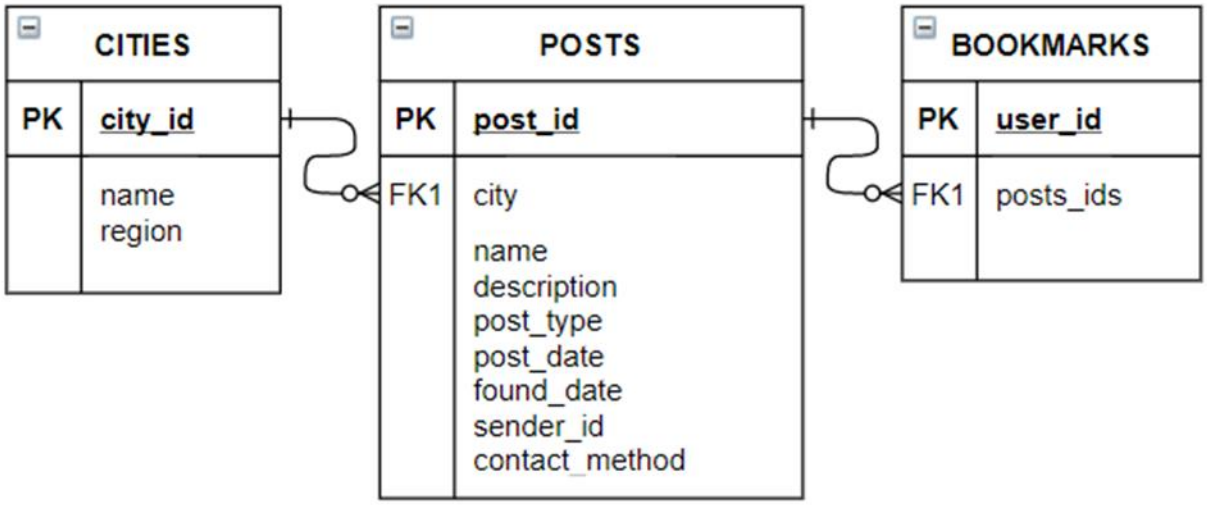

Рисунок 2.1 –ER-діаграма телеграм-бота по пошуку речей

Отже, було створено такі сутності як:

– cities: список міст;
– posts: таблиця зі створеними постами;
– bookmarks: додавання посту до заміток;

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ТЕЛЕГРАМ-БОТА

3.1 Розробка

3.1.1 Реєстрація боту

Перш за все необхідно створити бота в системі Telegram та отримати доступ до нього через месенджер. Отримання доступу було виконане стандартними методами.

Отримання API токену зображено на рисунку 3.1.

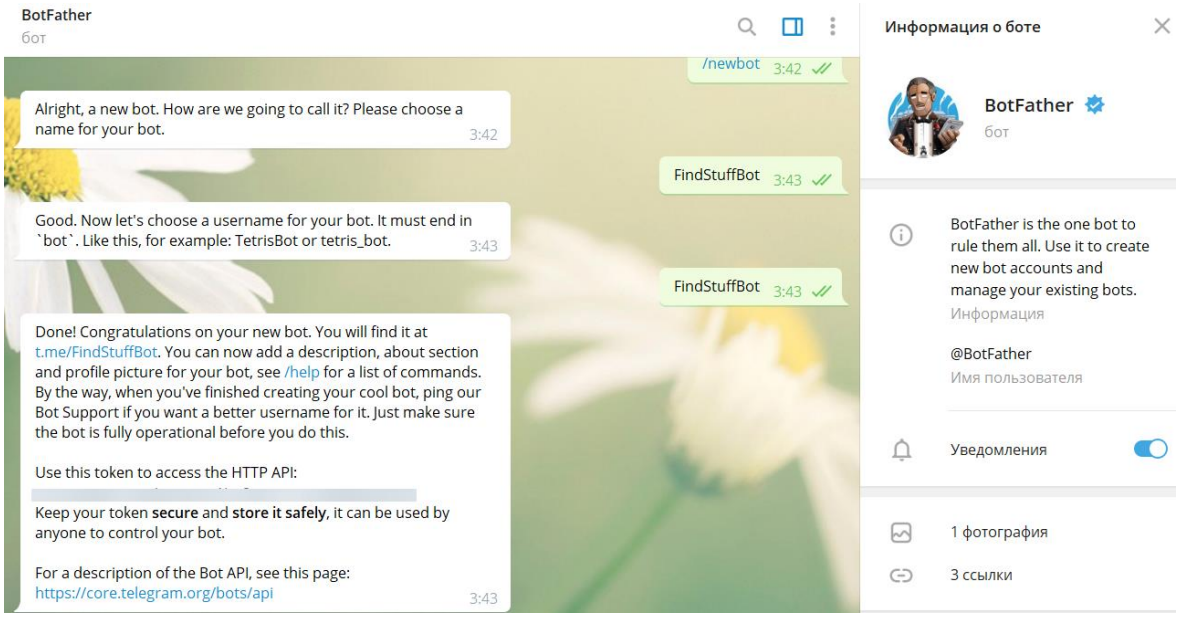

Рисунок 3.1 – Отримання доступу до Telegram боту

Далі потрібно було вказати URL на який Telegram буде відсилати всі оновлення за допомогою «Webhook».

Для цього необхідно надіслати POST-запит на адресу: https://api.telegram.org/bot<Токен боту>/setWebhook?url=<Адресса для оновлень>/ (рис.3.2).

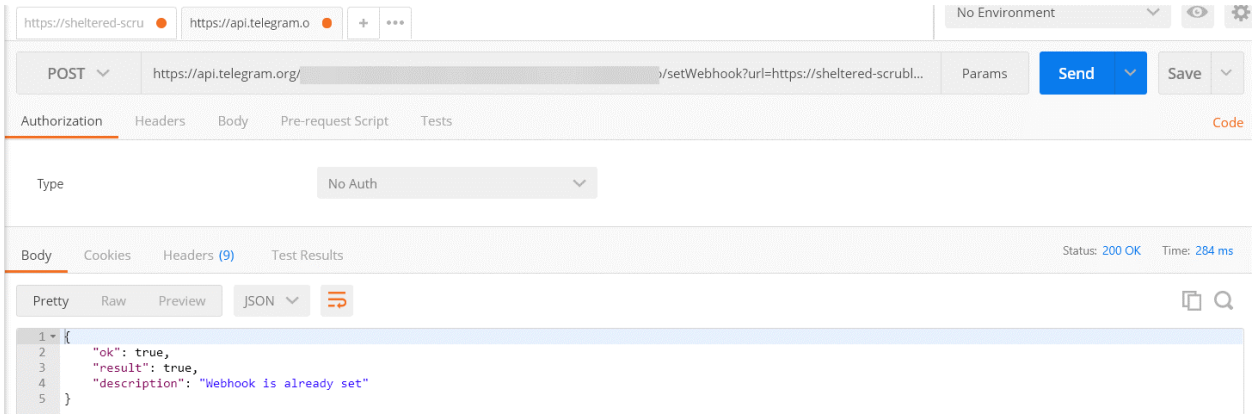

Рисунок 3.2 – Встановлення Webhook адресси для отримання оновлень від Telegram

3.1.2 Конфігурація БД

Скористаймося хмарною платформою «Bonsai». Вона потрібна для отримання та використання у майбутньому бази даних на хостингу Telegram. Для цього необхідно зареєструватися на веб-сервісі https://bonsai.io/ для отримання безкоштовного кластера.

Головна сторінка веб-сервісу представлена на рис.3.3.

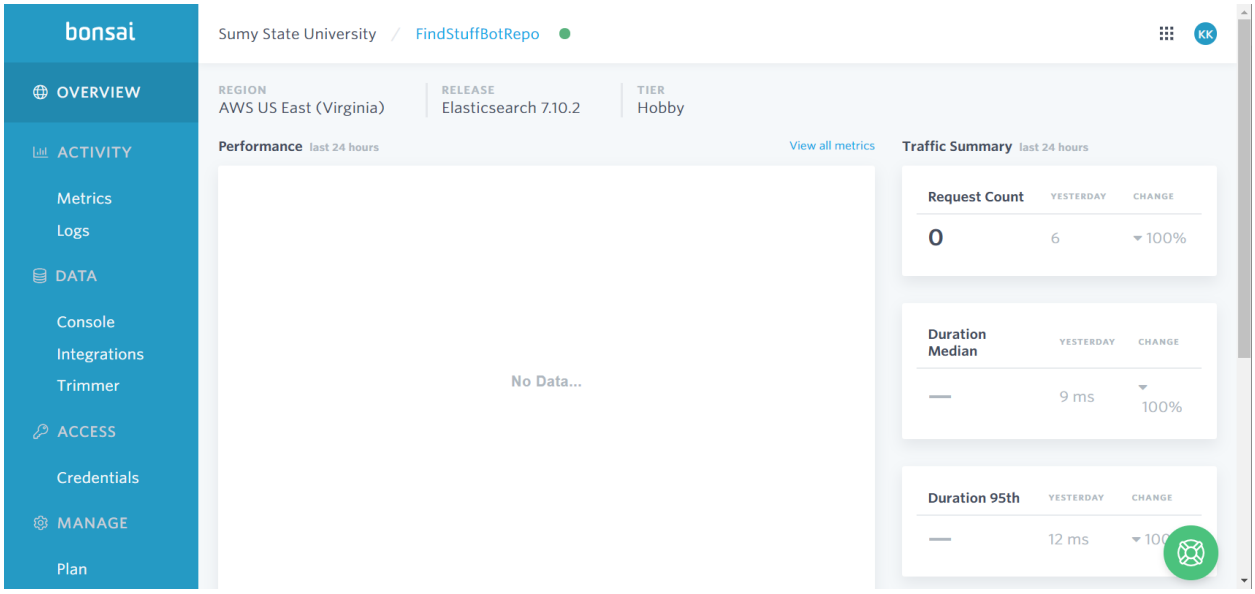

Рисунок 3.3 – Головна сторінка Bonsai з інформацією про кластер

Далі необхідно створити мапінги для індексів атрибутів таблиць нашої бази даних. Визначення зіставлення також включає поля метаданих, такі як поле _source, яке налаштовує спосіб обробки пов'язаних метаданих документа.

Розглянемо структуру проекту на рис.3.4.

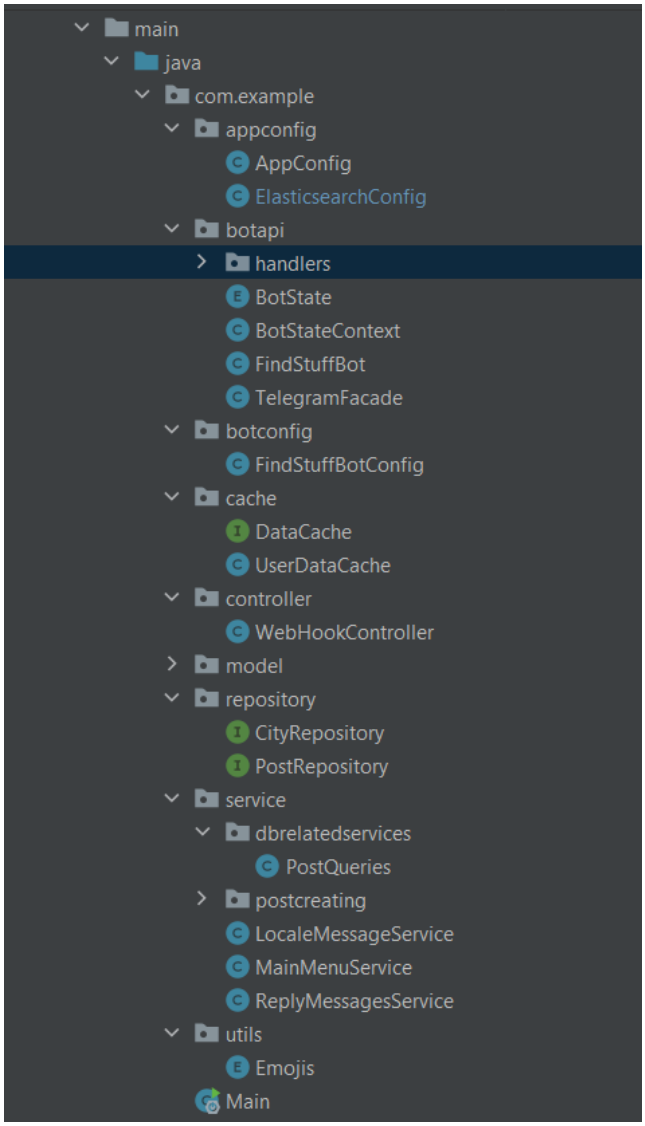

Рисунок 3.4 – Структура проекту

Розберемо детальніше кожен із підструктур проекту:

– Класи з пакету appconfg призначені для створення конфігурацій додатку та параметрів підключення до БД.

– Пакет cache містить класи й данні про взаємодію з користувачами, для того щоб бот мав змогу знати на якому етапі знаходиться взаємодія з кожним користувачем.

– Пакет repository містить інтерфейси для взаємодій з БД за допомогою бібліотеки Spring data.

– Пакет model містить класи створенні для представлення та використання сутностей із БД в програмі (міста, об’яви, закладки).

– Пакет botapi містить допоміжні класи для роботи бота та логіку обробки вхідних повідомлень. Він направляє повідомлення до потрібного класу з пакету service у якому буде звернення до БД через інтерфейси з папки repository та формування відповіді користувачу.

3.2 Використання інформаційної системи

На рис.3.5 представлено приклад створення посту, для цього використовується функція «Додати пропажу».

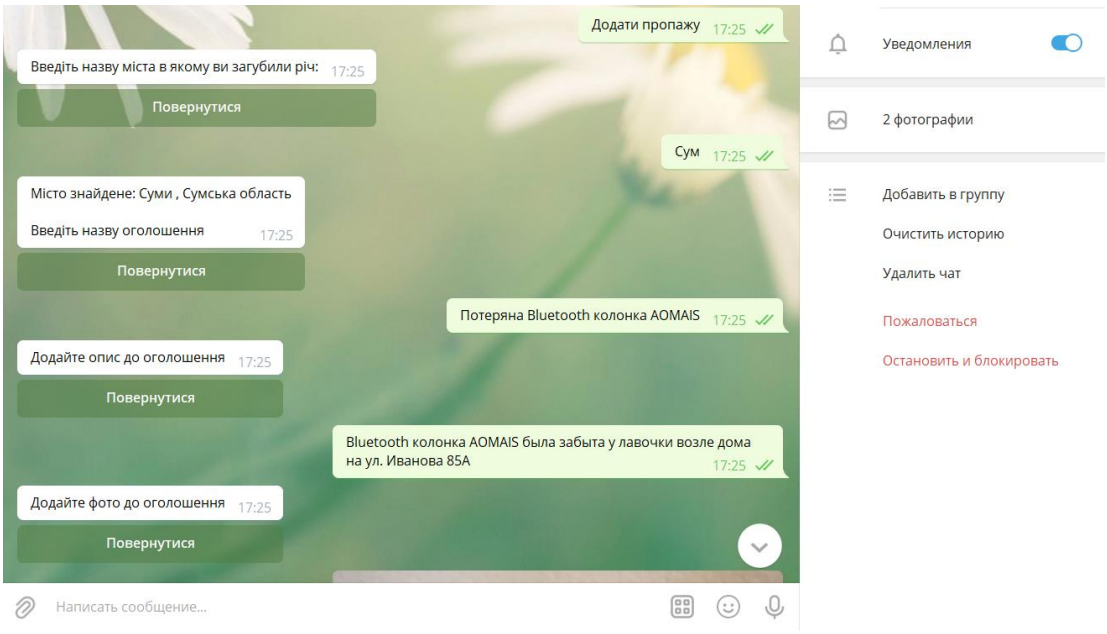

Рисунок 3.5 – Створення посту

Для цього користувач повинен заповнити певний перелік даних, а саме:

– назва міста;
– назва оголошення;
– фото пропажі;
– дата загублення речі;
– номер телефону.

Приклад заповнення за всіма атрибутами представлено на рис.3.5-6. У результаті користувач отримує сформований пост (рис.3.7).

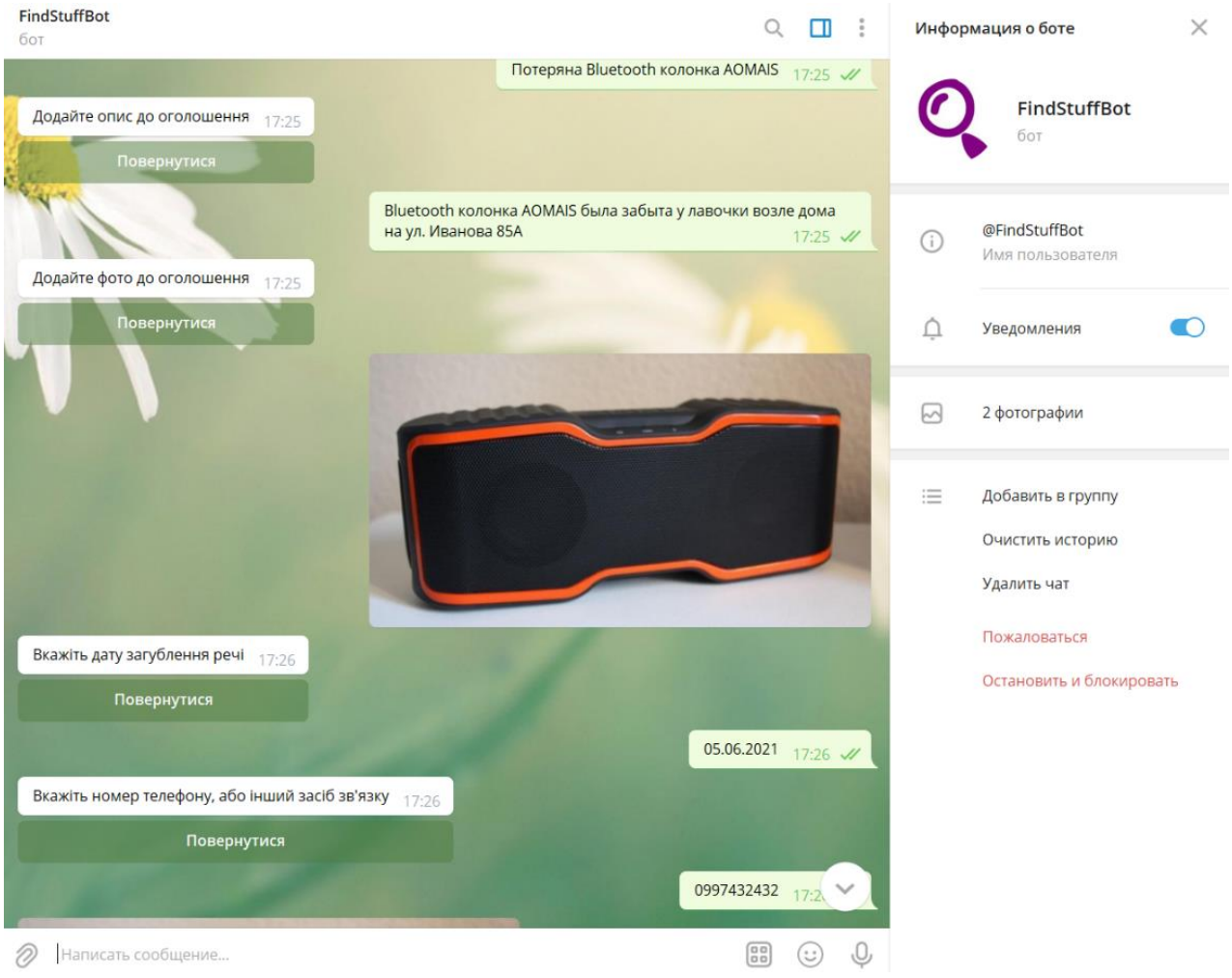

Рисунок 3.6 – Заповнення даних

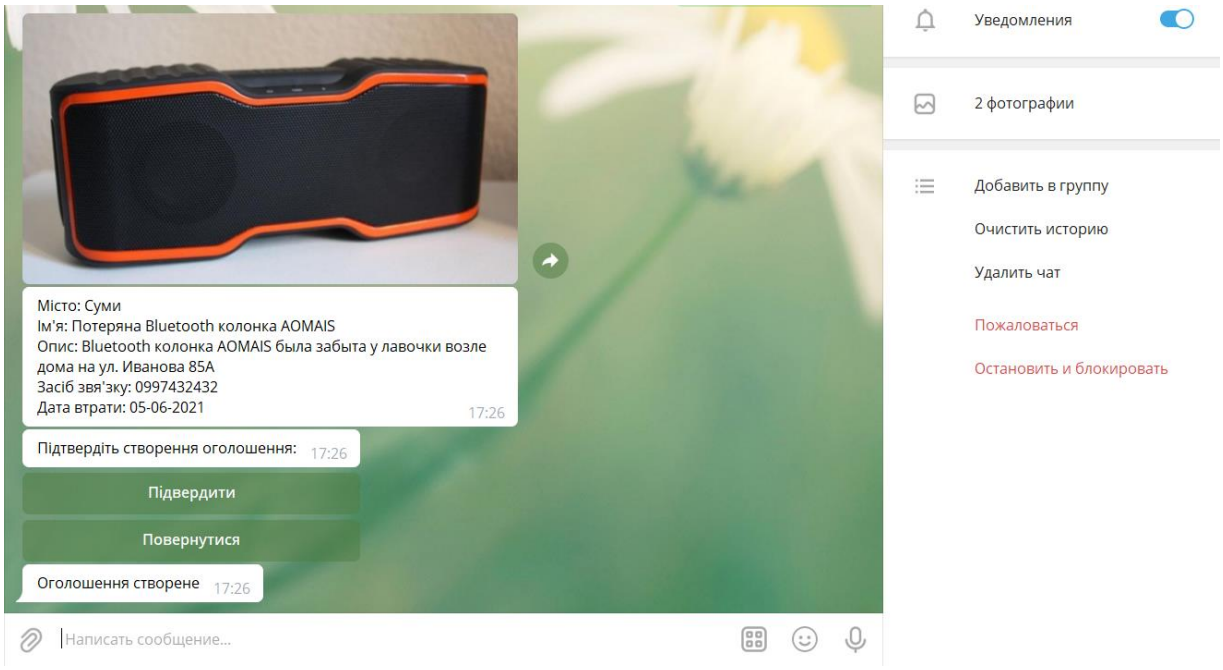

Рисунок 3.7 – Сформований пост

Також можна переглянути всі створені оголошення користувач використовуючи «Переглянути мої оголошення» (рис.3.8).

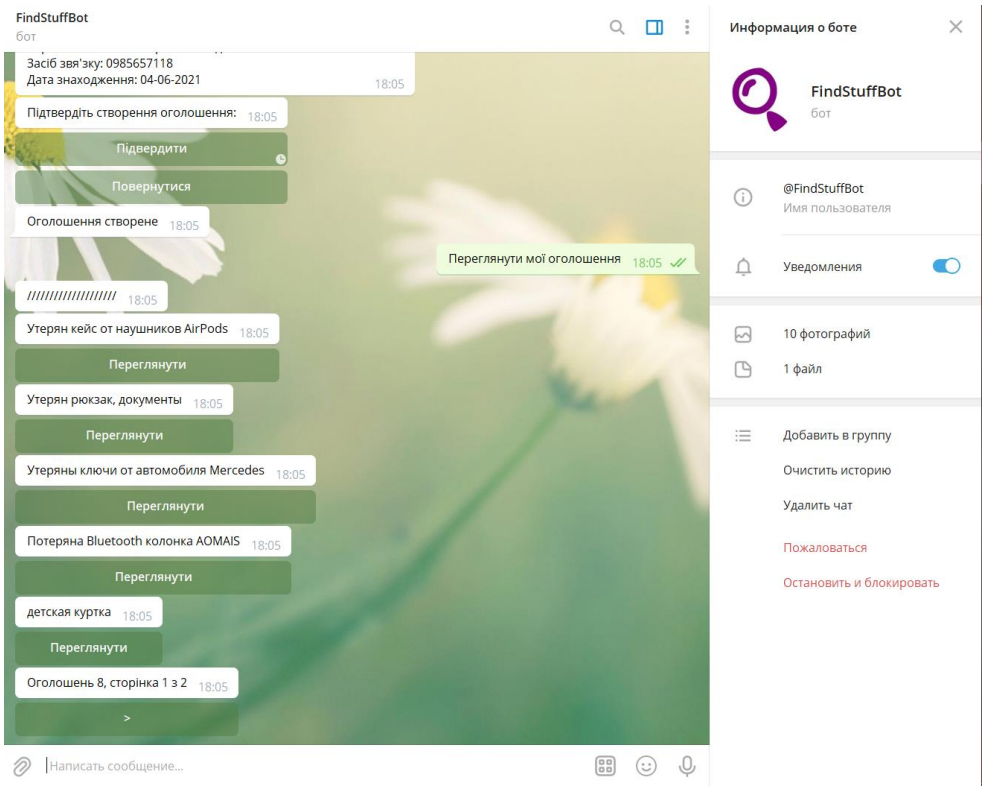

Рисунок 3.8 – Переглянути мої оголошення

Можна виконувати навігацію та переглядати список із створеними заявками про загублені речі (рис.3.9).

Відкривши власний допис користувач має можливість: додати до обраного, видалити та повернутися до початкового списку (рис.3.10-11).

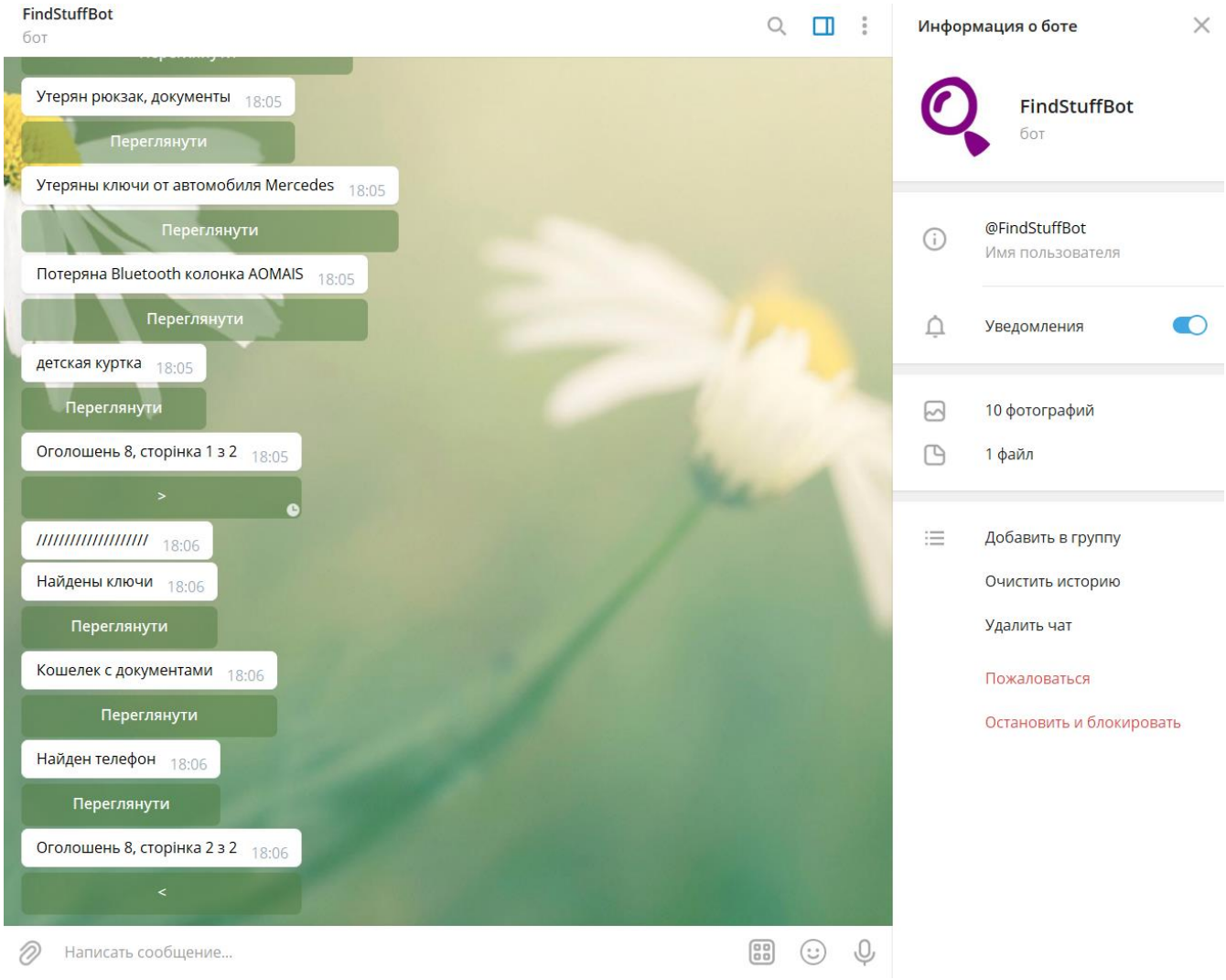

Рисунок 3.9 – Навігація у телеграм-боті

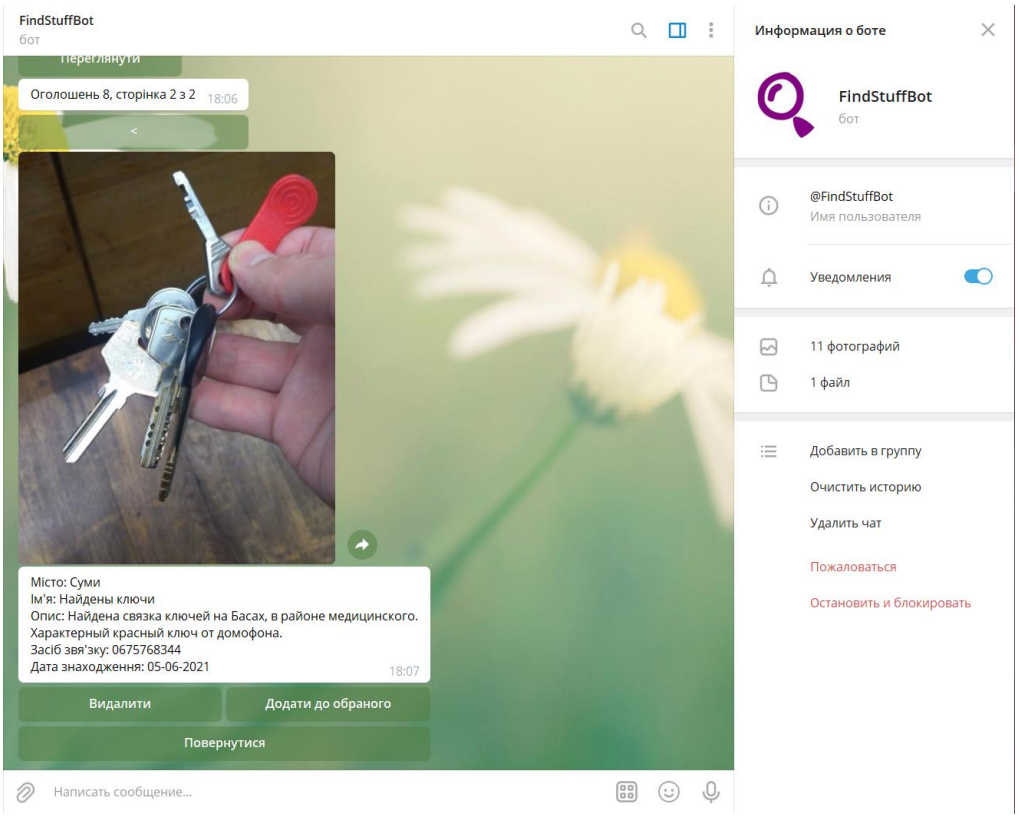

Рисунок 3.10 – Перегляд розміщеного допису

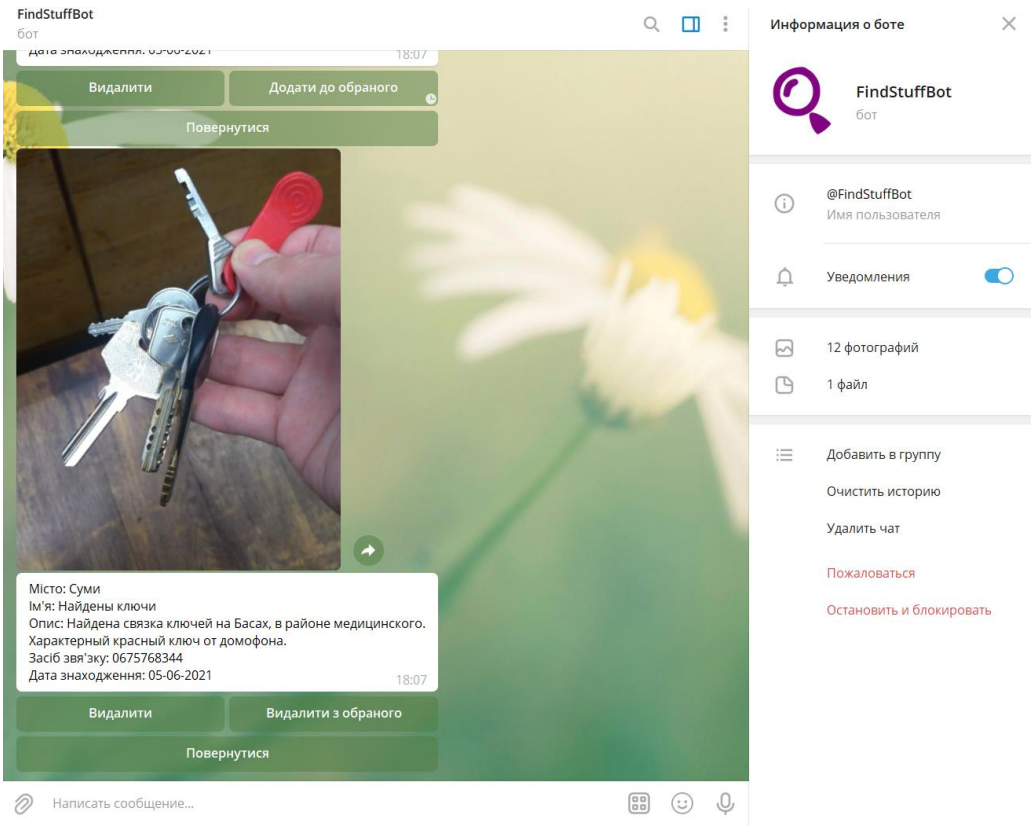

Рисунок 3.11 – Додавання до обраного

Використовуючи функцію «Обране», користувач має можливість переглянути публікації, що були раніше додані до власного списку (рис.3.12).

Крім того, як було сказано вище, можна видалити запис. Приклад видалення запису представлений на рис.3.12-14. Після натискання на кнопку «Видалити», з’являється додаткове запитання для підтвердження дії (рис.3.14).

Пошук оголошень у реалізованому телеграм-боті відбувається за допомогою введення назви міста. У прикладі на рис.3.15 був виконаний пошук за містом Суми.

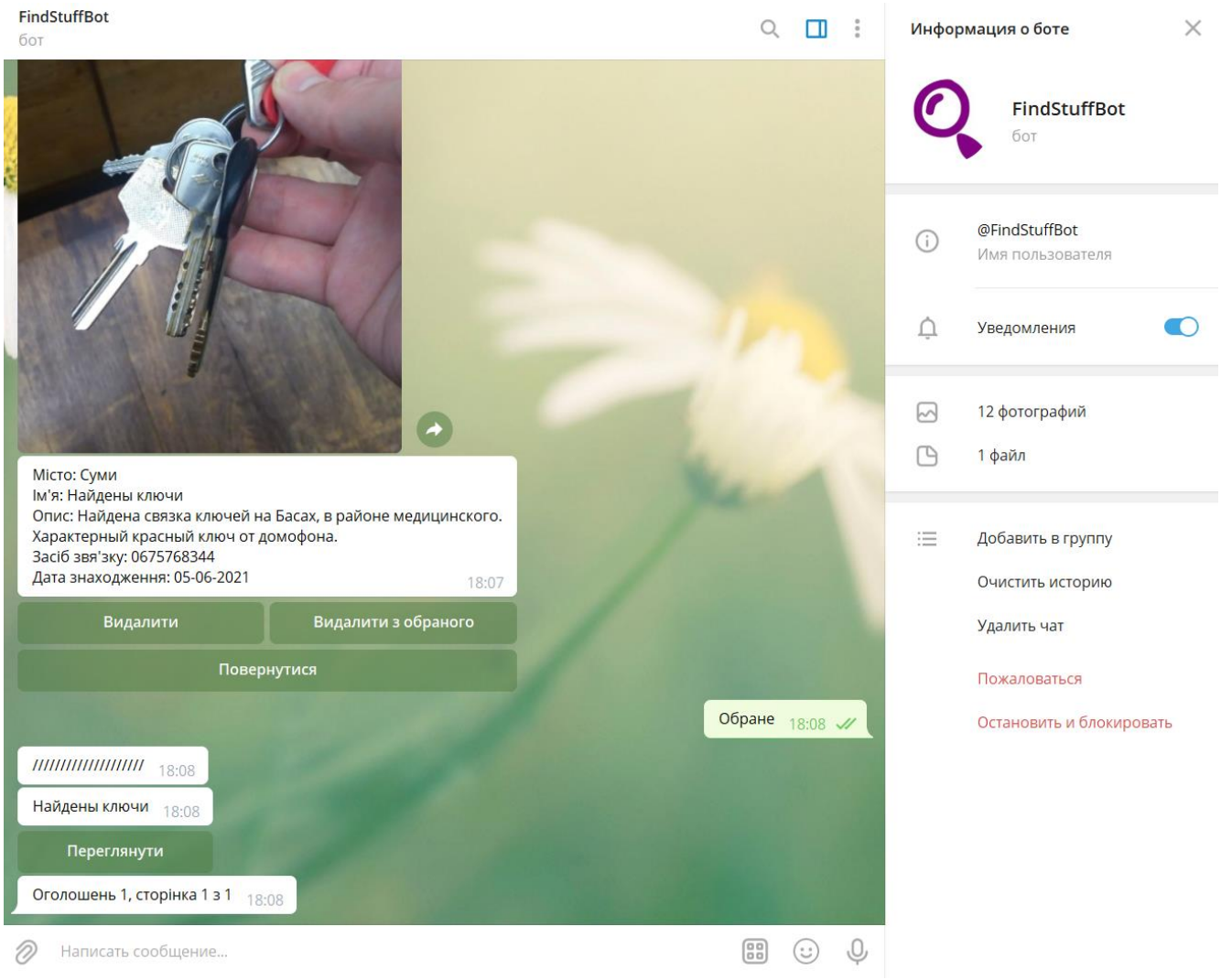

Рисунок 3.12– Переглянути обране

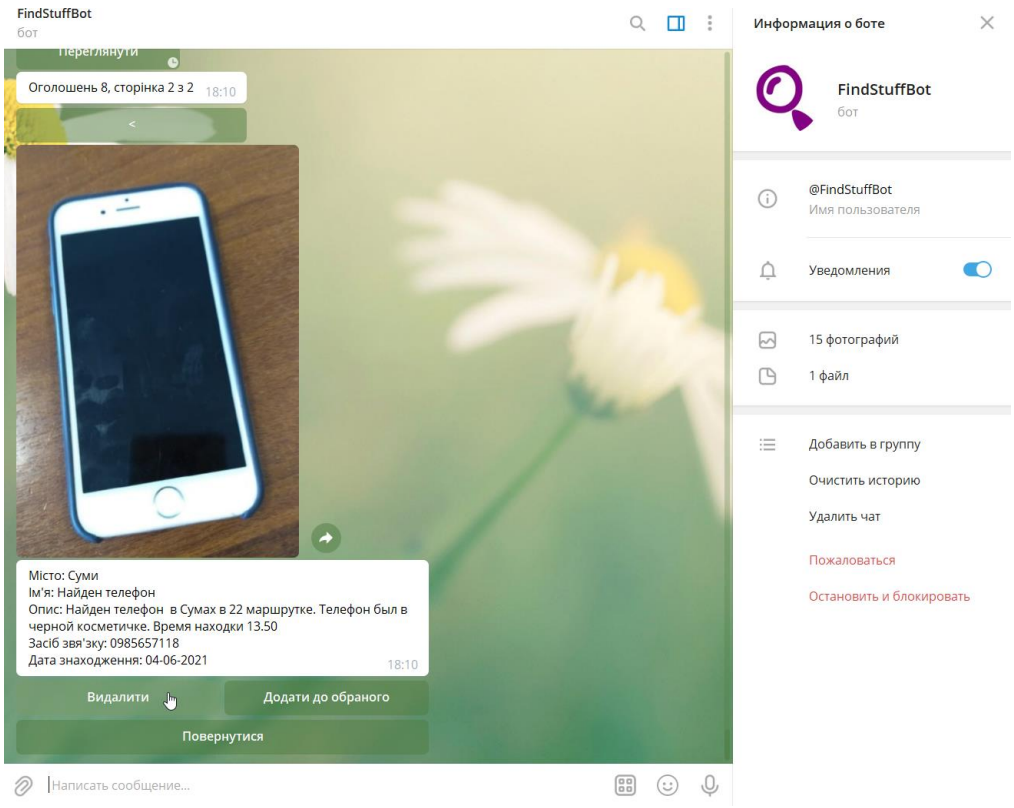

Рисунок 3.13 – Видалення оголошення

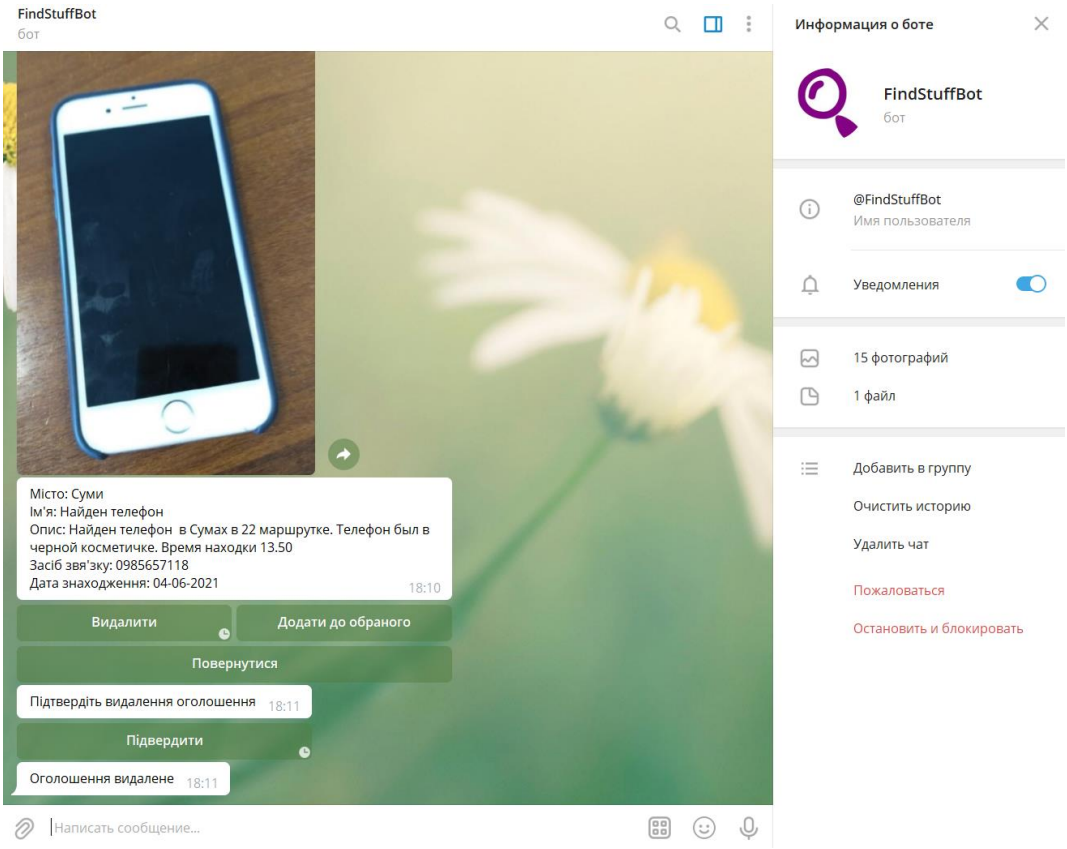

Рисунок 3.14 – Підтвердження видалення

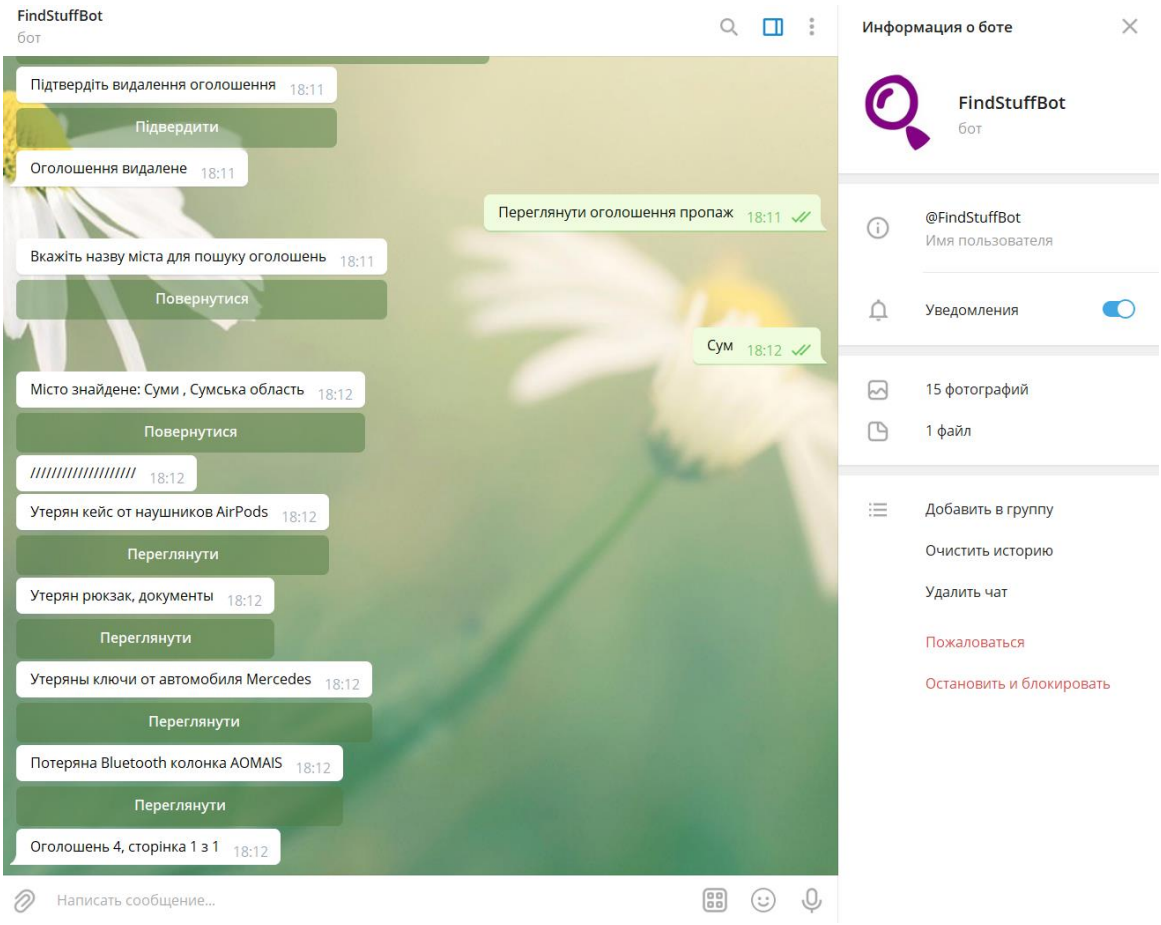

Рисунок 3.15 – Пошук оголошення

ВИСНОВКИ

В ході виконання дипломного проекту було проведено дослідження предметної області, визначено головні вимоги до системи та бізнес-процеси. Проаналізовано вимоги до системи в цілому, вимоги до функцій системи, програмного і технічного забезпечення.

Було проведено дослідження технологій для побудови телеграм-бота. В результаті дослідження були обрані наступні мови програмування та технології: Java, Spring framework, Telegram API та база даних Elasticsearch.

Тестування розробленої інформаційної системи із реальними даним стало важливим етапом при завершені реалізації.

Результатом роботи є розроблений телеграм-бот для розміщення об'яв про загублені речі. Було реалізовано весь запланований функціонал, а саме: створення оголошення, пошук за містом, перегляд інших заяв, видалення власних оголошень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 10 Problems That Ecommerce Solutions [Електронний ресурс] – 2021. – Режим доступу до ресурсу: https://acquire.io/blogs/problems-solutions-ecommerce (дата звернення: 03.05.2021);

2. Статистика крадіжок та категорій загублених речей [Електронний ресурс] – 2021 – Режим доступу: https://ukraine.segodnya.ua/ukraine/chto-i-gde-chashche-vsego-teryayut-ukraincy-nac-policiya-raskryla-statistiku-1229054.html (дата звернення: 03.05.2021);

3. Elasticsearch [Електронний ресурс] – 2021 – Режим доступу: https://uk.wikipedia.org/wiki/Elasticsearch (дата звернення: 04.05.2021);

4. Apache Kafka [Електронний ресурс] – 2021 – Режим доступу: https://ru.wikipedia.org/wiki/Apache_Kafka (дата звернення: 04.05.2021);

5. Telegram Bot API [Електронний ресурс] – 2021 – Режим доступу: https://core.telegram.org/bots/api (дата звернення: 06.05.2021);

6. REST API [Електронний ресурс]. – 2019 – Режим доступу: https://habr.com/ru/post/351890/ (дата звернення: 06.05.2021);

7. JSON[Електронний ресурс]. – Режим доступу: https://www.json.org/json-ru.html (дата звернення: 06.05.2021);

8. Elasticsearch documentation [Електронний ресурс]. – 2020 – Режим доступу: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html (дата звернення: 13.05.2021);

9. Heroku – Вікіпедія [Електронний ресурс]. – 2021 – Режим доступу: https://ru.wikipedia.org/wiki/Heroku (дата звернення: 13.05.2021);

10. Deploying Javalin on Heroku [Електронний ресурс]. – 2020 – Режим доступу: https://javalin.io/tutorials/heroku (дата звернення: 14.05.2021);

11. Bots: An introduction for developers [Електронний ресурс] – 2021 – Режим доступу: https://core.telegram.org/bots (дата звернення: 14.05.2021);

12. Telegram Bot List, 5 Examples Ideas to Boost Your Channel [Електронний ресурс] – 2020 – Режим доступу: https://www.xenioo.com/telegram-bot-list-5-ideas-for-your-channel/ (дата звернення: 15.05.2021);

13. Pros and Cons of Java [Електронний ресурс] – 2021 – Режим доступу: https://data-flair.training/blogs/pros-and-cons-of-java/

14. Python Advantages and Disadvantages – Step in the right direction [Електронний ресурс] – 2021 – Режим доступу: https://techvidvan.com/tutorials/python-advantages-and-disadvantages/ (дата звернення: 16.05.2021);

15. Python Pros and Cons [Електронний ресурс] – 2020 – Режим доступу: https://www.netguru.com/blog/python-pros-and-cons (дата звернення: 03.05.2021);

16. Java Telegram Bot Tutorial [Електронний ресурс] – 2020 – Режим доступу: https://monsterdeveloper.gitbook.io/java-telegram-bot-tutorial/ (дата звернення: 17.05.2021);

17. Learn Python Programming [Електронний ресурс] – 2021 – Режим доступу: https://www.python.org/about/gettingstarted/ (дата звернення: 18.05.2021);

18. Python For Beginners [Електронний ресурс] – 2021 – Режим доступу: https://www.programiz.com/python-programming (дата звернення: 18.05.2021);

19. Всеукраїнский сайт бюро знахідок – «Верни!» [Електронний ресурс] – 2021 – Режим доступу: verni.com.ua (дата звернення: 18.05.2021);

20. LuckFind – це online бюро знахідок [Електронний ресурс] – 2021 – Режим доступу: luckfind.me (дата звернення: 18.05.2021);

ДОДАТОК А. КОД РЕАЛІЗАЗІЇ

Main.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.telegram.telegrambots.ApiContextInitializer;

@SpringBootApplication
@EnableScheduling
public class Main {
  public static void main(String[] args) {
    ApiContextInitializer.init();
    SpringApplication.run(Main.class, args);
  }
}
```

FindStuffBot.java

```
package com.example.botapi;

import lombok.AccessLevel;
import lombok.Getter;
import lombok.Setter;
import lombok.experimental.FieldDefaults;
import lombok.extern.slf4j.Slf4j;
import org.telegram.telegrambots.bots.DefaultBotOptions;
import org.telegram.telegrambots.bots.TelegramWebhookBot;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import org.telegram.telegrambots.meta.api.methods.BotApiMethod;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import java.io.File;

@Slf4j
@Getter
```

```
@Setter
@FieldDefaults(level = AccessLevel.PRIVATE)
public class FindStuffBot extends TelegramWebhookBot {

    String botPath;
    String botUsername;
    String botToken;
    private TelegramFacade telegramFacade;
    public static FindStuffBot bot;

    public FindStuffBot(DefaultBotOptions options, TelegramFacade telegramFacade) {
        super(options);
        this.telegramFacade = telegramFacade;
        bot = this;
    }

    @Override
    public BotApiMethod<?> onWebhookUpdateReceived(Update update) {
        return telegramFacade.handleUpdate(update);
    }

    public void sendMessage(long chatId, String textMessage) {
        SendMessage sendMessage = new SendMessage();
        sendMessage.setChatId(chatId);
        sendMessage.setText(textMessage);
        try {
            execute(sendMessage);
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }

    public void sendMessage(SendMessage sendMessage) {
        try {
            execute(sendMessage);
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

TelegramFacade.java

```
package com.example.botapi;

import com.example.cache.UserDataCache;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.methods.BotApiMethod;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.Update;



@Service
@Slf4j
public class TelegramFacade {

    @Bean
    public UserDataCache getUserDataCache() {
        return userDataCache;
    }

    private final UserDataCache userDataCache;
    private final BotStateContext botStateContext;



    public TelegramFacade(UserDataCache userDataCache, BotStateContext botStateContext) {
        this.userDataCache = userDataCache;
        this.botStateContext = botStateContext;
    }

    public BotApiMethod<?> handleUpdate(Update update) {
        SendMessage replyMessage = null;

        if (update.hasCallbackQuery()) {
            log.info("New callbackQuery from User: {} with data: {}",
update.getCallbackQuery().getFrom().getUserName(),
```

```
                update.getCallbackQuery().getData());
            BotState botState = userDataCache.getUsersCurrentBotState(update.getCallbackQuery().getFrom().getId());
            return botStateContext.processCallbackQuery( botState, update.getCallbackQuery());
        }


        Message message = update.getMessage();
        if (message != null) {
            log.info("New message from User:{}, chatId: {},  with text: {}",
                message.getFrom().getUserName(), message.getChatId(), message.getText());
            replyMessage = handleInputMessage(message);
        }

        return replyMessage;
    }

    public SendMessage handleInputMessage(Message message) {
        String inputMsg = message.getText();
        int userId = message.getFrom().getId();
        BotState botState;
        SendMessage replyMessage;
        if (inputMsg!=null){
            switch (inputMsg) {
                case "Додати пропажу":
                    botState = BotState.CREATE_LOSS_POST;
                    break;
                case "Додати знахідку":
                    botState = BotState.CREATE_GODSEND_POST;
                    break;
                case "Переглянути оголошення знахідок":
                    botState = BotState.SEARCH_GODSEND_POSTS;
                    break;
                case "Переглянути оголошення пропаж":
                    botState = BotState.SEARCH_LOSS_POSTS;
                    break;
                case "Переглянути мої оголошення":
                    botState = BotState.SEARCH_MY_POSTS;
                    break;
                case "Обране":
```

```
                botState = BotState.CHECK_BOOKMARKS;
                break;
            default:
                botState = userDataCache.getUsersCurrentBotState(userId);
                break;
        }
    } else {
        botState = userDataCache.getUsersCurrentBotState(userId);
    }

    userDataCache.setUsersCurrentBotState(userId, botState);
    replyMessage = botStateContext.processInputMessage(botState, message);
    return replyMessage;
  }


}
```

BotState.java

```
package com.example.botapi;

public enum BotState {
   SHOW_MAIN_MENU,
   CREATE_LOSS_POST,
   CREATE_GODSEND_POST,
   SEARCH_LOSS_POSTS,
   SEARCH_GODSEND_POSTS,
   SEARCH_MY_POSTS,
   CHECK_BOOKMARKS
}
```

Bookmark.java package com.example;

```
package com.example.model;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.annotation.Id;
import org.springframework.data.elasticsearch.annotations.Document;
```

```
import java.util.List;

@Getter
@Setter
@Document(indexName = "bookmarks")
public class Bookmark {

    @Id
    private String userId;

    private List<String> postIDs;

    public Bookmark(String userId, List<String> postIDs) {
        this.userId = userId;
        this.postIDs = postIDs;
    }


}
```

City.java

```
package com.example.model;

import lombok.Getter;
import lombok.Setter;
import org.springframework.data.annotation.Id;
import org.springframework.data.elasticsearch.annotations.Document;

@Setter
@Getter
@Document(indexName = "cities")
public class City {
    @Id
    private String id;

    private String city;

    private String region;
```

```
    public City(String id, String city, String region) {
        this.id = id;
        this.city = city;
        this.region = region;
    }

}
```

Post.java

```
package com.example.model;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;
import org.springframework.data.annotation.Id;
import org.springframework.data.elasticsearch.annotations.Document;

import java.util.Date;
import java.util.UUID;

@ToString
@Getter
@Setter
@Document(indexName = "posts")
public class Post {

    @Id
    String id;
    PostType postType;
    Date postDate;
    String senderId;

    String name;
    String city;
    String description;
    String image;
    Date foundDate;
    String contactMethod;
```

```
    public Post(){}

    public Post(String id,String name, String city, PostType postType, String description, String image, Date postDate, Date foundDate, String contactMethod, String senderId) {
        this.id = id;
        this.name = name;
        this.city = city;
        this.postType = postType;
        this.description = description;
        this.image = image;
        this.postDate = postDate;
        this.foundDate = foundDate;
        this.contactMethod = contactMethod;
        this.senderId = senderId;
    }
}
```

PostType.java

```
package com.example.model;

public enum PostType {
    LOSS,
    GODSEND
}
```

Main.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;
```

Main.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;
```

MainMenuService.java

```
package com.example.service;

import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;

import java.util.ArrayList;
import java.util.List;

@Service
public class MainMenuService {

    private ReplyMessagesService messagesService;

    private MainMenuService(ReplyMessagesService messagesService){
        this.messagesService = messagesService;
    }

    public SendMessage getMainMenuMessage(final long chatId, final String textMessage) {
        final ReplyKeyboardMarkup replyKeyboardMarkup = getMainMenuKeyboard();

        return createMessageWithKeyboard(chatId, textMessage, replyKeyboardMarkup);
    }

    private ReplyKeyboardMarkup getMainMenuKeyboard() {

        final ReplyKeyboardMarkup replyKeyboardMarkup = new ReplyKeyboardMarkup();
        replyKeyboardMarkup.setSelective(true);
        replyKeyboardMarkup.setResizeKeyboard(true);
        replyKeyboardMarkup.setOneTimeKeyboard(true);

        List<KeyboardRow> keyboard = new ArrayList<>();

        KeyboardRow row1 = new KeyboardRow();
```

```
        KeyboardRow row2 = new KeyboardRow();
        KeyboardRow row3 = new KeyboardRow();
        KeyboardRow row4 = new KeyboardRow();
        KeyboardRow row5 = new KeyboardRow();
        KeyboardRow row6 = new KeyboardRow();
        row1.add(new KeyboardButton(messagesService.getReplyText("buttons.mainMenu.addLostPost")));
        row2.add(new KeyboardButton(messagesService.getReplyText("buttons.mainMenu.addGodsendPost")));
        row3.add(new KeyboardButton(messagesService.getReplyText("buttons.mainMenu.searchLostPosts")));
        row4.add(new
KeyboardButton(messagesService.getReplyText("buttons.mainMenu.searchGodSendPosts")));
        row3.add(new KeyboardButton(messagesService.getReplyText("buttons.mainMenu.myPosts")));
        row4.add(new KeyboardButton(messagesService.getReplyText("buttons.mainMenu.bookmarks")));
        keyboard.add(row1);
        keyboard.add(row2);
        keyboard.add(row3);
        keyboard.add(row4);
        keyboard.add(row5);
        keyboard.add(row6);
        replyKeyboardMarkup.setKeyboard(keyboard);
        return replyKeyboardMarkup;
    }

    private SendMessage createMessageWithKeyboard(final long chatId,
                                                  String textMessage,
                                                  final ReplyKeyboardMarkup replyKeyboardMarkup) {
        final SendMessage sendMessage = new SendMessage();
        sendMessage.enableMarkdown(true);
        sendMessage.setChatId(chatId);
        sendMessage.setText(textMessage);
        if (replyKeyboardMarkup != null) {
            sendMessage.setReplyMarkup(replyKeyboardMarkup);
        }
        return sendMessage;
    }
}
```

LocaleMessageService.java

```
package com.example.service;

import org.springframework.beans.factory.annotation.Value;
```

```
import org.springframework.context.MessageSource;
import org.springframework.stereotype.Service;

import java.util.Locale;

@Service
public class LocaleMessageService {
    private final Locale locale;
    private final MessageSource messageSource;

    public LocaleMessageService(@Value("${localeTag}") String localeTag, MessageSource messageSource) {
        this.messageSource = messageSource;
        this.locale = Locale.forLanguageTag(localeTag);
    }


    public String getMessage(String message) {
        return messageSource.getMessage(message, null, locale);
    }

    public String getMessage(String message, Object... args) {
        return messageSource.getMessage(message, args, locale);
    }


}
```

bookmarksOperations/Bookmarks.java

```
package com.example.service.bookmarksOperations;

import com.example.model.Bookmark;
import com.example.repository.BookmarkQueries;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class Bookmarks {

    BookmarkQueries bookmarkQueries;
```

```
private Bookmarks(BookmarkQueries bookmarkQueries){
   this.bookmarkQueries = bookmarkQueries;
}

public void addBookmark(String userId,String postId){
   Bookmark bookmark = bookmarkQueries.getBookmark(userId);
   if(bookmark == null){
      List<String> ids = new ArrayList<>();
      ids.add(postId);
      bookmark = new Bookmark(userId,ids);
      bookmarkQueries.addBookmark(bookmark);
   } else {
      List<String> ids = bookmark.getPostIDs();

      if (ids.contains(postId)) return;
      ids.add(postId);
      bookmark.setPostIDs(ids);
      bookmarkQueries.addBookmark(bookmark);
   }
}

public void deleteBookmark(String userId,String postId){
   Bookmark bookmark = bookmarkQueries.getBookmark(userId);
   if(bookmark == null) return;
   List<String> ids = bookmark.getPostIDs();
   if(!ids.contains(postId)) return;
   ids.remove(postId);
   bookmark.setPostIDs(ids);
   bookmarkQueries.addBookmark(bookmark);

}

public Bookmark getBookmark(String userId){
   return bookmarkQueries.getBookmark(userId);
}

public boolean isAddedToBookmarks(String userId,String postId){
   Bookmark bookmark = bookmarkQueries.getBookmark(userId);
   if (bookmark == null) return false;
```

```
        List<String> ids = bookmark.getPostIDs();
        return ids.contains(postId);
    }

}
```

PostBuilderService.java

```
package com.example.service.postcreating;

import com.example.botapi.BotState;
import com.example.botapi.FindStuffBot;
import com.example.botapi.handlers.InputMessageHandler;
import com.example.cache.UserDataCache;
import com.example.model.City;
import com.example.model.PostType;
import com.example.service.ReplyMessagesService;
import com.example.repository.CityQueries;
import com.example.repository.PostQueries;
import com.example.service.postpresntation.PostSender;
import org.apache.commons.io.FileUtils;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.methods.GetFile;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.CallbackQuery;
import org.telegram.telegrambots.meta.api.objects.Message;
import org.telegram.telegrambots.meta.api.objects.PhotoSize;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import java.io.File;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

@Service
public class PostBuilderService {

    PostQueries postQueries;
    ReplyMessagesService messagesService;
```

```
    CityQueries cityQueries;
    PostSender postSender;

    private final Map<BotState, InputMessageHandler> messageHandlers = new HashMap<>();

    private PostBuilderService(PostQueries postQueries, ReplyMessagesService messagesService, CityQueries cityQueries, PostSender postSender){
        this.postQueries = postQueries;
        this.messagesService = messagesService;
        this.cityQueries = cityQueries;
        this.postSender = postSender;
    }

    public SendMessage handleCallbackQuery(CallbackQuery callbackQuery, PostCache postCache, UserDataCache userDataCache){
        Message message = callbackQuery.getMessage();
        if(callbackQuery.getData().equals(messagesService.getReplyText("buttons.postCreating.back"))){
            PostCreatingStage currentStage = postCache.getCurrentStage();
            if( currentStage == PostCreatingStage.START_CREATING ){
                userDataCache.deletePostCache(message.getFrom().getId(),postCache);
                userDataCache.setUsersCurrentBotState(message.getFrom().getId(), BotState.SHOW_MAIN_MENU);
                InputMessageHandler messageHandler = messageHandlers.get(BotState.SHOW_MAIN_MENU);
                return messageHandler.handle(message);

            } else {
                postCache.previousStage();
                userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);
                return getRepliedText(message,postCache,userDataCache);
            }

        } else if(callbackQuery.getData().equals(messagesService.getReplyText("buttons.postCreating.confirm")) && postCache.getCurrentStage() == PostCreatingStage.CONFIRM_CREATION) {
            SendMessage result = new SendMessage(message.getChatId(), messagesService.getReplyText("reply.createPost.created"));
            postQueries.SavePost(postCache.cashedPost);
            userDataCache.deletePostCache(callbackQuery.getFrom().getId(),postCache);
            userDataCache.setUsersCurrentBotState(message.getFrom().getId(), BotState.SHOW_MAIN_MENU);
            return result;
        } else {
            return new SendMessage(message.getChatId(),"");
```

```
        }
    }

    public SendMessage getRepliedText(Message message, PostCache postCache, UserDataCache userDataCache){
        Long chatId = message.getChatId();
        SendMessage result;


        switch (postCache.getCurrentStage()){
            case START_CREATING:
                result = new SendMessage(chatId, postCache.cashedPost.getPostType().equals(PostType.LOSS)
                        ? messagesService.getReplyText("reply.createLostPost.askCity") :
messagesService.getReplyText("reply.createGodsendPost.askCity"));
                postCache.nextStage();
                userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);
                result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                break;
            case ASK_CITY:
                String cityName = message.getText();
                if(cityName == null || cityName.length() < 3) {
                    result = new SendMessage(chatId,
messagesService.getReplyText("reply.createPost.cityMinLengthValidation"));
                    result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                    break;
                }

                List<City> cities = cityQueries.searchCity(cityName);

                if(cities.size() == 0){
                    result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.cityNotFound"));
                } else if(cities.size() == 1) {
                    City city = cities.get(0);
                    String replyCityName = city.getCity() + " , " + city.getRegion();
                    result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.cityFound") + " "
+ replyCityName + "\n\n" +
                            messagesService.getReplyText("reply.createPost.askName"));
                    result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                    postCache.cashedPost.setCity(city.getCity());
                    postCache.nextStage();
                    userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);
```

```
                    result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                } else {
                    result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.fewCities"));
                    result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                }
                break;
            case ASK_NAME:
                String postName = message.getText();
                if(postName == null || postName.length() < 5) {
                    result = new SendMessage(chatId,
messagesService.getReplyText("reply.createPost.postNameMinLengthValidation"));
                    result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                    break;
                }

                result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.askDescription"));
                postCache.cashedPost.setName(postName);
                postCache.nextStage();
                userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);

                result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                break;
            case ASK_IMAGE:
                if(message.hasPhoto()){
                    List<PhotoSize> photos = message.getPhoto();
                    PhotoSize photo = photos.stream().max(Comparator.comparing(PhotoSize::getFileSize)).get();
                    File file;
                    try {
                        GetFile getFile = new GetFile().setFileId(photo.getFileId());
                        String filePath = FindStuffBot.bot.execute(getFile).getFilePath();
                        file = FindStuffBot.bot.downloadFile(filePath);
                    } catch (TelegramApiException e) {
                        result = new SendMessage(chatId,
messagesService.getReplyText("reply.createPost.photoDownloadException"));
                        result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                        e.printStackTrace();
                        break;
                    }

                    byte[] fileContent;
```

```
                try {
                    fileContent = FileUtils.readFileToByteArray(file);
                } catch (IOException e) {
                    result = new SendMessage(chatId, "File convertation Error");
                    result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                    e.printStackTrace();
                    break;
                }
                String encodedString = Base64.getEncoder().encodeToString(fileContent);

                postCache.cashedPost.setImage(encodedString);
                result = new SendMessage(chatId, postCache.cashedPost.getPostType().equals(PostType.LOSS)
                        ? messagesService.getReplyText("reply.createLostPost.askFoundDate") :
messagesService.getReplyText("reply.createGodsendPost.askFoundDate"));
                postCache.nextStage();
                userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);
            } else {
                result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.validatePhoto"));
            }
            result.setReplyMarkup(postSender.getBackButtonForPostCreating());

            break;
        case ASK_DESCRIPTION:
            String postDescription = message.getText();
            if( postDescription == null || postDescription.length() < 5) {
                result = new SendMessage(chatId,
messagesService.getReplyText("reply.createPost.descriptionMinLengthValidation"));
                result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                break;
            }

            result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.askImage"));

            postCache.cashedPost.setDescription(postDescription);
            postCache.nextStage();
            userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);

            InlineKeyboardMarkup replyKeyboardMarkup = postSender.getBackButtonForPostCreating();
            result.setReplyMarkup(replyKeyboardMarkup);
            break;
```

```
                case ASK_FOUND_DATE:
                    result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.askContactMethod"));

                    Date dateDepart;
                    try {
                        dateDepart = new SimpleDateFormat("dd.MM.yyyy").parse(message.getText());
                    } catch (ParseException e) {
                        return messagesService.getWarningReplyMessage(chatId, "reply.createPost.incorrectDateFormat");
                    }

                    postCache.cashedPost.setFoundDate(dateDepart);
                    postCache.nextStage();
                    userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);

                    result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                    break;
                case ASK_CONTACT_METHOD:
                    String contactMethod = message.getText();

                    if(contactMethod == null) {
                        result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.contactMethodNullValidation"));
                        result.setReplyMarkup(postSender.getBackButtonForPostCreating());
                        break;
                    }

                    postCache.cashedPost.setContactMethod(contactMethod);

                    try {
                        postSender.sendPostWithoutButtons(chatId , postCache.cashedPost);
                    } catch (IOException e) {
                        result = new SendMessage(chatId,"image convertation error");
                        e.printStackTrace();
                        break;
                    } catch (TelegramApiException e) {
                        result = new SendMessage(chatId,"send image error");
                        e.printStackTrace();
                        break;
                    }
```

```
                result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.askConfirmation"));
                postCache.nextStage();
                InlineKeyboardMarkup inlineKeyboardMarkup = postSender.getBackButtonForPostCreating();
                List<List<InlineKeyboardButton>> keyboard = inlineKeyboardMarkup.getKeyboard();
                keyboard.add(keyboard.get(0));

                InlineKeyboardButton backButton = new InlineKeyboardButton().setText(messagesService.getReplyText("buttons.postCreating.confirm"));
                backButton.setCallbackData(messagesService.getReplyText("buttons.postCreating.confirm"));
                List<InlineKeyboardButton> keyboardRow = new ArrayList<>();
                keyboardRow.add(backButton);
                keyboard.set(0,keyboardRow);
                inlineKeyboardMarkup.setKeyboard(keyboard);

                userDataCache.setUsersPostCache(message.getFrom().getId(),postCache);
                result.setReplyMarkup(inlineKeyboardMarkup);
                break;
            case CONFIRM_CREATION:
                result = new SendMessage(chatId, messagesService.getReplyText("reply.createPost.reAskConfirmation"));
                break;
            default:
                result = new SendMessage(chatId, "error");
                break;
        }

        return result;
    }



}
```

WebHookController.java

```
package com.example.controller;


import com.example.botapi.FindStuffBot;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.*;
import org.telegram.telegrambots.meta.api.methods.BotApiMethod;
```

```
import org.telegram.telegrambots.meta.api.objects.Update;


@Slf4j
@RestController
public class WebHookController {
    private final FindStuffBot telegramBot;

    public WebHookController(FindStuffBot telegramBot) {
        this.telegramBot = telegramBot;
    }

    @RequestMapping(value = "/", method = RequestMethod.POST)
    public BotApiMethod<?> onUpdateReceived(@RequestBody Update update) {
        return telegramBot.onWebhookUpdateReceived(update);
    }

}
```

CreateGodsendPostHandler.java

```
package com.example.botapi.handlers.menu;

import com.example.botapi.BotState;
import com.example.botapi.handlers.InputMessageHandler;
import com.example.cache.UserDataCache;
import com.example.service.postcreating.PostBuilderService;
import com.example.service.postcreating.PostCache;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.Message;

@Component
public class CreateGodsendPostHandler implements InputMessageHandler {

    private UserDataCache userDataCache;
    private PostBuilderService postBuilderService;

    public CreateGodsendPostHandler(UserDataCache userDataCache,PostBuilderService postBuilderService) {
        this.userDataCache = userDataCache;
        this.postBuilderService = postBuilderService;
    }
```

```
    @Override
    public SendMessage handle(Message message) {
        PostCache postCache = userDataCache.getUsersGodsendPostCache(message.getFrom().getId());
        return postBuilderService.getRepliedText(message, postCache , userDataCache);
    }

    @Override
    public BotState getHandlerName() {
        return BotState.CREATE_GODSEND_POST;
    }

}
```

CreateGodsendPostCallbackHandler.java

```
package com.example.botapi.handlers.callbackquery;

import com.example.botapi.BotState;
import com.example.cache.UserDataCache;
import com.example.service.postcreating.PostBuilderService;
import com.example.service.postcreating.PostCache;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.objects.CallbackQuery;

@Component
public class CreateGodsendPostCallbackHandler implements CallbackQueryHandler{

    private UserDataCache userDataCache;
    private PostBuilderService postBuilderService;

    public CreateGodsendPostCallbackHandler(UserDataCache userDataCache,PostBuilderService
postBuilderService) {
        this.userDataCache = userDataCache;
        this.postBuilderService = postBuilderService;
    }

    @Override
    public SendMessage handleCallbackQuery(CallbackQuery callbackQuery) {
        PostCache postCache = userDataCache.getUsersGodsendPostCache(callbackQuery.getFrom().getId());
        return postBuilderService.handleCallbackQuery(callbackQuery, postCache, userDataCache);
```

```
    }

    @Override
    public BotState getHandlerName() {
        return BotState.CREATE_GODSEND_POST;
    }
}
```