МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

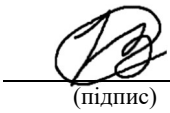
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна онлайн-система моніторингу гуманітарних потреб населення»
здобувача групи ІН – 91 Ільченка Єгора Валентиновича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.


_____ Єгор ІЛЬЧЕНКО
(підпис)

Керівник,
старший викладач,
кандидат технічних наук Борис КУЗІКОВ _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-91 Ільченка Єгора Валентиновича

1. Тема роботи: «Інформаційна технологія прогнозування курсу валют»
затверджую наказом по СумДУ від _____

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що можуть бути використанні при створенні системи моніторингу. 3) Розробка інформаційної онлайн-системи моніторингу гуманітарних потреб. 4) Демонстрація і аналіз результатів розробки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

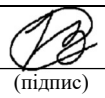
7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз предметної області, порівнювання і виявлення недоліків сучасних інструментів, пов'язаних з гуманітарною допомогою.*	25.04.23	
2	*Аналіз і вибір технологій та підходів, які будуть доцільними*	03.05.23	

	у розробці інформаційної системи моніторингу гуманітарних потреб населення.		
3	*Розробка інформаційної системи моніторингу гуманітарних потреб населення*	27.05.23	
4	*Демонстрація і перевірка працездатності розробленої системи.*	29.05.23	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	04.06.23	

Здобувач вищої освіти  Керівник

(підпис) (підпис)

АНОТАЦІЯ

Записка: 73 стр., 22 рис., 1 таблиця, 1 додаток, 21 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи актуальна, оскільки присвячена вирішення вкрай важливої проблеми насьогодні – відслідковуванні потреб населення у гуманітарних потребах. У результаті аналізу наявних рішень було виявлено унікальність розроблюваної системи.

Об’єкт дослідження — процес відслідковування гуманітарних потреб населення.

Мета роботи — розробка інформаційної системи моніторингу гуманітарних потреб з зрозумілим інтерфейсом і простою системою опитувань.

Методи дослідження — інструменти побудови веб інтерфейсів, системи управління базами даних, алгоритми вибору декількох центрів для скупчення географічних точок.

Результати — розроблено інформаційну систему, яка дозволяє проводити опитування населення за допомогою месенджеру Telegram з метою проведення моніторингу гуманітарних потреб населення у певному населеному пункті. В результаті опитування результати будуть відображені на карті з можливістю вибору точок розташування гуманітарних вантажів.

ІНФОРМАЦІЙНА СИСТЕМА, МОНІТОРИНГ ПОТРЕБ НАСЕЛЕННЯ, JAVA, PYTHON, REACTJS, SPRING, AIOGRAM.

ЗМІСТ

ВСТУП

Насьогодні інформаційні технології використовуються для різних цілей: у виробництві і транспорті, в банках та на біржах, в ЗМІ і видавництвах, для оборонних систем, соціальних та правоохоронних галузях, для підтримання апаратів у сфері охорони здоров'я, у навчальних процесах, нарешті, щоб усунути перешкоди для вільного спілкування і взаємодії на відстані – створення мережі інтернет.[1] Але єдине що об’єднує усі вищенаведені приклади використання – це, в першу чергу, допомога людям з автоматизацією тих процесів, які раніше були рутинними і займали багато часу. І неважливо був цей час використаний на оформлення, розрахунки або ж банальне подолання відстані між точкою А до точки Б.

Зараз, в досить складних умовах, будь-яке рішення, яке може покращити життя людей шляхом спрощення якихось базових процедур - вкрай важливе. А тому розробка інформаційної онлайн-системи моніторингу гуманітарних потреб населення є досить актуальною у сучасних реаліях.

Метою роботи є створення інформаційної системи, що удосконалює методи моніторингу гуманітарної допомоги та задовольняє наступні потреби: може зберігати інформацію про потреби населення в гуманітарній допомозі і відображати результати на карті, зможе приймати і обробляти запити від користувачів та відправляти сповіщення користувачам про найближчий пункт видачі гуманітарних потреб. А також майбутня система повинна зрозумілий і більш-менш зручний інтерфейс для того, хто спостерігає за ситуацією у населених пунктах.

Це, як мінімум, для більшості може скоротити відстань до пунктів видачі, а також допоможе оцінити ситуацію в населеному пункті, а саме скільком людям необхідне додаткове забезпечення продуктами харчування в певний часовий проміжок.

Додаток буде створений як прототип системи з усім необхідним функціоналом, але який може бути в майбутньому покращений і введений у дію.

Для досягнення поставленої мети дослідження сформульовані наступні задачі :

- Аналіз наявних рішень за тематикою роботи.
- Аналіз методики вирішення задач: найбільш вдалі для використання технології, алгоритми, тощо.
- Опис архітектури системи,
- Реалізація серверної частини з використанням Spring framework,
- Розробка Telegram боту, використовуючи бібліотеку aiogram
- Розробка клієнтської частини з використанням бібліотеки React JS та Redux.
- Демонстрація роботи реалізованої системи

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ТЕМАТИКОЮ РОБОТИ

Варто проаналізувати наявні інструменти для спостеріганням за потребами гуманітарної допомоги. Аналіз складається з наступних кроків: підтвердження актуальності проблеми, яку вирішує інформаційна система, аналіз подібних веб-систем, спроби знайти спільний функціонал, аналізуючи Telegram ботів із схожими задачами.

1.1. Підтвердження доцільності розробки системи

Основними завданнями гуманітарної допомоги є:

- порятунок людського життя;
- надання допомоги для задоволення основних потреб людини
- забезпечення базової гігієни та медичної допомоги [2]

Додаток націлений на спрощення механізму надання двох останніх видів допомоги.

Згідно з доповіддю Координаційного штабу від 16.04.2022 року, дев'ять областей та місто Київ отримали продуктові набори:

- Харківська обл. - 1 931 845 наб.
- м. Київ - 1 420 000 наб.
- Запорізька обл. - 950 000 наб.
- Херсонська обл. - 935 000 наб.
- Чернігівська обл. - 850 000 наб.
- Донецька обл. - 706 760 наб.
- Сумська обл. - 528 000 наб.
- Київська обл. - 500 000 наб.
- Миколаївська обл. - 400 000 наб.
- Луганська обл. - 100 000 наб.
- Всього - 8 321 605 наб.

Як видно, обсяги цієї допомоги досить значні, було б доречно, особливо, якщо наборів за один раз багато – розташовувати видачу наборів в найбільш вдалих для громадян, які її потребують, місцях.

Не буде зайвим також додати, що згідно до звіту «МЕХАНІЗМИ НАДАННЯ ДЕРЖАВОЮ ГУМАНІТАРНОЇ ДОПОМОГИ В УМОВАХ ВОЄННОГО СТАНУ» за червень 2022 року до ключових проблем функціонування механізму забезпечення гуманітарної можна віднести:

- подекуди нескоординована робота місцевої влади та волонтерських ініціатив / благодійних та громадських фондів (організацій);
- змішування процесів постачання гуманітарної допомоги на усіх рівнях, внаслідок чого навіть у державного апарату відсутнє єдине розуміння як працює система забезпечення гуманітарною допомогою у воєнний час.
- відсутня ефективно працююча системи визначення, аналізу та систематизації потреб у гуманітарній допомозі, особливо у регіонах, на території яких ведуться бойові дії, що також впливає на ефективність надання допомоги (пошуку оптимального донора) та забезпечення належного контролю за її розподілом та цільовим використанням; система здебільшого працює не знизу догори, а у зворотному напрямку – згори донизу;
- відсутність прозорого механізму розподілу допомоги як на складах першого вивантаження, так і отримувачами допомоги для її передачі кінцевим набувачам;

Хотілося б додатково підкреслити наступні проблеми:

- населення недостатньо поінформоване про те, де і коли можна повідомити про потреби та отримати гуманітарну допомогу;
- відсутність практики прозорого звітування. [3]

Якщо перші три наведені проблеми наш додаток напряму вирішити не може, то додатково відокремлені – може хоча б частково. Наприклад, система

дасть змогу проінформувати населення про надходження вантажів з гуманітарною допомогою шляхом опитувань, час можна буде дізнатися, якщо ввести відповідну команду. Система може розширюватися і вирішити проблему, пов’язану з прозорим звітуванням - хоча б частково, адже усі дії, пов’язані з розподілом вантажів, опитуваннями, виконані користувачем, відповідальним за моніторинг, записуються, а в майбутньому можна призначати відповідальних за доставку на ту чи іншу точку розташування і ввести систему рейтингу, яка допоможе краще вести звітування.

На основі приведених фактів, можна зробити висновок, що проблеми, які частково може вирішити інформаційна система є актуальними насьогодні, а значить її розробка є доцільною.

1.2. Аналіз існуючих рішень

Для того, щоб визначитися чи буде корисним функціонал, який надає система, проаналізуємо наявні рішення, які є станом на зараз.

Аналіз веб додатків, які дозволяють полегшити надання допомоги.

United24 –дозволяє полегшити переведення і контроль коштів.

#HelpUkraineWithCrypto – дозволяє полегшити переведення коштів, направлених для ЗСУ і цивільного населення у криптовалюті, має веб-інтерфейс з описом мети проекту.

Help.gov.ua – надає можливість збирати допомогу для ЗСУ та цивільного населення з-за кордону, містить загальні категорії потреб.

#Є_Допомога – інформаційна система націлена на координацію запиту на надання допомоги та можливостей для її забезпечення, що базуються на внесках донорів, партнерів бізнесу та волонтерів. Дозволяє отримати можливість отримати грошову підтримку від держави, волонтерську допомогу та пошук допомоги для переїзду.

Help Ukraine Center – платформа для того, щоб самостійно закуповувати гуманітарну і медичну допомогу та направляти її в Україну.

СпівДія – платформа для об'єднання волонтерських та державних ініціатив для гуманітарної допомоги під час війни. Проводить координацію потреб і допомоги для цивільного населення.

Prozorro+ - система для пошуку постачальників товару для забезпечення існуючих потреб та пошук фінансування міжнародних донорів.

Паляниця.Інфо – інформаційний портал для пошуку гуманітарної допомоги, тимчасового житла, медикаментів. Можна як подати оголошення так і знайти допомогу з можливістю зареєструватися в якості волонтеру.

#ЛогістикаРазом – об'єднує державу, волонтерів бізнес для забезпечення логістики. Для користування необхідно залишити заявку у телеграм-бот та очікувати дзвінка від оператора, після чого узгодити з ним додаткові деталі.

Залізна_допомога – збирає потреби щодо здійснення доставки гуманітарної допомоги. Створена акціонерним товариством «Укрзалізниця» і націлена на перевезення вантажів саме з використанням потягів. Головний набувач допомоги ОВА.

Можемо відразу викреслити наступні платформи: *United24, #HelpUkraineWithCrypto, Help Ukraine Center, Залізна_допомога, Prozorro+*. Система Залізна_допомога напряму не взаємодіє з цивільним населенням. Інші системи не націлені на збір потреб у допомозі, а більше підходять для полегшення можливості надання матеріальної допомоги як із-за кордону, так і всередині країни. У нашій системі головна функція – саме моніторинг гуманітарних потреб, а тому перейдемо до більш детального аналізу систем, які мають функцію збору потреб.

Таблиця 1.1 - аналіз особливостей наявних систем

Особливості системи / Наявні Системи	Швидкий пошук наявної допомоги за тегами	Категорізація допомоги (ліки, їжа, одяг)	Є можливість надання допомоги	Націлена на збір індивідуальних потреб	Має систему моніторингу
#ЛогістикаРазом	-	+	+	+	-
Паляниця.Інфо	+	+	+	+	-
СпівДія	-	+	+	+	-
Help.gov.ua	-	+	-	+	-
#Є_Допомога	-	+	+	+	-

Усі ці системи різні: є, наприклад, системи для швидкого пошуку наявної допомоги (Паляниця.Інфо), є системи для координації волонтерської роботи і потреб у допомозі, є ті, які націлені більше на збір потреб і не мають функціоналу реєстрації в якості волонтера. Але їх дещо об’єднує – вони не мають системи моніторингу, а більше націлені на індивідуальну допомогу. Майже в усіх системах є рутинна схожа схема реєстрації – чи то з використанням Google Forms або з спеціальною реєстрацією. Але коли необхідно провести моніторинг за містом і визначитися скільком людям необхідна допомога або де краще розташувати набори, коли їх багато і треба швидко визначитися з їх розподілом – усі проаналізовані системи, нажаль, не можуть в повній мірі забезпечити вирішення цієї проблеми. Наш додаток може усунути цей недолік – обробка заявок проходить не «знизу догори», а навпаки, тобто користувач, який проводить моніторинг, може зазначити про прибуття нової партії гуманітарних наборів і провести опитування про потреби населення в гуманітарній допомозі в певний час.

1.3. Аналіз телеграм ботів для збору потреб у гуманітарній допомозі і близьких за функціоналом до нашої системи

Stop Russian war bot – бот, на який можна відправляти повідомлення про пересування ворожої техніки за координатами. Не підходить для наших задач, оскільки необхідно проводити моніторинг ситуації у конкретних населених пунктах, а також розсилати повідомлення про розташування найближчої точки. Архітектура також не дуже підходить під наші задачі, оскільки 1 учасник може відповісти на опитування лише 1 раз, а не відправляти повідомлення кожного разу. Очевидно, що бот не підходить для збору потреб у гуманітарній допомозі.

Odeshyna help bot - бот дозволяє долучитися до волонтерської роботи або ж отримувати допомогу в Одеській області та оптимізувати роботу волонтерських організацій в регіоні. Цей бот нам не підходить, бо він прив'язаний до одного регіону і не дозволяє проводити моніторинг.

Рисунок 1.1 – odeshyna help bot

@saveua_bot - бот для пошуку і надання допомоги (транспорт, їжа, медикаменти).

Приклад роботи:

- Необхідно вибрати місто, де потрібна допомога;
- Вибрати категорію допомоги;

- Ввести номер телефону;
- Як можна докладніше описати запит: що потрібно, коли, скільки.

Волонтери, які заздалегідь організували запит відповідно до вказаних потреб, будуть проінформовані про звернення, що надійшло. У разі відгуку якогось із волонтерів на цей запит, надсилається повідомлення з їхніми контактними даними для подальшої комунікації. Бот не підходить для наших цілей, бо за допомогою нього неможливо проводити моніторингові процеси. [4]

@share_help_bot – бот дозволяє користувачам, які потребують допомоги знайти тих, хто може допомогти. Аналогічно до попереднього боту – не має можливостей для моніторінгу.

@nampodorozi_bot – бот допомагає користувачам з пошуком автомобілів. Не підходить для наших задач.

@turbotnyk_bot – бот дозволяє отримати додаткову інформацію про набуття допомоги внутрішнім переселенцям і постраждалим – видає контактні дані за регіонами і типом допомоги. Також не підходить для вирішення нашої проблеми.

@DopomogaRazomBot – дозволяє сформувати запит на отримання допомоги, після чого зв’язатися з оператором для уточнення деталей. Запити надходять за заявками «знизу до гори», тому не підходить для задач моніторингу.

Тобто існуючі на момент середини 2023 року рішення не підходять для наших задач, а тому можна сказати, що створення цієї системи може бути актуальною задачею насьогодні.

1.4. Постановка задачі

Для створення інформаційної системи необхідно реалізувати наступні задачі:

1. виконати огляд інструментів для розробки телеграм ботів та Restful web services та обрати оптимальні;
2. розробити інтуїтивно зрозумілий інтерфейс панелі для моніторингу;
3. розробити зрозумілу структуру і порядок опитувань;
4. виконати проектування архітектури підсистеми, яка відповідає за процеси моніторингу та підсистеми, відповідальної за опитування, способи взаємодії цих підсистем;
5. розробити алгоритм роботи моніторингової системи, системи сповіщень, опитування, підбору найкращих точок розташування;
6. виконати мануальне тестування системи моніторингу, опитування та зробити відповідні висновки по виконанню роботи;
7. розробити функціонал надання адміністратору можливості проводити опитування;
8. розробити функціонал моніторингу за ситуацією по містах для адміністратора;
9. впровадити можливість адміністратору проставляти точки розташування вантажів гуманітарної допомоги;
10. розробити функціонал оновлення і додавання інформації про користувача в telegram боті;
11. розробити функціонал, який дозволяє брати участь у голосуванні за обраним населеним пунктом в telegram боті;

У разі успішної реалізації усіх пунктів у адміністратора буде можливість проводити моніторинг гуманітарних потреб населення, проводити опитування, а користувачі зможуть голосувати та отримувати найближчі точки видачі.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

За визначеною метою роботи можна провести літературний огляд наявних інструментів розробки, за допомогою яких можна досить швидко і більш-менш якісно розробити онлайн-систему, яка має адміністративний/моніторинговий інтерфейс користувача, зможе в реальному часі надсилати опитування і надавати змогу реагувати на них, а також сповіщати користувачів, якщо за допомогою цієї системи було визначено точки розташувань гуманітарної допомоги у деякому населеному пункті.

Базові інструменти, які треба проаналізувати – це:

- Мови програмування, за допомогою яких можна створити вибрану систему.
- Способи створення опитувань.
- Фреймворки, за допомогою яких можна прискорити розробку.
- Вибір бази даних, яка найбільше підходить для вирішення поставленої задачі.

2.1. Аналіз рішень для підсистеми опитування

Рішень для підсистеми опитування може бути декілька: створення веб-сайту, мобільного додатку і тп. Але усі вони займають досить багато часу на розробку, а також ускладнюють процес голосування (необхідністю кожного разу заходити на сайт раз або завантажувати мобільний додаток). Замість цього можна використати достатньо відомий месенджер telegram та API, за допомогою якого можна створити бота, який автоматично буде відправляти та обробляти усі необхідні запити. Також в системі необхідно буде відсилати своє місцерозташування, ця функція підтримується в telegram bot API.

2.2. Вибір мов програмування.

Якщо подивитися на рейтинг найбільш популярних мов програмування для back-end, який був проведений редакцією DOU, то маємо наступні результати:

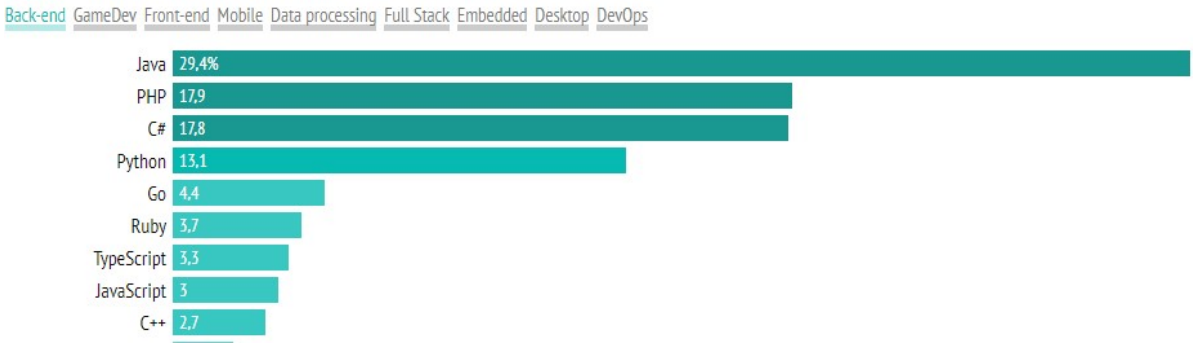


Рисунок 2.1 – рейтинг мов програмування [5]

Можна підмітити, що Java у цьому списку на першому місці. Все це завдяки вузькій спеціалізації (для enterprise рішень) цієї мови і створенні багатьох фреймворків і бібліотек, які допомагають розробити ці рішення з легкістю, високою швидкістю, використанням сучасних підходів розробки (створенні тих же мікросервісів) і дуже високій гнучкості та масштабованості.

Та все ж таки, як мінімум, топ 5 мов програмування мають спеціальні фреймворки, які дозволяють використовувати їх для побудови серверної частини системи і, можна сказати, урівнюють можливості і швидкість розробки (якщо не брати до уваги проблеми з оптимізацією, які виникають при неправильному виборі технології для певних інформаційних систем).

Існують, звісно, особливі інструменти як, наприклад, C++, який, скоріше за все, підходить для створення оптимізованих систем з високим навантаженням, бо розробляти на ньому ті ж мікросервісні рішення довго і дорого. Подібні інструменти не знадобляться нам при виконанні роботи. Тому вибір мови програмування зводиться лише до того, щоб вона не виходила за

рамки «швидкості обробки» запитів, виконувала деякі специфічні вимоги і подобалася розробнику.

Для розробки цієї системи добре підходить мова програмування Java. Використовуючи цей інструмент можна створити адмін-панель та панель моніторингу, беручи за основу монолітну архітектуру. А також можна використати Python для створення Telegram боту - завдяки специфічним безкоштовним, гарно задокументованим бібліотекам, які існують у ньому і значно спростять нашу роботу, дякуючи принципу відкритості коду і гнучкості мови програмування. Для розробки клієнтської частини буде використовуватися Javascript з використанням бібліотеки ReactJS.

2.3. Фреймворки та бібліотеки, за допомогою яких можна розробити інформаційну систему.

Фреймворки і бібліотеки використовуються для того, щоб уникнути написання базового функціоналу кожного разу з нуля, однак різниця між ними полягає в терміні, що називається інверсією контролю.

Коли використовується бібліотека, розробник сам відповідає за потік виконання програми. Він сам обирає, коли і у яких місцях звернутися до готових модулей бібліотеки. Коли використовується фреймворк, то саме він відповідає за виконання потоку. Фреймворк надає місця, куди можна підключити свій код, але виконує його лише при певних ситуаціях.[6]

Як відомо, лідером серед фреймворків на мові програмування Java є Spring. Spring Framework забезпечує комплексну модель програмування та конфігурації для сучасних корпоративних систем на основі Java на будь-якій платформі розгортання.

Ключовим елементом Spring є інфраструктурна підтримка на рівні додатків: Spring зосереджується на «сантехніці» корпоративних додатків, щоб команди могли зосередитися на написанні бізнес-логіки додатку без

непотрібних зв’язків із певними середовищами розгортання [7], що дуже спрощує розробку.

Фреймворк Spring складається з багатьох модулів, таких як core, beans, context, EL, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts тощо. Ці модулі згруповані наступним чином (рис. 2.6).

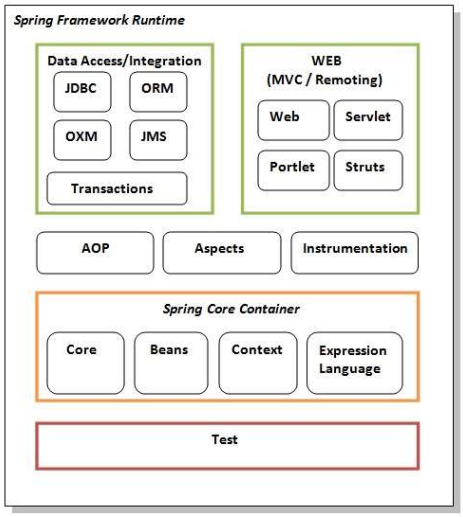


Рисунок 2.2 – опис модулів Spring Framework [8]

Для розробки системи буде доцільним використання Spring Core, Beans, Context, Expression Language, Spring Webmvc, DA (JPA).

Для швидкої конфігурації скористуємося надбудовою Spring Boot, яка автоматично налаштовує проект та надає можливість набагато простіше підключати необхідні бібліотеки. Для реалізації специфікації JPA використаємо ORM бібліотеку Hibernate.

Для реалізації боту на мові python використаємо фреймворк aiogram а також бібліотеки, що надають назви населених пунктів за координатами (reverse geocoding) – geopy.

2.4. Вибір системи керування базою даних

Для основної бази даних використаємо саме реляційну базу даних, завдяки простоті запитів (SQL), широкій підтримці таких СКБД мовами програмування Java та Python, підтримці усього необхідного нам функціоналу, низькими обсягами займаної пам’яті та узгодженості. [9] Проаналізуємо найпопулярніші системи, які можна використати для виконання роботи.

MySQL

- Проста у розгортанні та управлінні.
- Підтримує ACID що робить її дуже надійною.
- СКБД підтримує утиліти з різною кеш-пам’яттю для обслуговування та адміністрування серверів.
- MySQL можна налаштувати будь-якою мовою програмування, але в основному використовується з PHP.
- Забезпечує високопродуктивні результати без шкоди для основних функцій.
- Забезпечує повний захист даних, оскільки складається з надійних рівнів безпеки даних: тільки авторизовані користувачі можуть отримати доступ до бази даних за допомогою зашифрованих паролів

PostgreSQL

- Містить численні обмеження, які можуть забезпечити цілісність даних. Ці обмеження включають Primary Keys, Foreign Keys, Explicit Locks, Advisory Locks, Exclusion Constraints.

- Підтримує різні функції SQL, такі як багатоверсійний контроль паралелізму, підвибір SQL, складні запити SQL, потокова реплікація тощо.
- Сумісний із кількома типами даних, включаючи структуровані, примітивні, створені власноруч, геометричні та документи.
- Розширювана різними способами, як-от виразами JSON/SQL і за допомогою збережених процедур та функцій.

Microsoft SQL Server

- Microsoft SQL Server та інші інструменти Big Data можна використовувати для створення чудового Data Lake.
- Постачається з вбудованими функціями для класифікації, захисту та моніторингу даних, завдяки чому визначає та надає сповіщення про підозрілу діяльність, прогалини в безпеці та неправильні налаштування.
- Підтримує структуровані, напівструктуровані та просторові типи даних.
- Поставляється зі спеціально побудованим графічним інтерфейсом.

Oracle

СКБД Oracle є однією з найбільш широко використовуваних баз даних у галузі, оскільки вона підтримує усі типи даних, включаючи реляційну, графову, структуровану та неструктуровану інформацію, і тому вважається однією з найкращих баз даних, доступних на ринку. Ключові особливості Oracle Database такі:

- Містить численні функції, такі як Real Application Clustering і Portability, що робить цю СКБД набагато більш масштабованою під час зростання бізнесу.
- Пропонує високопродуктивне обчислювальне середовище, яке є достатньо потужним, щоб забезпечити постійну доступність даних. Оснащений комплексними функціями відновлення, щоб виводити дані після збоїв.

MariaDB

- Сумісна з протоколом і клієнтами MySQL. MariaDB можна легко замінити сервер на MySQL без будь-яких додаткових вимог.
- Підтримує стовпчасте зберігання та має масивно-паралельну розподілену архітектуру даних.

SQLite

- Не потребує налаштування та навіть не потребує сервера чи інсталяції.
- Пропонує популярні функції програмного забезпечення системи керування базами даних для використання в мобільній веб-розробці, наприклад React Native.
- Але через це має обмежену функціональність (не містить процедур і тд) [10]

При розробці системи доцільним буде використання PostgreSQL, адже ця СКБД не містить зайвого функціоналу для розробки і масштабування великих систем як Oracle, має весь базовий функціонал, який потрібен для розробки системи і сервер в порівнянні з SQLite, досить часто

використовується у зв’язці з Java, Python на відміну від MySQL, яка в широкому загалі працює з PHP та не займає додаткове місце впровадженням аналізаторів, моніторингових систем. Тобто це досить проста і універсальна система, якої достатньо для вирішення наших задач.

Також буде використана noSQL MongoDB для збереження сесій у Telegram боті.

Для реалізації поставлених задач необхідно спроектувати базову архітектуру додатку та бази даних, а також розробити алгоритм підбору найкращих точок і створити зрозумілий UI.

Для проектування можна використати наступні діаграми:

- Use case
- Entity-Relationship

2.5. Проектування бази даних

База даних створюється за допомогою бібліотеки Hibernate, яка реалізовує принцип JPA.

Інформаційна система має зв’язки між сутностями в базі даних, продемонстровані на рисунку 2.3.

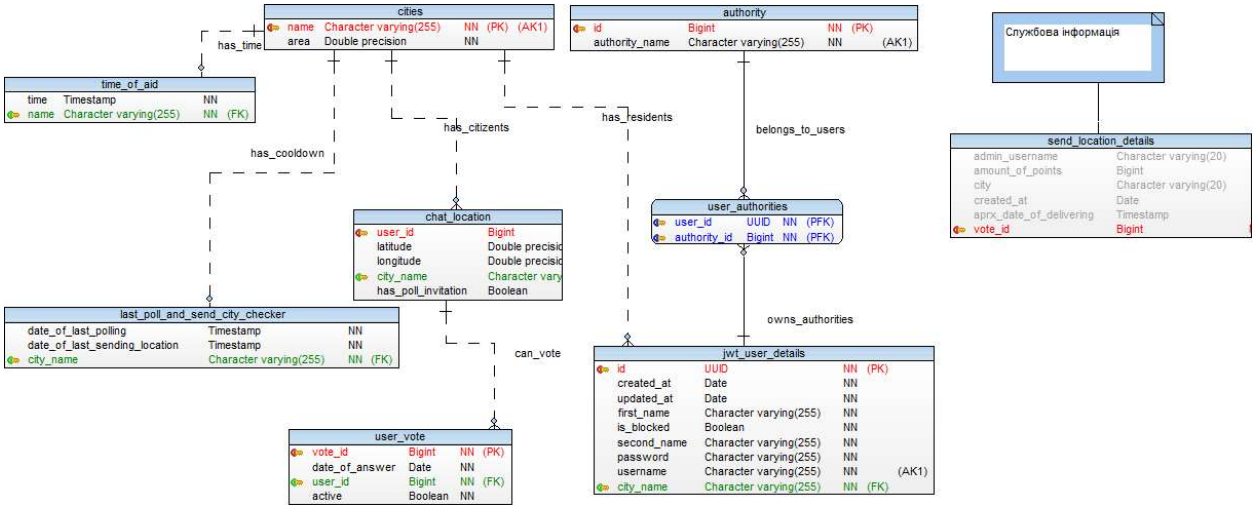


Рисунок 2.3 – ERD діаграма бази даних

Сутність user_authorities відноситься безпосередньо до автентифікації в Веб-додатку і видачі ролей. Натомість chat_location відповідає за інформацію про людей, які потенційно можуть бути опитаними з використанням telegram API. Send_location_details – службова інформація.

Перевага цієї системи в тому, що вона легко може бути розширена і в неї можна додати новий функціонал з найменшими зусиллями. Наприклад, додати можливості для волонтерів, користувачів у веб додатку і тд.

2.6. Use case діаграма

Для розуміння функціонування систему розробимо Use Case діаграму (рис 2.4)

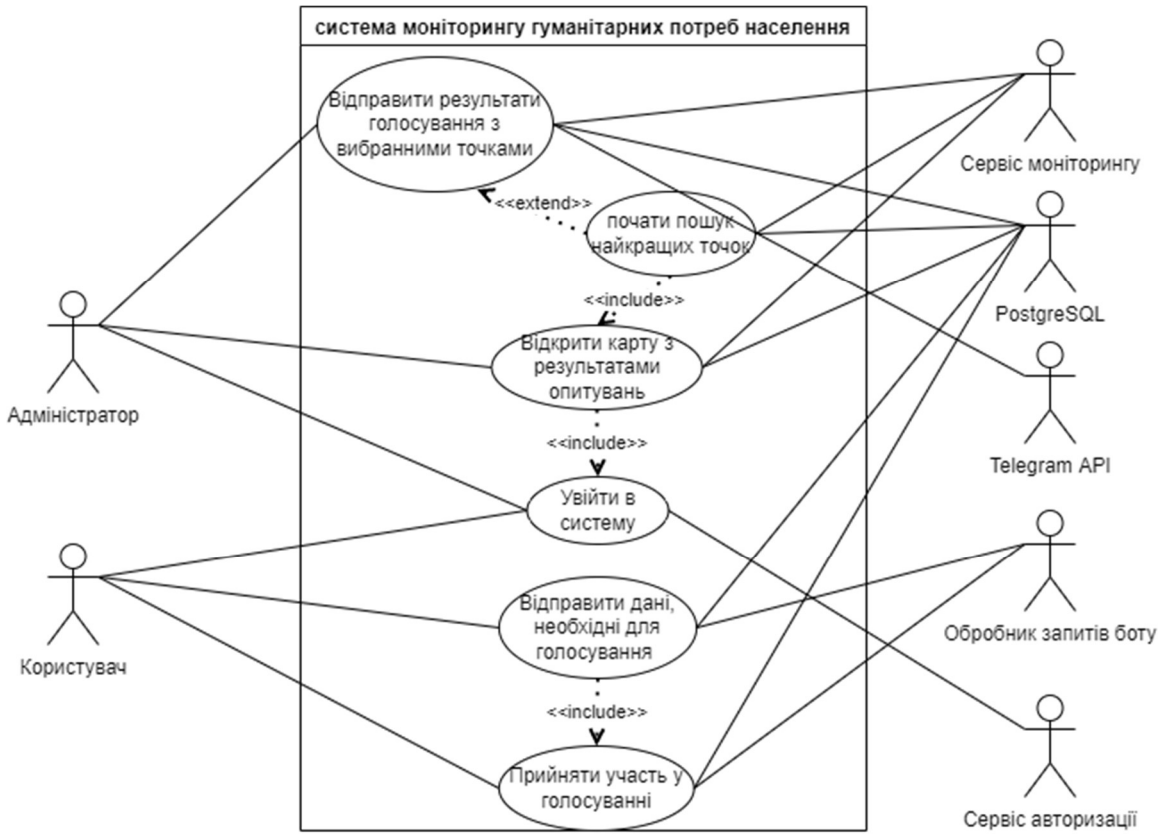


Рисунок 2.4 – Use case діаграма ІС

2.7. Алгоритм підбору найкращих точок

Для впровадження алгоритму необхідно проаналізувати і вибрати спосіб знаходження відстані між двома точками, вираженими у широті й довготі, для цього є декілька способів:

Відстань між точками за формулою Гаверсина (Рівн. 2.1).

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi1 \cdot \cos\varphi2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

Рівняння 2.1 – формула Гаверсина

, де φ - latitude, λ - longitude, R - радіус землі (radius = 6,371km);Кути необхідно вимірювати в радіанах.

За сферичним законом косинусів:

$$d = \operatorname{acos}(\sin\varphi1 \cdot \sin\varphi2 + \cos\varphi1 \cdot \cos\varphi2 \cdot \cos\Delta\lambda) \cdot$$

Law of cosines: R

Рівняння 2.2 – формула за правилом косинусів

Але якщо важлива швидкодія і точність не критична, а в нашому випадку це так, то можна використовувати звичайну теорему Піфагора [11]

$$x = \Delta\lambda \cdot \cos\varphi m$$

$$y = \Delta\varphi$$

$$d = R \cdot \sqrt{x^2 + y^2} \quad (2.2)$$

Рівняння 2.3 – відстань за формулою Піфагора

Розташування точок є орієнтовним, бо на остаточний вибір впливають локальні особливості чи фактори які важко врахувати (наприклад, наявність під'їзних шляхів, особливості із точки зору забезпечення безпеки тощо). Тому достатнім буде використання алгоритму кластеризації k-means - мабуть, найбільш широко використовуваною та дослідженою Алгоритм кластеризації k-means, який також називають узагальненим алгоритмом Ллойда (GLA), є окремим випадком узагальненої схеми жорсткої кластеризації, коли приймаються точкові представники (в нашому випадку координати) та евклідові відстані (або ж можна використовувати формулу Гаверсина 2.1) використовуються для вимірювання відмінності між вектором X і представником його кластера C [12]. А точніше його модифікацію kmeans++ , яка покращує алгоритм k-means за допомогою дуже простої рандомізованої методики вибору точок (seeding). Як наслідок, можна отримати алгоритм, який має складність Θ(log(k)) з оптимальною кластеризацією. Експерименти показують, що модифікація покращує як швидкість, так і точність k-means, і часто досить суттєво [13].

У випадку, якщо потрібна додаткова точність і якщо хочемо розцінювати географічні координати як неевклідові дані, то в такому разі їх кластеризація доволі складна і потребує використання додаткових алгоритмів. Одним із найбільш поширених алгоритмів для таких задач, окрім ієрархічної кластеризації, є популярний алгоритм Partitioning Around Medoids (PAM), який також просто називають k-medoids [14] та CLARA [15].

Якщо ж методи кластеризації з якихось причин не підходять, то можна спробувати використати наступний алгоритм:

Для обчислення відстані можна скористатися теоремою Піфагора (рівн. 2.3).

Наступним кроком буде обчислення коефіцієнтів кожної з точок, в яких люди приймали участь у голосуванні за наступною формулою:

$$k = \sum_{i=0}^{n} e^{-L*d_i} \quad (2.4)$$

, де n – кількість точок в населеному пункті, L – сталий коефіцієнт для населеного пункту, di – відстань між поточною точкою і тою, в якій шукають коефіцієнт в кілометрах.

Саме експонента з від'ємним степенем була обрана в цій формулі, бо ця функція спадаюча на проміжку [-∞;+∞] та $e^0 = 1$.

В залежності від константи L можна регулювати наскільки слабко на коефіцієнт впливає відстань між далекими точками.

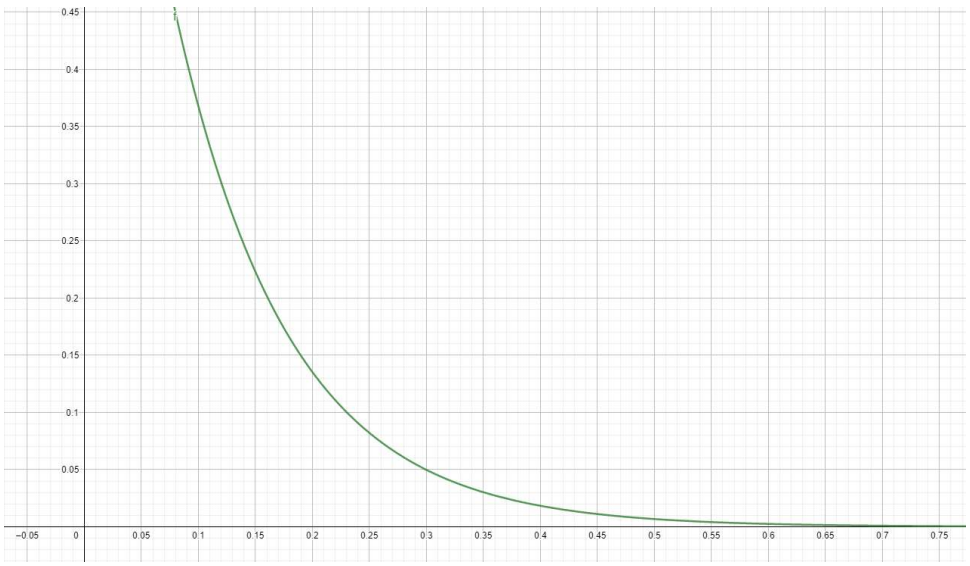


Рисунок 2.5 – графік функції e^{-10x}

Якщо проаналізувати вищенаведений графік, то стає зрозумілим, що відстань в 300 метрів буде додавати до коефіцієнту 0.05 пункту. Користуючись цією інформацією можна припустити, що сталу L для населеного пункту можна обчислити наступним чином:

$$L = \frac{t}{S} * K \tag{2.5}$$

, де t – кількість точок, які треба знайти, S – площа міста, K – сталий коефіцієнт.

Наступним кроком буде обчислення радіусу «кіл» з центром у найкращій позиції в межах яких видаляються усі точки. За формулою

$$r = 1.5 * \sqrt{\frac{a}{\pi}}/b \tag{2.6}$$

, де a – площа міста, b – кількість точок, які необхідно знайти.

Варто зазначити, що використання алгоритму є опціональним. Тобто якщо людині, яка відповідає за моніторинг не сподобається розташування пунктів, то вона зможе виставити їх самостійно (можливо, виникла деяка некоректна робота алгоритму при великих відстанях між поодинокими точками).

Тобто, підсумовуючи, алгоритм має наступний порядок дій:

1) Для першої потенційної точки проаналізувати усіх претендентів, використовуючи вищенаведені формули (рівн. 2.4) та знайти точку з найкращим коефіцієнтом.
2) Зберегти цю точку у список, зменшити лічильник точок, які треба прорахувати на 1.
3) Видалити усі точки з відстанню менш ніж радіус кола, який обчислюємо за формулою (рівн. 2.6)

4) Повторюємо операції 1,2,3 до моменту, коли лічильник не дійде до 0, якщо лічильник став дорівнювати кількості точок які залишилися, то використати їх в якості найкращих, щоб лічильник дійшов до нуля.

Зрозуміло, що розроблений алгоритм далеко не ідеальний, але тому його використання і є опціональним. Якщо точок багато, а місць для пошуку мало, то він зможе обрати найкращі точки для розташування гуманітарних вантажів. Було також розраховано, що алгоритм кластеризації kmeans++ набагато швидше знаходить центри розташувань скупчень розташувань користувачів, які хотіли б отримати гуманітарну допомогу, тому кращим рішенням буде використання саме цього надійного рішення.

2.8.Серверна частина

При розробці серверної частини додатку буде використана мова програмування Java та фреймворк Spring.

Основним способом взаємодії і обміну даними між частинами системи, які відповідають за моніторинг і проведення опитувань було обрано REST. Це досить легкий у використанні і простий в реалізації, порівняно з іншими, архітектурний стиль, якого можна дотримуватися шляхом впровадження низки певних архітектурних рішень у системі[16]. Його було обрано з деяких причин. По-перше, цей стиль досить популярний насьогодні, особливо, якщо вести мову про його розробку на мові програмування Java. Через це існує багато корисної і якісної документації, а також додаткових інструментів для полегшення створення таких систем. Дуже важливо, що цей метод взаємодії також досить добре підходить для роботи з так званими односторінковими додатками, адже може використовувати для маніпуляцій над ресурсами такий формат повідомлень як JSON, який був створений для роботи з JavaScript. Наша клієнтська частина додатку використовує бібліотеку React JS, тому цей тип взаємодії підходить для нас.

Основні принципи побудови REST сервісу за допомогою Spring Boot наступні:

- Створення моделі (усі звичайні класи, які будуть використовуватися в проекті і представляють собою каркас даних, якими він маніпулює).
- Визначення відношень між класами моделі між собою (за допомогою jpa анотацій).
- Створення спеціальних інтерфейсів, які заносять дані, представлені об’єктами класів моделі у базу даних (Repository).
- Створення усіх необхідних конфігураційних бінів.
- Створення сервіс класів, які вміщують у собі методи, які виконують бізнес логіку системи.
- Створення REST контролера (за шаблоном MVC) і прописом базових шляхів.
- Створення користувальницьких помилок для якісної обробки непередбачуваної роботи системи.
- Захист системи (Spring security).
- Тестування (або тестування можна поставити на перше місце TDD).

Детальніше можна поговорити про захист системи, адже стандартна авторизація за допомогою сесій не дуже підходить для нашого додатку, через те, що ми використовуємо бібліотеку ReactJS в якості клієнтської частини.

Одним із варіантів вирішення цієї проблеми може бути JWT token. Веб-токен JSON (JWT) — це компактний формат представлення тверджень (інформації про користувача), призначений для середовищ з обмеженим простором, таких як заголовки авторизації HTTP та параметри запиту URI. JWT кодують твердження для передачі як об’єкт JSON [RFC7159] [17]. Суть роботи наступна: користувач заходить під логіном та паролем, після успішного входу генерується web token, в якому зашиті різні відомості про

користувача (claims) та про сам токен і перевіряючий підпис, уся інформація кодується в base 64, але останню частину (підпис) можна отримати лише якщо знати секрет. Завдяки цьому можна легко передавати твердження користувачу (можливо, для відображення додаткової інформації в клієнтській частині), але якщо недоброчесний користувач захоче відправити дані, які йому не належать, то він не зможе зробити цього просто закодувавши інформацію за допомогою base64, йому необхідно буде відправити підпис, який без знання секрету неможливо підробити.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Реалізація архітектури .

Для реалізації RESTful API сервісу було використано фреймворк Spring Boot і деякі супутні технології, які полегшують розробку подібних систем. Робота частини додатку, яка відповідає за процес моніторингу продемонстрована на наступній ілюстрації:

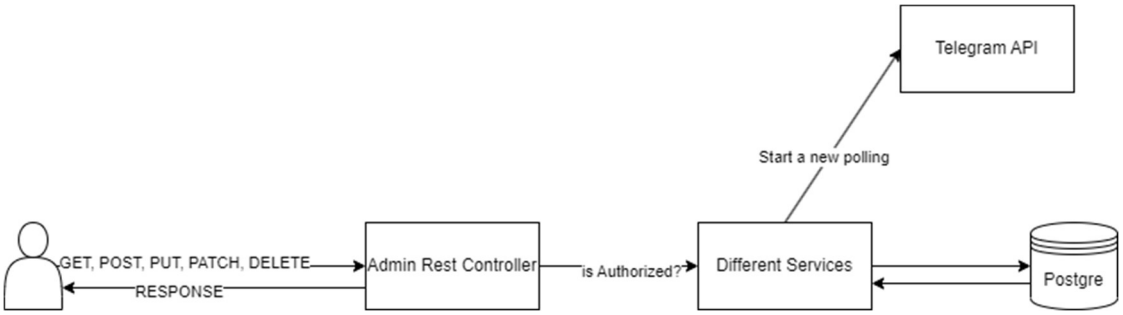


Рисунок 3.1 - архітектура взаємодії частин додатку, які відповідають за моніторинг і голосування

На ілюстрації (рис. 3.1) в якості користувача виступає клієнтська частина (frontend), створена за допомогою бібліотеки React JS. В цій схемі припускається, що користувач авторизований та має спеціальну роль адміністратора, який може використовувати усі можливості додатку. Саме за допомогою React JS запити відправляються на обробку сервером. Також на рисунку окремо підкреслені сервіси, які відповідають за виконання бізнес-логіки додатку, деякі з них звертаються до бази даних, але інші – можуть звертатися до Telegram API, тим самим ініціюючи процес опитування людей, які потребують допомоги у певному населеному пункті.

Треба також згадати про функціонал і архітектуру Telegram бота, який є незамінною частиною нашого додатку. Для його розробки було використано бібліотеку Aiogram та Geopy. Його робота продемонстрована на наступній ілюстрації:

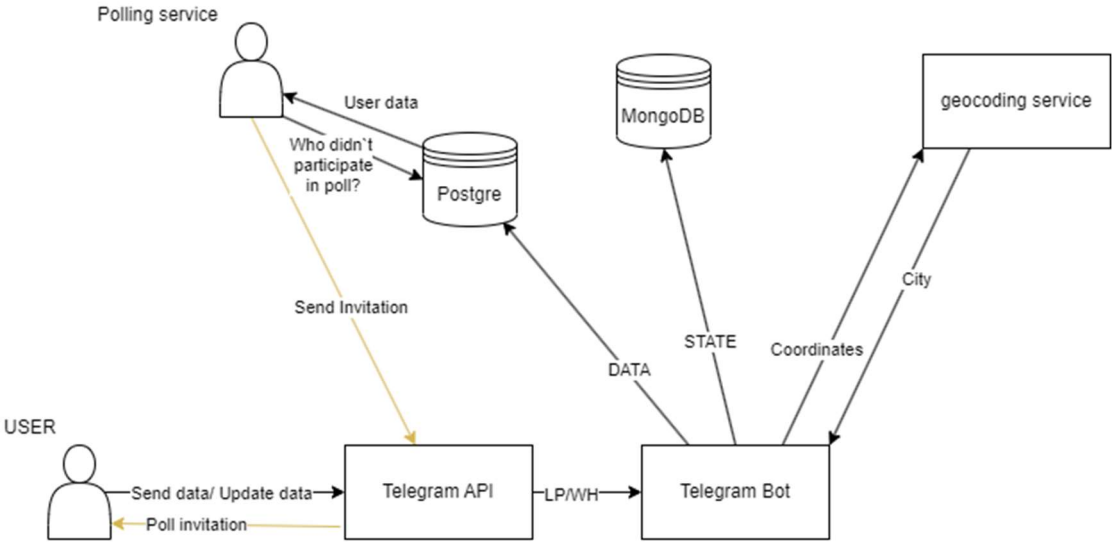


Рисунок 3.2 - архітектура взаємодії з ботом Telegram

За допомогою цього бота користувачі можуть залишити інформацію про своє місце перебування (без додаткових подробиць таких, як: ім’я, прізвище, дата народження і тп). Після чого автоматично визначається місто громадянина і його координати. В подальшому, якщо буде проводитися опитування за цим містом, то користувачу прийде повідомлення з вибором: чи потрібна йому гуманітарна допомога. Якщо ж він голосував у опитуванні, то при визначенні позицій розташування вантажів з гуманітарною допомогою, йому буде надіслане повідомлення про найближче до нього місце.

На рисунку 3.2 можна побачити, що система проведення моніторингу та голосування (Polling service) тісно пов’язана з даними, які надсилає користувач. Також спеціальна інформація стану користувача зберігається в документо-орієнтованій СКБД Mongo DB та використовує сервіс оберненого геокодування (або ж визначення міста за координатами).

3.2. Програмна реалізація серверної частини.

Як вже було зазначено, для маніпуляції над даними і роботою за базою даних було обрано технологію Hibernate, яка, в свою чергу, може бути застосована в реалізації специфікації JPA. Вона може значно спростити роботу з базою даних. В нашому проекті будемо використовувати варіант реалізації Spring JPA (який використовує Hibernate), адже було вирішено, що для реалізації усього проекту Spring Boot.

Використання JPA технології може мати наступний вигляд:

Спершу, необхідно створити клас, який в майбутньому буде зберігатися у базі даних за допомогою технології ORM (Object-Relational Mapping). І помітити його анотацією «@Entity»:

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ChatLocation {

    @ManyToOne
    @JoinColumn(name = "city_name", referencedColumnName = "name")
    private City cityName;

    private boolean hasPollInvitation;

    @Embedded
    private LocationCoordinates locationCoordinates;

    @Id
    @Column(name = "user_id")
    private long chatId;
}
```

У цьому прикладі був наведений клас ChatLocation, який може вміщувати у собі інформацію про користувачів, які зареєструвалися, використовуючи телеграм бота. В цьому прикладі є ще декілька анотацій, які

грають дуже важливу роль у створенні інформаційної системи. Анотація @ManyToOne – означає існування відношення «одного до багатьох» між таблицями City та chat_location, @JoinColumn – використовується для встановлення зв'язку між сутностями за допомогою зовнішнього ключа (foreign key) в базі даних, @Embedded – поле, яке не є примітивом і може вміщувати в себе об’єкт класу, позначеного як @Embeddable. @Data, @AllArgsConstructor, @NoArgsConstructor – анотації, які дозволяють спростити процес створення конструкторів класів, методів для доступу до полів класу (таких як get, set) за допомогою бібліотеки Lombok.

Після того, як усі необхідні класи, які будуть перенесені у вигляді таблиць у базу даних, були створені, ми можемо написати необхідні запити, а також отримати вже готові - ті, які надає нам Spring JPA. Для цього необхідно створити необхідний нам інтерфейс та успадкуватися від інтерфейсу JpaRepository<Object, Id> :

```
      public interface ChatLocationRepository extends
JpaRepository<ChatLocation, Long> {
          public  List<ChatLocation> findByCityName(City name);

          @Query("select c.chatId from ChatLocation c where
c.hasPollInvitation = ?2 and c.cityName = ?1")
          public List<Long> getChatIdBy(City city , Boolean
hasPollInvitation);

          @Modifying
          @Query("update ChatLocation c set c.hasPollInvitation=false where
c.chatId in (select r.chatLocation.chatId from UserVote r where
r.chatLocation.cityName = ?1 and r.active = true)")
          public void setHasInvitedToFalseForVotedLocationsByCity(City city);

          @Modifying
          @Query("update ChatLocation c set c.hasPollInvitation = ?2 where
c.cityName = ?1")
          public void setHasInvitedToFalse(City city, boolean status);

          public List<ChatLocation> findAllByCityNameAndHasPollInvitation(City
name,boolean hasPollInvitation);
```

```
    }
```

В нашому прикладі, для простоти, було приведено приклад створення репозиторію, пов’язаного з попереднім класом. Як можна побачити, ми можемо створювати операції як по ключовим словам, так і за допомогою JPQL. Метод public List<ChatLocation> findAllByCityNameAndHasPollInvitation(City name,boolean hasPollInvitation); автоматично генерує запит, аналізуючи необхідні слова в назві методу. А ось метод setHasInvitedToFalse(City city, boolean status) вже використовує JPQL - Java Persistence Query Language, але можна використовувати і SQL, вказавши nativeQuery = true у анотації.

Тепер ми можемо використовувати наш репозиторій для роботи з базою даних у наших класах, які займаються бізнес-логікою. Для цього необхідно створити необхідні інтерфейси, які помічають класи як ті, котрі реалізовують певний функціонал:

```
    public interface ChatLocationService {
        public List<ChatLocation> getChatsByCity(City city);

        void updateChatInvitationStatusByCity(City city, boolean status);
    }
```

І реалізовуємо функціонал, як нам потрібно:

```
    @AllArgsConstructor
@Service
public class ChatLocationServiceImpl implements ChatLocationService {
    private final ChatLocationRepository chatLocationRepository;

    @Override
    public List<ChatLocation> getChatsByCity(City city) {
        return chatLocationRepository.findByCityName(city);
    }

    @Override
    public void updateChatInvitationStatusByCity(City city, boolean status) {
        chatLocationRepository.setHasInvitedToFalse(city, status);
```

```
    }
}
```

В цьому випадку бізнес-логіки небагато, ми лише використовуємо сервісний клас у ролі проміжного класу між тим, який займається взаємодією з базою даних та контролером, який буде описано в наступному прикладі. Але бізнес-логіка може виконувати набагато складніші операції та бути значно комплекснішою.

Spring Boot, на основі якого побудована наша інформаційна система, дозволяє легко створювати REST сервіси. В якості основного архітектурного шаблону програмного забезпечення Spring традиційно використовує Model-Veiw-Controller. Особливість Spring Boot в тому, що нам непотрібно встановлювати усі залежності окремо та дивитися на сумісність тих або інших бібліотек. В одну залежність

org.springframework.boot:spring-boot-starter-web

Вміщується цілих пакет сумісних бібліотек, таких як:

org.springframework:spring-core

org.springframework:web-mvc

org.springframework:spring-web

com.fasterxml.jackson.core:jackson-databind [18]та ін.

Тому, якщо ми хочемо працювати з модулем starter-web, найбільш правильним підходом при використанні цього модуля буде застосування архітектурного шаблону Model-View-Controller (це найбільш легка у реалізації архітектура при використанні цього фреймворку). Це обмеження має на меті написання більш структурованого та масштабованого коду. Тому нам необхідно створити контролер, щоб оброблювати запити до нашої серверної частини інформаційної системи. Для цього треба створити клас і помітити його як @RestController

```
      @RestController
@AllArgsConstructor
@CrossOrigin(origins = "http://localhost:3000")
```

```
@RequestMapping("/admin")
public class CityController {

    private final MapValidationErrorService validationErrorService;
    private final CityValidator cityValidator;
    private final CityService cityService;

    @PostMapping("/addCity")
    public ResponseEntity<?> AddCity(@RequestBody CityDTO cityDTO,
BindingResult bindingResult){
        cityValidator.validate(cityDTO, bindingResult);
        ResponseEntity errors =
validationErrorService.mapErrors(bindingResult);
        if(errors!=null){
            return errors;
        }
        City city =  DtoConverterUtils.convertCity(cityDTO);
        cityService.saveCities(city);
        return ResponseEntity.ok().build();
    }
}
```

Було приведено приклад контролеру з невеликою ендпоінтів, щоб показати принцип роботи з цим архітектурним шаблоном. Як можна побачити, окрім головної анотації «@RestController», ми також додаємо аннотацію «@CrossOrigin» – для можливості відправлення запитів з використанням React js локально з серверу на Node JS і «@RequestMapping» – додаткового єдиного ендпоінту для усього контролеру, який виступає в ролі маршрутизатору.

У самому контролері ми можемо оброблювати 5 різних http методів: put, patch, delete, get, post. Тобто усі ті, які необхідні для маніпуляції над ресурсами з дотриманням архітектурного стилю REST. В нашому випадку ми використали «@PostMapping», який дозволяє створити новий ресурс на сервері, інформація про який надходить в тілі запиту. Інформація, в нашому випадку, у форматі JSON автоматично транспортується в java клас, який помічений анотацією «@RequestBody», важливо, щоб поля класу і назви відповідних ключів у JSON-форматі мали однакове ім’я.

Ендпоінт «/admin/addCity» дозволяє користувачу, який має права адміністратора системи, додавати інформацію про нові міста, а саме: координати його центру, площа міста у км^2, назва міста. Ці параметри передаються у тілі запиту у форматі JSON. Дуже важливо щоб місто існувало і було правильно записане (з великої літери, англійською мовою), адже, якщо назва неправильна, то користувачі просто не зможуть прив’язатися до неї за допомогою механізму оберненого геокодування (так само як і адміністратор). Цей метод також проводить валідацію полів, і, якщо виникли проблеми, повертає помилку у зручному форматі JSON.

Увесь інший функціонал серверної частини розроблюється і впроваджується за схожим механізмом, тому на цих прикладах було продемонстровано основну архітектуру серверної частини інформаційної системи.

3.3. Розробка Telegram Bot за допомогою бібліотеки aiogram.

Найбільш легкий і швидкий в розробці, спосіб проводити опитування – це використання можливостей Telegram Bot API. Так як на меті є розробка досить комплексного додатку, модулі якого можна в майбутньому змінювати, то цей спосіб нам підходить. Для спрощення роботи з цим API було створено велику кількість бібліотек. В інформаційній системі буде використана бібліотека Aiogram, написана мовою програмування Python, адже бібліотека досить популярна, а це значить, що існує багато документації і додаткової інформації про неї, що значно спростить розробку цього модуля. В Telegram Bot API є свої обмеження, тому щоб ця частина ІС не була слабким місцем в майбутньому, необхідно буде створити мобільний додаток для проведення опитувань.

Для того, щоб бот міг відповідати на певні запити, необхідно прив’язати команди до наших функцій.

```
def register_handlers_find_loc(dp1: Dispatcher):
    dp1.register_message_handler(start_application, commands="start_locate",
state="*")
    dp1.register_message_handler(location_confirmed,
content_types=['location'],
                                 state=SendLocation.waiting_for_location)
    dp1.register_message_handler(location_confirmed,
state=SendLocation.waiting_for_location)
    dp1.register_message_handler(warning_confirmed,
state=SendLocation.waiting_for_message)
    dp1.register_message_handler(get_time_of_receiving,
state=SendLocation.waiting_for_polling, commands="time")
    dp1.register_callback_query_handler(polling_confirmation_successful,
state=SendLocation.waiting_for_polling)
```

Як можна побачити, окрім методів ми також прив’язуємо стан користувачів, або ж використовуємо машину станів, наприклад state = “*” означає, що команду можна визивати у будь-якому стані. Скінченний автомат - це математична модель, що складається з кінцевої кількості станів та переходів між ними, і виробляє виходи на переходах після отримання вхідних даних. Цей інструмент широко використовується для моделювання систем у різних галузях, таких як послідовні схеми, програми і протоколи зв’язку, а також у програмуванні, зокрема в мовах, що підтримують концепцію станів. [20]. Бібліотека aiogram надає механізм використання кінцевого автомату, для цього необхідно успадкуватися від класу StatesGroup і визначити необхідні нам стани наступним чином:

```
class SendLocation(StatesGroup):
    waiting_for_location = State()
    waiting_for_message = State()
    waiting_for_polling = State()
```

Завдяки цій конструкції ми можемо почергово проводити дії з користувачем, тобто спочатку попросити надіслати розташування місця проживання, потім запропонувати підтвердити цю інформацію і вже тільки

після цього надавати можливість брати участь у голосуваннях. Наступний код ілюструє це:

```
async def location_confirmed(message: types.Message, state: FSMContext):
    if message.location is None:
        await message.answer("please click on the
button\"{}\"".format(button_location_name))
        return
    await state.update_data(longt=message.location.longitude,
latit=message.location.latitude)

    await message.answer(outside_warning_msg, reply_markup=confirm_keyboard)
    await SendLocation.next()


async def warning_confirmed(message: types.Message, state: FSMContext):
    if message.text.lower() not in [YES, NO, button_outside_warning_y,
button_outside_warning_n]:
        await message.answer(CONFIRMATION_MESSAGE)
        return
    else:
        if message.text.lower().__eq__(NO):
            await SendLocation.next()
            await message.answer(NOT_SAVED_CHANGES_MESSAGE,
reply_markup=types.ReplyKeyboardRemove())
            return
        else:
            user_data = await state.get_data()
            lat = user_data.get(LAT)
            lon = user_data.get(LNG)
            try:
                city = app.converters.street_converter.convert_street(lat,
lon)
                await sql_handler.insert_or_update_user_info(lon=lon,
lat=lat, chat_id=message.chat.id, city=city)
            except AttributeError as atr:
                await message.answer(atr.name)
                return
            except Exception:
                await message.answer(CANT_FIND_CITY_MESSAGE)
                print(traceback.format_exc())
                return
```

```
    await message.answer(button_outside_warning_send_message,
reply_markup=types.ReplyKeyboardRemove())
    await SendLocation.next()
```

Цей код виконує реєстрацію або оновлення даних від користувача у разі потреби. Можна побачити, що використовується state, і якщо дія успішна, то машина станів переходить до наступного стану з використанням команди SendLocation.next() а також виконує збереження даних за допомогою команди state.update_data(), тому під час створення Dispatcher, який оброблює усі запити, які надходять до бота необхідно вказати спосіб збереження стану. Можна вказати стан «у пам'яті», з використанням Redis та Mongo DB. Звісно, що у більшості випадків використовується саме Mongo DB, адже збереження даних у енергонезалежній пам'яті гарантує додаткову надійність і що дані не будуть втрачені після перезавантаження пристрою або додатку. Її можна використати наступним чином:

```
storage = MongoStorage(db_name='PythonPollStates', uri=config.get("mongoDB",
"uri"))
dp = Dispatcher(bot, storage=storage)
```

Цього достатньо, щоб наша система зберігала інформацію про стани у документо-орієнтовній базі даних MongoDB.

Щоб скористуватися механізмом оберненого геокодування можемо використати бібліотеку Geopy - клієнт Python для кількох популярних веб-служб геокодування[20], ми скористаємося службою Nominatim. Наступний код ілюструє роботу цієї бібліотеки в нашій системі:

```
import geopy
from geopy.geocoders import Nominatim



def convert_street(lat, lon):
    locator = Nominatim(user_agent="myGeocoder")
    location: geopy.location.Location
    location = locator.reverse('{}, {}'.format(lat, lon), language='en',
exactly_one=True)
```

```
    if 'city' in location.raw['address']:
        return location.raw['address']['city']
    else:
        raise AttributeError("make sure that you are in city or location
which supports this functionality")
```

Для роботи з базою даних PostgreSQL використаємо бібліотеку psycopg2. Щоб уникнути SQL ін’єкцій, хоча й небагато місць, де їх можна реалізувати, необхідно використати наступну конструкцію:

```
async def insert_or_update_participating(self, chat_id):
    sql = 'insert into user_vote(vote_id, date_of_answer, user_id)
values(default, default, %s)'
    cursor = self.conn.cursor()
    try:
        cursor.execute(sql, [chat_id])
        self.conn.commit()
    except Exception:
        self.conn.rollback()
        cursor.close()
        raise Exception
    cursor.close()
```

Згідно з документацією psycopg, правильний спосіб передачі змінних у команді SQL – використання другого аргументу методу Cursor.execute(), адже Psycopg може автоматично перетворювати об’єкти Python на значення SQL.

На цих прикладах було показано спосіб реалізації боту в нашій інформаційній системі.

3.4. Розробка клієнтської частини додатку.

Як вже зазначалося, для реалізації клієнтської частини додатку буде використана бібліотека React JS та технологія Redux, Redux Thunk для зручнішого підходу підтримки станів компонентів. Redux – це бібліотека для

керування станом додатку, архітектура якої схожа на Flux, проте відмінність в тому, що Redux зберігає стан усіх компонентів додатку в єдиному об'єкті – контейнері станів, який має назву store замість збереження цих станів у кількох контейнерах, що є характерним для інших бібліотек. [21]

Для створення клієнтської частини додатку з використанням бібліотеки React JS необхідно створити index.js файл та скористуватися наступною конструкцією:

```
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

BrowserRouter – компонент необхідний для роботи з бібліотекою ReactRouter, яка дозволяє прив'язувати шляхи в браузері до певних компонентів React.

Приклад прив'язування шляху до компонентів:

```
<Route
              path="/users"
              element={
                <SecureRoute requeredRole="ADMIN">
                  <UsersInfoPage></UsersInfoPage>
                  <MainFooter></MainFooter>
                </SecureRoute>
              }
            ></Route>
```

Для того щоб працювати з React Redux, необхідно створити store:

```
const initialState = {};
const middleware = [thunk];

let store;
```

```
if (window.navigator.userAgent.includes("Chrome")) {
  store = createStore(
    rootReducer,
    initialState,
    compose(
      applyMiddleware(...middleware),
      window.__REDUX_DEVTOOLS_EXTENSION__ &&
        window.__REDUX_DEVTOOLS_EXTENSION__()
    )
  );
} else {
  store = createStore(
    rootReducer,
    initialState,
    compose(applyMiddleware(...middleware))
  );
}
```

У цьому прикладі ми також застосовуємо middleware – Redux Thunk, який дозволяє працювати з асинхронними діями (async actions) та rootReducer – це комбінація усіх редукторів у нашому додатку:

```
export default combineReducers({
  errors: errorReducer,
  mapPoints: mapReducer,
  security: authReducer,
  getPoints: getDataFromServerReducer,
  loading: loadingReducer,
  userList: usersListReducer,
});
```

Приклад редуктору, який відповідає за обробку помилок:

```
const initialState = {};

export const errorReducer = (state = initialState, action) => {
  switch (action.type) {
    case GET_ERRORS:
      return action.payload;
    case SET_NO_ERRORS:
      return {};
```

```
    default:
      return state;
  }
};
```

У Redux Редуктори можуть бути використані у діях (actions), наступний код відповідає за можливість ініціювання створення нового міста сервером з боку клієнтської частини. Саме у діях ми можемо відправити get/post/patch/put/delete запити на наш сервер:

```
export const addCityInfo = (navigate, city) => async (dispatch) => {
  dispatch(mainLoading(true));
  try {
    await axios.post("/admin/addCity", city);
    navigate("/main");
  } catch (e) {
    dispatch({
      type: GET_ERRORS,
      payload: e.response.data,
    });
  }
  dispatch(mainLoading(false));
};
```

У цьому прикладі за допомогою бібліотеки axios ми відправляємо запит на збереження інформації про нове місто, якщо виникли якісь помилки – оброблюємо їх за допомогою редуктору.

Для роботи з картою скористуємося бібліотекою react-google-maps/api, а також, щоб не навантажувати карту, скористаємося хуком useSuperclaster, щоб об’єднати багато географічно близьких між собою точок в один кластер:

```
 const { clusters } = useSupercluster({
    points: locationCoordinates,
    bounds,
    zoom,
    options: {
      radius: 100,
      maxZoom: 20,
    },
```

```
  });
```

Для рендерінгу кластерів скористаємося наступною конструкцією:

```
<GoogleMap [params1={…},…]>
…
{clusters.map((cluster, id) => {
          const [lng, lat] = cluster.geometry.coordinates;
          const { cluster: isCluster, point_count: pointCount } =
            cluster.properties;
          if (isCluster) {
            return (
              <ClusterMarker
                key={id}
                position={{ lng, lat }}
                pointCount={pointCount}
              ></ClusterMarker>
            );
          }
…
</GoogleMap>
```

За допомогою станів компоненту ми зможемо відслідковувати взаємодію (використовуючи [params1={…}, …], які в дійсності мають назви onZoomChanged, onDragEnd і тп.) з картою та генерувати відображення кластерів з новими параметрами bounds, zoom.

3.5. Демонстрація роботи додатку.

Спершу необхідно зареєструватися у боті, зробити це можна, якщо виконати команду /start_locate (рис. 3.3).

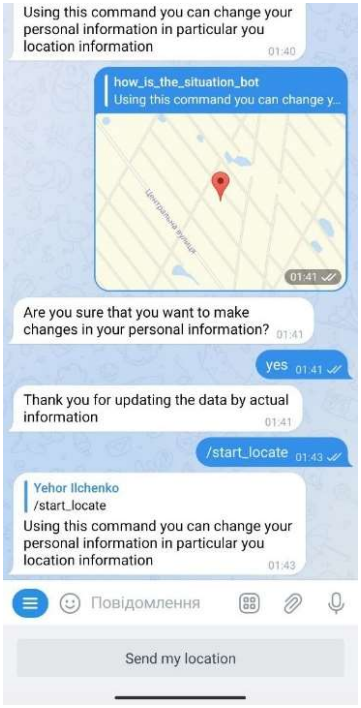


Рисунок 3.3 - спроба оновити дані

Як можна помітити, система намагається дізнатися місцеположення користувача (Рисунок 3.4).

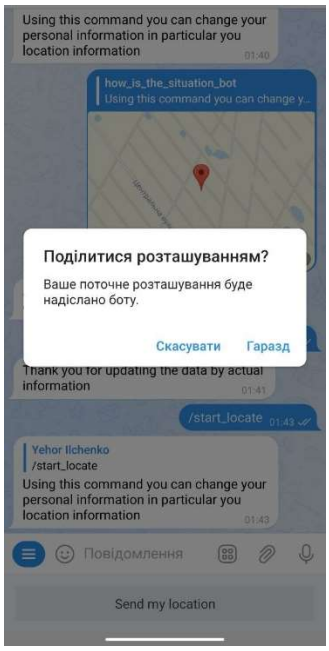


Рисунок 3.4 - запит місцеположення

Якщо вибрати місцерозташування під яке не налаштована система, бот повідомить користувачу про це (Рисунок 3.5). Все це працює автоматично, бо використовується технологія оберненого геокодування.

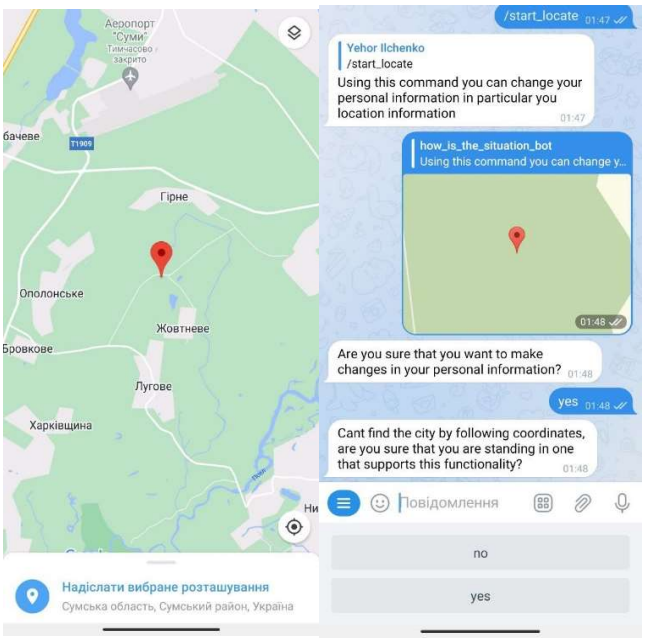


Рисунок 3.5 - місцеположення вибрано невдало

Якщо ж користувач надав місцеположення в межах міста, яке є в інформаційній системі, то отримає позитивну відповідь про зміну даних (рис. 3.6).

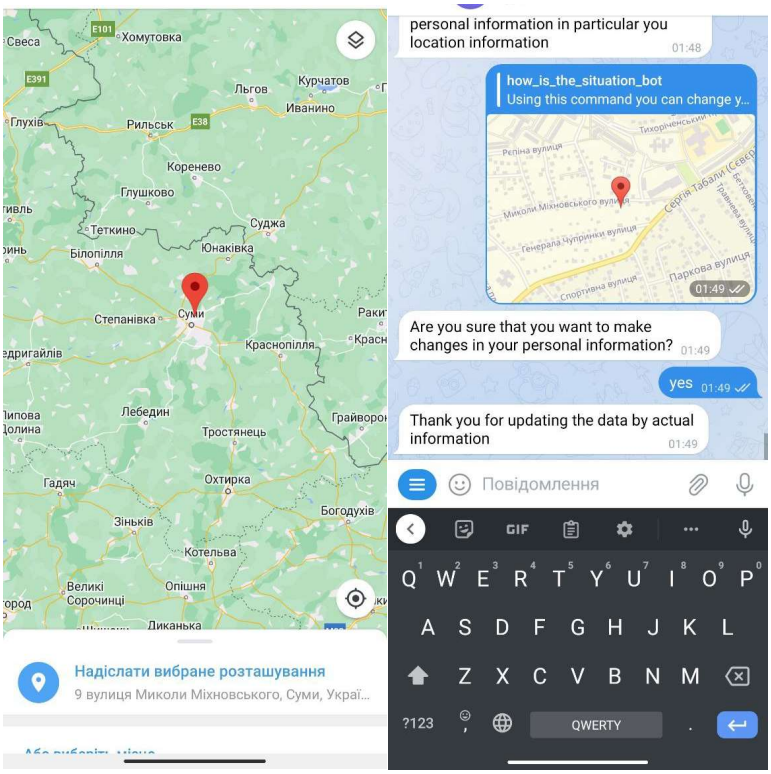


Рисунок 3.6 - вибране місцеположення, яке знаходиться в межах міста, яке є в системі

Після вказання цієї інформації користувачу надходитимуть запрошення прийняти участь у голосуванні (рис. 3.7).

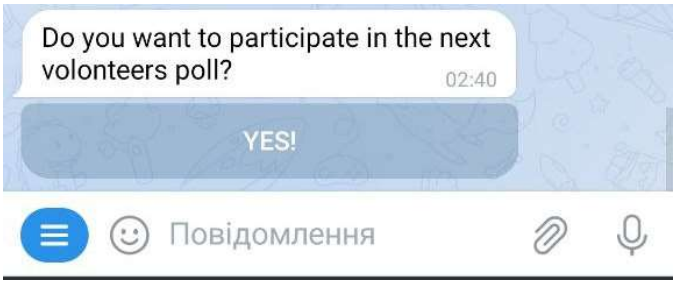


Рисунок 3.7 - запрошення прийняти участь у голосуванні

Проаналізуємо головну сторінку додатку (рис. 3.8).

Рисунок 3.8 - головна сторінка додатку

Зайти на цю сторінку зможе тільки користувач з правами адміністратору. На цій сторінці розміщена карта та вказівки про те, як працювати з нею. На цій карті з’являються усі користувачі які натиснули кнопку «YES» у запрошенні прийняти участь у голосуванні. Але основний функціонал системи розміщений у панелі зліва від карти (рис. 3.9).

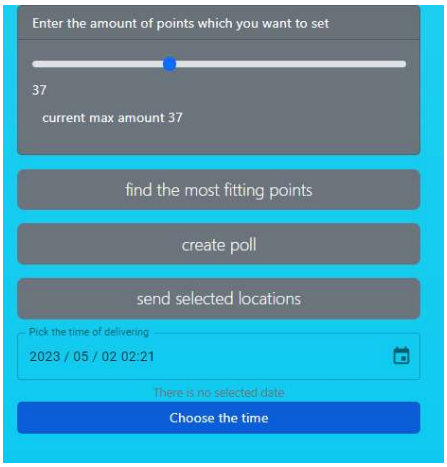


Рисунок 3.9 - інтерфейс для роботи з опитуваннями

На цій панелі можна обрати кількість вантажів (розділ «Enter the…»), які потенційно можуть бути позначені на карті як точки, в яких будуть розміщуватися центри надання гуманітарних вантажів.

Кнопка create poll дозволяє проводити опитування, вміст повідомлення продемонстрований на рис. 1.7. Після того як користувач натиснув кнопку «YES» у запрошенні на участь у голосуванні, місцеположення, яке він вказував у реєстрації буде показано на карті (рис. 3.10)

.

Рисунок 3.10 - моніторинг ситуації за містом (згенерована велика кількість адрес)

Кнопка «find the most fitting points» - дозволяє знайти найбільш вигідні точки розташування вантажів, вона є опціональною, адже алгоритм може давати незадовільні для користувачів результати і не є необхідним для задач моніторингу потреб населення, робота продемонстрована на рисунку 3.11.

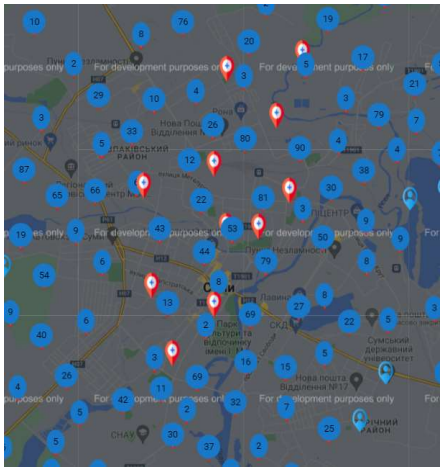


Рисунок 3.11 - підбір найбільш вдалих точок розташування

Користувач може додавати маркери на карті – потенційні точки центрів надання гуманітарної допомоги і самостійно (рис. 3.12).

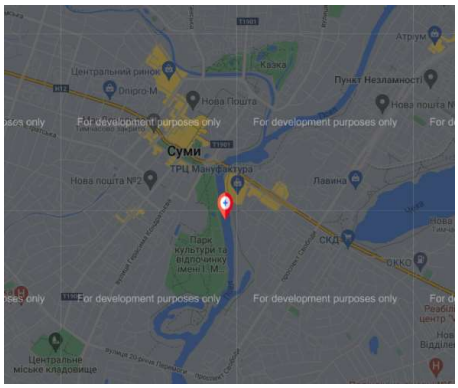


Рисунок 3.12 - можливість вибору точки

Після чого вибрати приблизний час доставки вантажів «pick the time of delivering» (рис. 3.9) і натиснути кнопку «Send selected locations», що відправить кожному користувачу одну з вибраних точок, яка знаходиться географічно найближче до нього (рис. 3.13).

Рисунок 3.13 - результат вибору точки розташування вантажу і відправки її користувачам

У системі можна додати дані про нове місто (рис. 3.14).

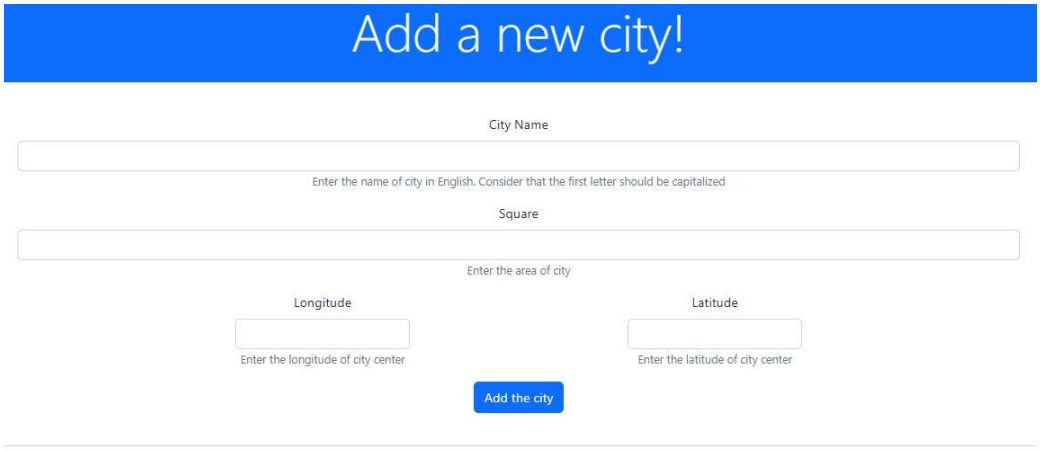


Рисунок 3.14 - додавання інформації про нове місто

Також користувачі з роллю адміністратору можуть вибирати місто, в якому проводити моніторинг потреб населення.

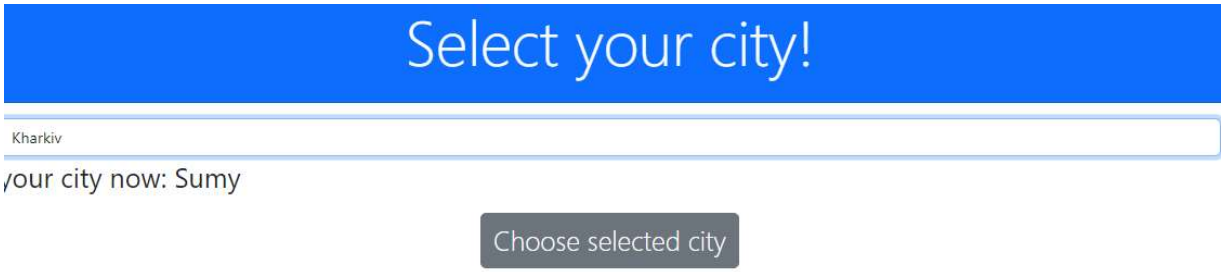


Рисунок 3.15 - вибір міста

Додаткову інформацію звичайні користувачі системи можуть подивитися в розділі «additional info» (рис. 3.16)

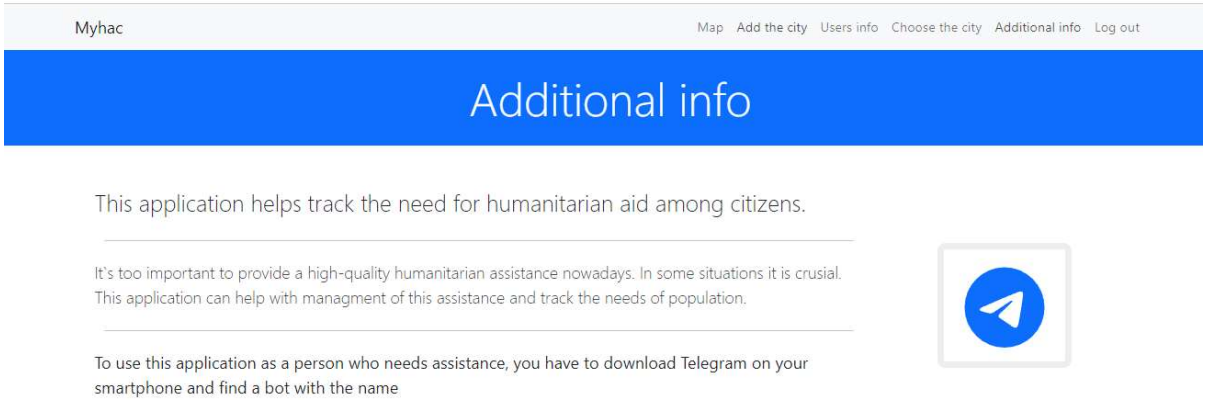


Рисунок 3.16 - додаткова інформація для користувачів

З демонстрації роботи додатку видно, що увесь функціонал системи моніторингу був реалізований у зручному і інтуїтивно-зрозумілому для користувача інтерфейсі.

ВИСНОВКИ

В результаті літературного огляду було виявлено ряд слабких сторін систем, які дозволяють збирати інформацію про гуманітарні потреби населення. Зокрема існують проблеми в моніторингу гуманітарних потреб в необхідних часових проміжках. Саме на вирішення цієї проблеми націлена розроблена інформаційна система.

Було проаналізовано архітектури веб-систем, мов програмування, які підходять для розробки веб-додатків, сучасних фреймворків, систем управління базами даних, та обрано найбільш підходящий для вирішення наших задач набір технологій. Внаслідок огляду було визначено оптимальні інструменти для створення системи: в якості мов програмування - Java та Spring Framework, Python та Aiogram; в якості баз даних - PostgreSQL, Mongo DB, ReactJS. А також було обрано архітектурний стиль - REST.

Аналіз існуючих рішень дозволив сформувати вимоги до додатку, які наведені у розділі 1.4 Постановка задачі (див. ст. 15) . Вказані функціональні вимоги були реалізовані у повному обсязі, приклад їх використання наведено на рисунках 3.1-3.16.

Для демонстрації роботи та архітектури додатку було створено діаграми use-case, ER для створення архітектури додатку, продемонстровано алгоритм пошуку точок найкращого розташування гуманітарних вантажів, приведені скріншоти роботи додатку.

В результаті, розроблена система може проводити опитування, збирати результати опитувань і надавати їх у вигляді маркерів на інтерактивній мапі, вибирати точки найбільш вдалих розташувань гуманітарних вантажів і надсилати цю інформацію користувачам, які її потребують.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Кондрашова, С.С. Інформаційні технології в управлінні [Текст] : Навч. посібник / С.С.Кондрашова . — К. : МАУП, 1998. — 560 с.
2. Гуманітарна катастрофа чи гуманітарна голка – дві сторони однієї медалі: доступ до гуманітарної допомоги в умовах збройного конфлікту на сході України / О.А. Біда, А.Б. Блага, О.А. Мартиненко, М. Г. Статкевич; за заг. ред. А.П. Бущенка / Українська Гельсінська спілка з прав людини. – К., КИТ, 2016. – 54 р. – с. 4 [Електронний ресурс] / Режим доступу: https://helsinki.org.ua/wpcontent/uploads/2016/06/Press_Humanitarian_aid_reportUKR.pdf
3. Механізми надання державою гуманітарної допомоги в умовах воєнного стану / С. Деркач та ін. Київ : Нац. агенство з питань запобігання корупції, 2022. 75 с. URL: https://nazk.gov.ua/wp-content/uploads/2022/07/Mehanizmy-nadannya-derzhavnoyu-gumanitarnoyi-dopomogy-v-umovah-voyennogo-stanu.pdf (дата звернення: 10.05.2023).
4. Карпусь В. Створений Telegram-бот SaveUA для координації волонтерської допомоги [Електронний ресурс] / Вадим Карпусь // itc.ua. – 2022. – Режим доступу до ресурсу: https://itc.ua/news/sozdan-telegram-bot-saveua-dlya-koordinaczii-volonterskoj-pomoshhi/.
5. Рейтинг мов програмування 2022 [Електронний ресурс] // Редакція DOU. – 2022. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/language-rating-2022/.
6. Wozniewicz B. The Difference Between a Framework and a Library [Електронний ресурс] / Brandon Wozniewicz. – 2019. – Режим доступу до ресурсу: https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/.

7. Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-framework.
8. Spring Modules [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/spring-modules.
9. Silnitsky N. How to choose the right database for your service [Електронний ресурс] / Natan Silnitsky. – 2021. – Режим доступу до ресурсу: https://medium.com/wix-engineering/how-to-choose-the-right-database-for-your-service-97b1670c5632.
10. Dharmendra K. Best Databases To Use In 2022 [Електронний ресурс] / Kumar Dharmendra. – 2022. – Режим доступу до ресурсу: https://hevodata.com/learn/best-database/.
11. Calculate distance, bearing and more between Latitude/Longitude points [Електронний ресурс] // Movable Type Scripts – Режим доступу до ресурсу: https://www.movable-type.co.uk/scripts/latlong.html.
12. Lai J. Z., Huang T.-J., Liaw Y.-C. A fast k-means clustering algorithm using cluster center displacement. Pattern recognition. 2009. Т. 42, № 11. С. 2551-2556. URL: https://doi.org/10.1016/j.patcog.2009.02.014.
13. Arthur D., Vassilvitskii S. K-Means++: the advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms*. USA, 2007. С. 1027–1035.
14. Schubert E., Rousseeuw P. J. Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. Similarity search and applications / ред.: G. Amato та ін. Cham, 2019. С. 171--187.
15. Clustering large applications (program CLARA). *Finding groups in data*. 1990. С. 126-163. URL: https://doi.org/10.1002/9780470316801.ch3.
16. REST: From Research to Practice - Google книги: / за ред. Erik Wilde, Cesare Pautasso. London: Springer Science, 2011. 21–22с.
17. Jones M. JSON Web Token (JWT) [Електронний ресурс] / M. Jones, J. Bradley, N. Sakimura // Internet Engineering Task Force. – 2015. – Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc7519.

18. Walls, C. Spring Boot in action. Manning, 2015.
19. Lee D., Yannakakis M. Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE. 1996. Т. 84, № 8. С. 1090-1123. URL: 10.1109/5.533956.
20. GeoPy’s documentation. URL: https://geopy.readthedocs.io/ (дата звернення: 01.05.2023).
21. Garreau M., Faurot W. Redux in action. Manning, 2018. URL: https://books.google.com.ua/books?id=CTgzEAAAQBAJ.

ДОДАТКИ

Додаток А. Лістинг програмного коду

```
@RestController
@AllArgsConstructor
//@PreAuthorize("ADMIN")
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping("/admin")
@Slf4j
public class MainController {
   private final MapValidationErrorService mapValidationErrorService;

   @Qualifier("kmeansImplementationUserVotesService") private final
UserVotesService userVotesService;
   private final TelegramBotPushingService telegramBotPushingService;
   private final LogHistoryService logHistoryService;
   private final LastPollAndSecurityCheckerService
pollAndSecurityCheckerService;
   private final CityService cityService;
   private final DeliveryTimerService deliveryTimerService;
   private final CustomUserDetailsService userDetailsService;

   @GetMapping("/getVotes")
   public ResponseEntity<?> getAll(@RequestParam String city){
      City currentCity= cityService.getCityByName(new CityName(city));
      return ResponseEntity.ok( userVotesService.getCoordinates(currentCity));
   }
   @GetMapping("/getVotesCount")
   public ResponseEntity<?> getVoteCount(@RequestParam String city){
```

```
        City currentCity= cityService.getCityByName(new CityName(city));
        return ResponseEntity.ok( userVotesService.getCountByCity(currentCity));
    }
    @GetMapping("/getLocationsByPeriod")
    public ResponseEntity getLocationsByPeriod(@RequestParam String cityName,
@RequestParam LocalDate startDate, @RequestParam LocalDate endDate){
        return null;
    }

    @GetMapping("/getCity")
    public ResponseEntity getCity(@RequestParam String city){
        City city1 =  cityService.getCityByName(new CityName(city));
        return ResponseEntity.ok(city1);
    }
    @PatchMapping("/resetCity")
    public ResponseEntity updateTheCity(@RequestBody ObjectNode
objectNode){
        String city = objectNode.get("city").asText();
        String username = objectNode.get("username").asText();
        City selectedCity = cityService.getCityByName(new CityName( city));
        JwtUserDetails userDetails = (JwtUserDetails)
userDetailsService.loadUserByUsername(username);
        userDetails.setCity(selectedCity);
        userDetailsService.updateUser(userDetails);
        return ResponseEntity.ok().build();
    }
    @PostMapping("/sendLocation")
    public ResponseEntity sendLocations(@RequestBody SelectedLocationsDTO
selectedLocations, BindingResult bindingResult) throws
TelegramSendMessageError {
```

```
        ResponseEntity errors =
mapValidationErrorService.mapErrors(bindingResult);
        if(errors!=null){
            return errors;
        }

        City city =  cityService.getCityByName(new
CityName(selectedLocations.getCityName()));
        long diff =
pollAndSecurityCheckerService.getLastSendMessageExpirationDifferance(city);
        if(diff<0){
            //TODO UNCOMMENT IN RELEASE
            return mapValidationErrorService.getErrorAsMap("sendLocationTimeout",
String.valueOf(-diff));
        }
        telegramBotPushingService.pushMessagesToUsers(city,
selectedLocations.getCoordinatesList());

logHistoryService.LogLocationSending(DtoConverterUtils.convertSelectedLocati
ons(selectedLocations));
        log.info(selectedLocations.toString());
        log.info(city.toString());
        deliveryTimerService.updateNextTimeDelivery(city,
selectedLocations.getTimeOfDelivering());
        return ResponseEntity.ok("locations was successfully sent");
    }

    @GetMapping("/getBestPoints")
    public ResponseEntity<?> getBestFittingPoints(@RequestParam int
amountOfPoints, @RequestParam String cityName) throws
```

```
IllegalAccessException {
        if(amountOfPoints<=0){
            throw new WrongAmountException("the number of requested points is not
positive");
        }
        City city = cityService.getCityByName(new CityName(cityName));
            List<LocationCoordinates> locationCoordinatesList =
userVotesService.getFittedCoordinatesByLocation(city, amountOfPoints);
            return ResponseEntity.ok( locationCoordinatesList);
    }

    @GetMapping("/sendMessage")
    public ResponseEntity<?> sendMessage(@RequestParam String city) throws
TelegramSendMessageError {
        CityName cityName = new CityName(city);
        City cityObj = cityService.getCityByName(cityName);
        long diff =
pollAndSecurityCheckerService.getLastPollingExpirationDifferance(cityObj);
        if(diff<0){
            //TODO UNCOMMENT IN RELEASE
            return mapValidationErrorService.getErrorAsMap("pollingTimeout",
String.valueOf(-diff));
        }
        telegramBotPushingService.createPoll(cityObj);
        return ResponseEntity.ok("your poll was successfully send");
    }
}
```

@Data
@AllArgsConstructor

```
public class CityDTO {
   private LocationCoordinates locationCoordinates;
   private String name;
   private double area;
}
```

```
      @AllArgsConstructor
@Data
public class CityName {
   private String name;
}
```

```
      @AllArgsConstructor
@Data
public class SelectedLocationsDTO {
   private String adminUsername;
   private List<LocationCoordinates> coordinatesList;
   private Timestamp timeOfDelivering;
   private long amountOfPoints;
   private String cityName;

}
```

```
      @Controller
@AllArgsConstructor
@Slf4j
@CrossOrigin
@RequestMapping("/volunteers")

public class VolunteerController {

   private final CityService cityService;
   private final MapValidationErrorService errorService;
```

```
    @GetMapping("/getCities")
    public ResponseEntity getAllCities(){
        return ResponseEntity.ok(cityService.getAllCities());
    }
}
```

```
        @RestController
@ControllerAdvice
@AllArgsConstructor
@Slf4j
public class CustomResponseEntityExceptionHandler extends
ResponseEntityExceptionHandler {
    private final String BASE_ERROR = "baseError";
    private final String BASIC_MESSAGE = "something went wrong while we
sending your message";
    private final String CITY_ERROR = "cityError";
    private final String CITY_ERROR_MESSAGE = "city does not exist";
    private final String AMOUNT_ERROR = "amountError";
    private final String AMOUNT_ERROR_MESSAGE = "amount is not set
propertly";
    private final String TELEGRAM_ERROR = "telegramError";
    private final String TELEGRAM_ERROR_MESSAGE = "something went
wrong while we sending your message";

    private final String USERNAME_NOT_FOUND = "usernameNotFoundError";

    private final String AUTHORITY_NOT_FOUND = "authorityNotFoundError";
    private final String AUTHORITY_NOT_FOUND_MESSAGE = "cant find a
given authority to add it!";
```

```
    private final String USERNAME_NOT_FOUND_MESSAGE = "cant find the
user with a given email";
    private final MapValidationErrorService mapValidationErrorService;

    @ExceptionHandler(UsernameAlreadyExistException.class)
    public final ResponseEntity<Object>
handleUsernameAlreadyExistException(UsernameAlreadyExistException ex,
WebRequest request){
        UsernameAlreadyExist exept = new
UsernameAlreadyExist(ex.getMessage());
        log.warn(ex.getMessage());
        return new ResponseEntity<>(exept, HttpStatus.BAD_REQUEST);
    }
    @ExceptionHandler(UsernameNotFoundException.class)
    public final ResponseEntity<Object>
handleUsernameNotFoundException(UsernameAlreadyExistException ex,
WebRequest request){
        log.warn(ex.getMessage());
        return (ResponseEntity<Object>)
mapValidationErrorService.getErrorAsMap(USERNAME_NOT_FOUND,
USERNAME_NOT_FOUND_MESSAGE);
    }

    @ExceptionHandler(AuthorityNotFoundException.class)
    public final ResponseEntity<Object>
handleAuthorityNotFoundException(AuthorityNotFoundException ex,
WebRequest request){
        log.warn(ex.getMessage());
        return (ResponseEntity<Object>)
mapValidationErrorService.getErrorAsMap(AUTHORITY_NOT_FOUND,
```

```
AUTHORITY_NOT_FOUND_MESSAGE);
    }

    @ExceptionHandler(CityNotFoundException.class)
    public final ResponseEntity<Object>
handleCityNotFoundException(CityNotFoundException ex, WebRequest request){
        log.warn(ex.getMessage());
        return (ResponseEntity<Object>)
mapValidationErrorService.getErrorAsMap(CITY_ERROR,
CITY_ERROR_MESSAGE);
    }

    @ExceptionHandler(TelegramSendMessageError.class)
    public final ResponseEntity
handleTelegramSendMessageError(TelegramSendMessageError error,
WebRequest request){
        log.error("Telegram send message error", error);
        return
mapValidationErrorService.getErrorAsMap(TELEGRAM_ERROR,TELEGRAM_
ERROR_MESSAGE);
    }

    @ExceptionHandler(WrongAmountException.class)
    public final ResponseEntity
handleWrongAmountException(WrongAmountException err, WebRequest
request){
        log.warn("amount error likely in main controller class",err.getMessage());
        return
mapValidationErrorService.getErrorAsMap(AMOUNT_ERROR,AMOUNT_ERR
OR_MESSAGE);
```

```
    }

    @ExceptionHandler(IllegalAccessException.class)
    public final ResponseEntity
handleIllegalAccessException(IllegalArgumentException ex, WebRequest
request){
        log.warn(ex.getMessage());
        return
mapValidationErrorService.getErrorAsMap(BASE_ERROR,BASIC_MESSAGE
);
    }
}
        @AllArgsConstructor
@Service("kmeansImplementationUserVotesService")
public class UserVotesServiceKMeansImpl implements UserVotesService {

    private final CitiesRepo citiesRepo;
    private final UserVotesRepository userVotesRepository;

    @Override
    public List<LocationCoordinates> getCoordinates(City city) {
        if(city == null){
            throw new NullPointerException("city cant be null!");
        }
        //citiesRepo.findById(city.getName()).orElseThrow(() -> {return new
NullPointerException("city does not exist!");});
        return
userVotesRepository.getUserVotesByActiveAndChatLocation_CityName(true
,city).stream().map(e->{return
e.getChatLocation().getLocationCoordinates();}).collect(Collectors.toUnmodifiabl
```

```
eList());
    }

    @Override
    public Long getCountByCity(City city) {
        if(city == null){
            throw new NullPointerException("city cant be null!");
        }
        //citiesRepo.findById(city.getName()).orElseThrow(() -> {return new
NullPointerException("city does not exist!");});
        return
userVotesRepository.countUserVotesByActiveAndChatLocation_CityName(true,
city);
    }

    @Override
    public List<LocationCoordinates> getFittedCoordinatesByLocation(City city,
int amountOfLocations) throws IllegalAccessException {
        if(amountOfLocations <=0){
            throw new IllegalAccessException("amount of locations cant be negative");
        }
        ua.edu.sumdu.volonteerProject.model.City authorizedCity =
citiesRepo.findById(city.getName()).orElseThrow(() -> new
IllegalAccessException("city is not found"));
        List<UserVote> chatLocations =
userVotesRepository.getUserVotesByActiveAndChatLocation_CityName(true,
authorizedCity);
        return  ClusterService.cluster(chatLocations.stream().map(e-
>e.getChatLocation().getLocationCoordinates()).collect(Collectors.toList()),
amountOfLocations).stream().map(e-
```

```
>(LocationCoordinates)e).collect(Collectors.toList());
    }
}
        @AllArgsConstructor
@Service
@Slf4j
public class TelegrmBotPushingServiceImpl implements
TelegramBotPushingService {
    private final ChatLocationRepository chatLocationRepository;
    private final CitiesRepo citiesRepo;
    private final UserVotesRepository userVotesRepository;
    private final LastPollAndSendCityCheckerRepo pollAndSendCityCheckerRepo;
    private final String MESSAGE = "Do you want to participate in the next
volonteers poll?";
    private final String REPLY_MESSAGE = "YES!";

    private TelegramBot telegramBot;

    @Transactional
    @Override
    public void pushMessagesToUsers(City city, List<LocationCoordinates>
locationCoordinates) throws TelegramSendMessageError {
        List<UserVote> chatLocations =
userVotesRepository.getUserVotesByActiveAndChatLocation_CityName(true,city
);
        Map<Long, LocationCoordinates> chatsAndLocations = new HashMap<>();
        log.debug("locations sent to chats:" + chatLocations.toString());
        log.debug("sent locations: " + locationCoordinates.toString());
        chatLocations.stream().parallel().forEach(e -> {
```

```
            chatsAndLocations.put(e.getChatLocation().getChatId(),
locationCoordinates.stream().min((a, b) -> {
                if(a.equals(b)){
                    return 0;
                }
                return

(CoordinateUtils.calculateDistance(e.getChatLocation().getLocationCoordinates(),
a) -
CoordinateUtils.calculateDistance(e.getChatLocation().getLocationCoordinates(),
b))<=0?-1:1;}).orElse(null));
            });
        pollAndSendCityCheckerRepo.updateSendDateByCity(city.getName());
        telegramBot.sendLocations(chatsAndLocations);

chatLocationRepository.setHasInvitedToFalseForVotedLocationsByCity(city);
        userVotesRepository.inactivateUserVoteByCity(city);
    }

    @Transactional
    @Override
    public void createPoll(City city) throws TelegramSendMessageError {
        List<Long> doesNotAnsweredThePrevPollUsers =
chatLocationRepository.getChatIdBy(city, true);
        List<Long> answeredThePrevPollUsers =
chatLocationRepository.getChatIdBy(city, false);
        pollAndSendCityCheckerRepo.updatePollDateByCity(city.getName());

telegramBot.sendMessagesWithInlineBrd(answeredThePrevPollUsers,MESSAGE,
REPLY_MESSAGE);
```

```
        telegramBot.sendMessage(doesNotAnsweredThePrevPollUsers, "You didn`t answer the in the previous poll, the next one has been startred, so you can vote this time, using the previous button!");

        chatLocationRepository.setHasInvitedToFalse(city,true);

    }
}
```

@Service
@AllArgsConstructor

```
public class DtoConverterUtils {
    private final CitiesRepo citiesRepo;

    public static City convertCity(CityDTO cityDTO) {
        return new City(cityDTO.getName(),cityDTO.getLocationCoordinates(), cityDTO.getArea());
    }

    public static UserInfoDTO convertUserInfo(JwtUserDetails jwtUserDetails){
        String cityName = jwtUserDetails.getCity()!=null?jwtUserDetails.getCity().getName():null;
        return new UserInfoDTO(jwtUserDetails.getUsername(),jwtUserDetails.getFirstName(), jwtUserDetails.getSecondName(), jwtUserDetails.getId(), jwtUserDetails.getAuthorityList().stream().toList(), jwtUserDetails.isBlocked(), cityName);
    }
    public JwtUserDetails convertUserDetails(UserDTO userDTO){
        Authority a=  new Authority();
        City city = null;
```

```
        if(userDTO.getCityName()!=null) {
            city = citiesRepo.findById(userDTO.getCityName()).get();
        }
        return new JwtUserDetails(
                new HashSet<>(),
                city,
                userDTO.getPassword()
                ,userDTO.getFirstName()
                ,userDTO.getSecondName()
                ,null
                ,null
                ,null
                ,userDTO.getUsername()
                ,false

        );
    }

    public static SendLocationsDetails
convertSelectedLocations(SelectedLocationsDTO selectedLocations) {
        return new SendLocationsDetails(selectedLocations.getAdminUsername(),
null, selectedLocations.getAmountOfPoints(), selectedLocations.getCityName(),
selectedLocations.getTimeOfDelivering() ,null);
    }
}
```