

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБМІНУ

(назва теми)

ЗАПИТАМИ З СУСІДАМИ. SOFTWARE FOR EXCHANGING

REQUESTS WITH NEIGHBORS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-120

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ДЕГТЯРЬ А.Ю.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ КНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ бакалавр _____
Спеціальність _____ 121 Інженерія програмного забезпечення _____
(код і найменування)
Освітня програма (спеціалізація) _____ Інженерія програмного забезпечення _____
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф. _____
Сергій СУББОТІН
« _____ » _____ 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ДЕГТЯРЯ Артема Юрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для обміну запитами з сусідами
Software for Exchanging Requests with Neighbors

керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 24 » квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 15 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

_____ Артем ДЕГТЯРЬ
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Валерій ЛЬОВКІН
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 90 с., 24 рис., 3 табл., 3 дод., 8 джерел.

ОБМІН ЗАПИТАМИ, СУСІДИ, РЕЄСТРАЦІЯ ПРОЖИВАННЯ, КОЛА СПІЛКУВАННЯ, ВЕБДОДАТОК.

Об'єкт роботи – процес комунікації з сусідами. Предмет роботи – програмне забезпечення для обміну запитами з сусідами. Мета роботи – розробити програмне забезпечення для обміну запитами з сусідами, щоб підвищити інформованість сусідів та забезпечити обмеження доступу на таку інформацію.

Матеріали, методи та технічні засоби: мультипарадигмальні засоби програмування мовою Python в комплексі з фреймворком Django, системою підтримки баз даних MySQL.

Результати. Програма для обміну запитами з сусідами надає можливість комунікувати користувачам, які проживають в безпосередній близькості один від одного, обмінюватися інформаційними повідомленнями, відповідати на запитання один одного, звертати увагу один одного, спільно вирішувати проблеми, забезпечуючи для цього як персональну комунікацію з можливістю обмеження її, так і комунікацію в колах спілкування, які враховують принцип сусідства.

Висновки. Створена в підсумку програма дозволяє не тільки обмінюватися запитами з сусідами, але і впорядковувати цей процес для забезпечення обмеження доступу на інформацію, уникнення потрапляння такої інформації до осіб, які могли б її використовувати зі шкідливими цілями в подальшому.

Галузь використання. Об'єднання людей за принципом місця проживання.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 90 pages, 24 figures, 3 tables, 3 appendixes, 8 sources.

EXCHANGING REQUESTS, NEIGHBORS, RESIDENCE REGISTRATION, COMMUNICATION CIRCLES, WEB APP.

The object of the work is the process of communication with neighbors. The subject of the work is software for exchanging requests with neighbors. The goal of the work is to develop software for exchanging requests with neighbors to increase awareness of neighbors and ensure access restrictions to such information.

Materials, methods and technical means: multi-paradigm programming tools in the Python language in combination with the Django framework, the MySQL database support system.

The results. The program for exchanging requests with neighbors provides an opportunity to communicate with users who live in close proximity to each other, exchange information messages, answer each other's questions, pay attention to each other, solve problems together, providing personal communication with the possibility of limiting it, as well as communication in communication circles that take into account the principle of neighborhood.

Conclusions. The program allows not only to exchange requests with neighbors, but also to organize this process in order to ensure the restriction of access to information, to prevent such information from reaching persons who could use it for malicious purposes in the future.

Field of use. Association of people based on the principle of residence place.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Огляд програм для обміну запитами з сусідами.....	10
1.2 Порівняння аналогів для визначення характеристик власної розробки.....	11
1.3 Висновки за розділом 1	13
2 Проєктування програми.....	14
2.1 Вибір засобів проєктування і розробки	14
2.2 Моделювання взаємодії користувачів з програмою.....	15
2.3 Визначення структури бази даних	19
2.4 Висновки за розділом 2	30
3 Розробка програми	31
3.1 Визначення порядку роботи з програмою.....	31
3.2 Особливості програмної реалізації.....	32
3.3 Висновки за розділом 3	42
4 Експлуатація та тестування програми	43
4.1 Призначення програми	43
4.2 Умови виконання програми	43
4.3 Експлуатація програми.....	43
4.4 Тестування програми.....	48
4.5 Висновки за розділом 4	51
Висновки	52
Перелік джерел посилання	53
Додаток А Технічне завдання	54
Додаток Б Текст програми	59
Додаток В Слайди презентації.....	83

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ДВВ – діаграма варіантів використання.

ВСТУП

Соціальні мережі відомі у всьому світі, впливають на розвиток людства, проте не завжди цей вплив є позитивним. Одним з таких негативних факторів є те, що люди стають менше спілкуватися в реальності, а частиною цієї проблеми є зменшення спілкування з сусідами. Це створює проблеми при переїзді, коли людина потребує адаптації, а значить підтримки оточуючих. Така підтримка потрібна через надання додаткової інформації, відповіді на певні запитання (запити). Окрім того часом потрібно розповсюдити інформацію між сусідами, що є важливою. Але в соціальних мережах друзі обираються не за принципом сусідства загалом. Тож достатньо корисним для забезпечення підтримки щодо окреслених проблем є врахування під час забезпечення комунікації принципу сусідства.

Більше того загалом для спілкування між сусідами описаний принцип потрібно не тільки враховувати, але і використовувати його в центрі забезпечення безпеки. Це пов'язано з тим, що спілкування між сусідами дозволяє розкрити особи, дозволяє отримати інформацію, яка може бути корисною для шахраїв. Тому принцип сусідства має бути не тільки основою для визначення участі в програмі та взаємодії з іншими, але і заборони на доступ до інформації, яка може бути критичною, ідентифікувати осіб.

Саме для надання такої концепції та забезпечення супутніх можливостей має бути направлена програма, яку потрібно створити в цій роботі.

Метою роботи є розробка програмного забезпечення для обміну запитамі з сусідами, щоб підвищити інформованість сусідів та забезпечити обмеження доступу на таку інформацію. Це потребує реалізації наступних завдань:

- проаналізувати проблему комунікації з сусідами на основі використання відповідних програм;

- спроектувати програмне забезпечення для обміну запитами з сусідами;
- реалізувати програмне забезпечення для обміну запитами з сусідами та забезпечити його тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд програм для обміну запитами з сусідами

Програми для обміну запитами з сусідами подібні до соціальних мереж, але використовуються в основі принцип близького проживання. На цьому принципі базується комунікація між користувачами. Програм, які б забезпечували на даний момент такий принцип, небагато.

Nextdoor [1] на даний момент використовується в 11 країнах світу, хоча і не є доступною в Україні, що фактично унеможлиблює її використання українськими користувачами [2]. Тому дана програма може розглядатися як концепція, але не як реально використовуваний на практиці інструмент для вирішення подібних задач користувачами в Україні.

Програма Nextdoor дозволяє створювати групи (сама така термінологія використовується в ній). Ці групи забезпечуються за адресним принципом. Перший користувач, який створює таку групу, визначає її межі, які в подальшому можуть змінюватися [3]. Користувачі, які знаходяться в місцях, які входять в межі групи, можуть до групи доєднуватися. Межі груп максимально можуть бути обмежені містом.

Користувачі можуть відправляти повідомлення один одному. Дозволяється відключити увідомлення про отримані повідомлення стосовно певних користувачів.

Програма має спеціальний напрямок для бізнесів, що дозволяє привертати до них увагу, надавати можливість пошуку потрібних бізнесів користувачам. Місцеві жителі таким чином можуть отримувати інформацію, комунікувати з представниками компаній, їх можна інформувати про зміни в компанії. Такі компанії не можуть бачити повідомлення, які створюють користувачі, вони можуть тільки бачити реакцію на їх повідомлення. Однак іншою стороною є те, що компанії менше витрачають часу на роботу в програмі в підсумку, а ось для звичайних користувачів програми це навпаки збільшує навантаження, адже кількість повідомлень зростає.

1.2 Порівняння аналогів для визначення характеристик власної розробки

Метою створення даної програми є підвищення інформованості сусідів та забезпечення обмеження доступу на таку інформацію. Це призвело до того, що серед аналогів був залишений тільки Nextdoor, який має подібну концепцію. У той же час інші програми, які також базуються на геолокації, направлені на забезпечення знайомств, звичайне спілкування. У такому випадку важливим є знаходження поруч, але це не передбачає, що люди обов'язково живуть поруч. У той же час, коли люди живуть поруч, у них виникає спільне коло проблем, які вони можуть потребувати вирішити. Але одночасно це пов'язано з тим, що вони мають персональні дані, які не варто розповсюджувати. Це і те, де вони проживають, і те, як їх звуть. Вся ця інформація є персональною, тому просто відправити повідомлення від одного користувача до іншого недостатньо. Потрібно враховувати правила доступу.

Якщо порівнювати між собою Nextdoor та програму Neirequestapp, яка в роботі розробляється, то слід зазначити, що між ними є певні важливі відмінності (табл. 1.1).

Окрім того, що Nextdoor взагалі не може бути використана в Україні, перспективи такого використання немає (як і для інших країні окрім 11 вже підтримуваних), існують додаткові відмінності. Так Nextdoor враховує геолокацію користувачів, однак при побудові груп дозволяється визначати їх межі, а максимальною межею є місто. Тому виходить, що певні групи можуть бути величезними, а до них включені далеко не тільки сусіди. Це сприяє розповсюдженню інформації з одного боку, але з іншого дещо підіймає питання безпеки доступу, адже достатньо широка кількість користувачів може взаємодіяти в групі.

Таблиця 1.1 – Порівняння існуючих аналогів та розроблюваного програмного забезпечення для обміну запитами з сусідами

Критерій	Nextdoor	Neirequestapp
Можливість участі в декількох групах сусідів	+	+
Варіанти комунікації	Персональне, в групах	Персональне, в групах
Підхід до визначення меж групи	Адміністратор з врахуванням геолокації, максимально – місто	Геолокація, тільки сусідні будинки
Об'єднання запитів	Гештеги	Типи
Підтвердження доступу користувача адміністратором	–	+
Реклама місцевого бізнесу	+	–

У Neirequestapp заплановано створити кола спілкування, обмеживши можливості доступу до них саме геолокацією. Базовим колом є коло, куди входять люди, які проживають в певному будинку. Це зручно для багатоквартирних будинків. Однак іншим варіантом створення груп є вже не базовий, а додатковий, коли адміністратор визначає тематику, але при цьому комунікація може відбуватися тільки з користувачами, які віддалені від адміністратора не більше ніж на 100 метрів. Відповідно між собою користувачі максимально віддалені на 200 метрів. Це призводить до того, що навіть якщо створюються додаткові групи, в них все одно беруть участь сусіди. Тобто базовий принцип зберігається.

Об'єднання запитів у Neirequestapp має відбуватися за типами. Основними типами є інформаційний, розважальний та безпековий. Це дозволяє користувачам одразу розуміти напрямок запиту та не витратити

додатковий час, якщо вони цього не хочуть. Для Nextdoor всі повідомлення розглядаються рівноцінно, хоча і доступні гештеги для звернень. Проте це призводить до того, що потрібно всі повідомлення розглядати разом. Нерідко користувачі можуть додавати просто власні фотографії, фотографії своїх домашніх улюбленців, а тому потрібно витрачати додатковий час, якого у людей може не бути, а тому і на важливіші проблеми вони в підсумку не відгукуються. В цьому сенсі розділення повідомлень на типи може бути достатньо корисним.

Neirequestapp повинен мати підтвердження доступу адміністратором. Це можливість, яка має додатково сприяти безпеці. Це направлено на те, щоб за необхідності адміністратор, який проживає за певною адресою, міг перевірити існування такої людини, її проживання за певною адресою. В іншому випадку все одно залишається можливість шахрайських дій, що часом і стається з користувачами в Nextdoor.

Окрім того Neirequestapp містить рекламу місцевого бізнесу. З одного боку це дозволяє привертати увагу користувачів до місцевих бізнесів, комунікувати додатково, але з іншого боку це призводить до наявності реклами, додаткової зайвої інформації, відповідно знову ж таки розпорошує увагу та не дозволяє досягати вузькоспрямованого використання для розв'язання тих проблем, які є нагальними саме для сусідів.

1.3 Висновки за розділом 1

У даному розділі розглянуто особливості проблеми комунікації з сусідами на основі використання відповідних програм стосовно підвищення інформованості сусідів та забезпечення обмеження доступу на таку інформацію. Основним аналогом розглянуто програму Nextdoor, але в підсумку розроблено певну концепцію, яка базується на визначеній меті та яка визначає особливості програми, яка розробляється.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Спершу вирішено виконувати розробку мовою Python, але далі необхідно було перевірити, чи точно ця мова достатня для веброзробки, а також вибрати той фреймворк, який би це дозволив реалізувати. Вибрався він з Django [4]-[5] як найбільш загально відомого та Tornado [6]-[7] як одного з фреймворків, які відзначаються швидкістю роботи (табл. 2.1). Tornado дійсно має відмінну швидкість роботи, але Django має достатню для реальних випадків використання швидкість, але до того ж кращу підтримку баз даних, наявну готову адміністративну панель, що дозволяє керувати будь-якими даними, які наявні в базі даних, кращу документацію і зокрема широту опису проблем та їх вирішення. Тому обрано Django, а його засоби підтверджують можливість і доцільність застосування Python.

Таблиця 2.1 – Порівняння фреймворків розробки програми для обміну запитами з сусідами

Критерій	Django	Tornado
Ліцензія	Відкрита, 3 пунктів	Відкрита, Apache License
Підтримка баз даних	Фреймворк об'єктно-реляційного відображення вбудований	Необхідно окремо налаштовувати, доступні фреймворки від інших розробників
Документація та простора застосування	Доступно багато варіантів, докладно описано	Менша наявної документації, окремі проблеми може бути не так легко вирішити
Керування через адміністративну панель	Вбудоване	Проблемне, немає готових для підключення засобів для реалізації

2.2 Моделювання взаємодії користувачів з програмою

Забезпечення неможливості неправомірного початкового доступу до керування колами спілкування запропоновано вирішувати за допомогою механізму скарг. Для підтримки цього серед користувачів було виділено адміністраторів програми. Тому всі користувачі є або звичайними користувачами, яких ще можна назвати сусідами, або адміністраторами кіл спілкування, або адміністраторами програми. Для всіх цих користувачів було створено відповідні діаграми варіантів використання (ДВВ).

Сценарії основного потоку, які виділено для звичайного користувача:

- реєстрація проживання користувача;
- відображення історії реєстрацій проживання;
- відображення активних реєстрацій проживання користувача;
- відображення всіх кіл спілкування користувача;
- відображення кіл спілкування користувача, де його участь підтверджено;
- відображення кіл спілкування користувача, де його участь не підтверджено;
- відображення кіл спілкування користувача, де його участь скасовано;
- відображення кіл спілкування користувача, що не є базовими;
- відображення учасників кола спілкування;
- авторизація;
- перевірка реєстрації проживання користувача;
- відправлення запиту на контакт до сусіда;
- підтвердження запиту на контакт від сусіда;
- відображення всіх адресованих користувачу запитів на контакт;
- відображення всіх створених користувачем запитів на контакт;
- відображення всіх встановлених контактів користувача;
- відображення всіх адресованих користувачу запитів на контакт, які не розглянуто;

– відображення всіх створених користувачем запитів на контакт, які не розглянуто;

– створення персонального запиту до сусіда;
 – створення відповіді на персональний запит до сусіда;
 – відображення всіх персональних запитів, адресованих користувачу;
 – відображення всіх персональних запитів, відправлених користувачем;

– відображення всіх персональних запитів за заданою іншою стороною;
 – подання запиту на участь в колі спілкування;
 – відображення доступних користувачу додаткових кіл спілкування;
 – відображення запитів у колі спілкування;
 – створення відповіді на запит у колі спілкування;
 – створення запиту в колі спілкування;
 – відображення всіх запитів користувача;
 – відображення всіх нових відповідей на запити користувача;
 – оформлення скарги на дії адміністратора кола спілкування;
 – редагування власних даних;
 – відображення скарги на дії адміністратора кола спілкування;
 – відображення власних скарг користувача.

Альтернативні сценарії:

– архівування реєстрації проживання для сценарію реєстрації проживання користувача, коли користувач більше не проживає за зареєстрованою адресою;

– реєстрація в програмі для сценарію авторизації, коли користувач не має акаунту і не може використовувати програму, але потребує цього;

– скасування запиту на контакт від сусіда для сценарію підтвердження запиту на контакт від сусіда, коли користувач не бажає отримувати персональні запити від цього користувача;

– скасування участі користувача в колі спілкування для сценарію подання запиту на участь в колі спілкування, коли запит поданий, але вже

неактуальний (рис. 2.1).

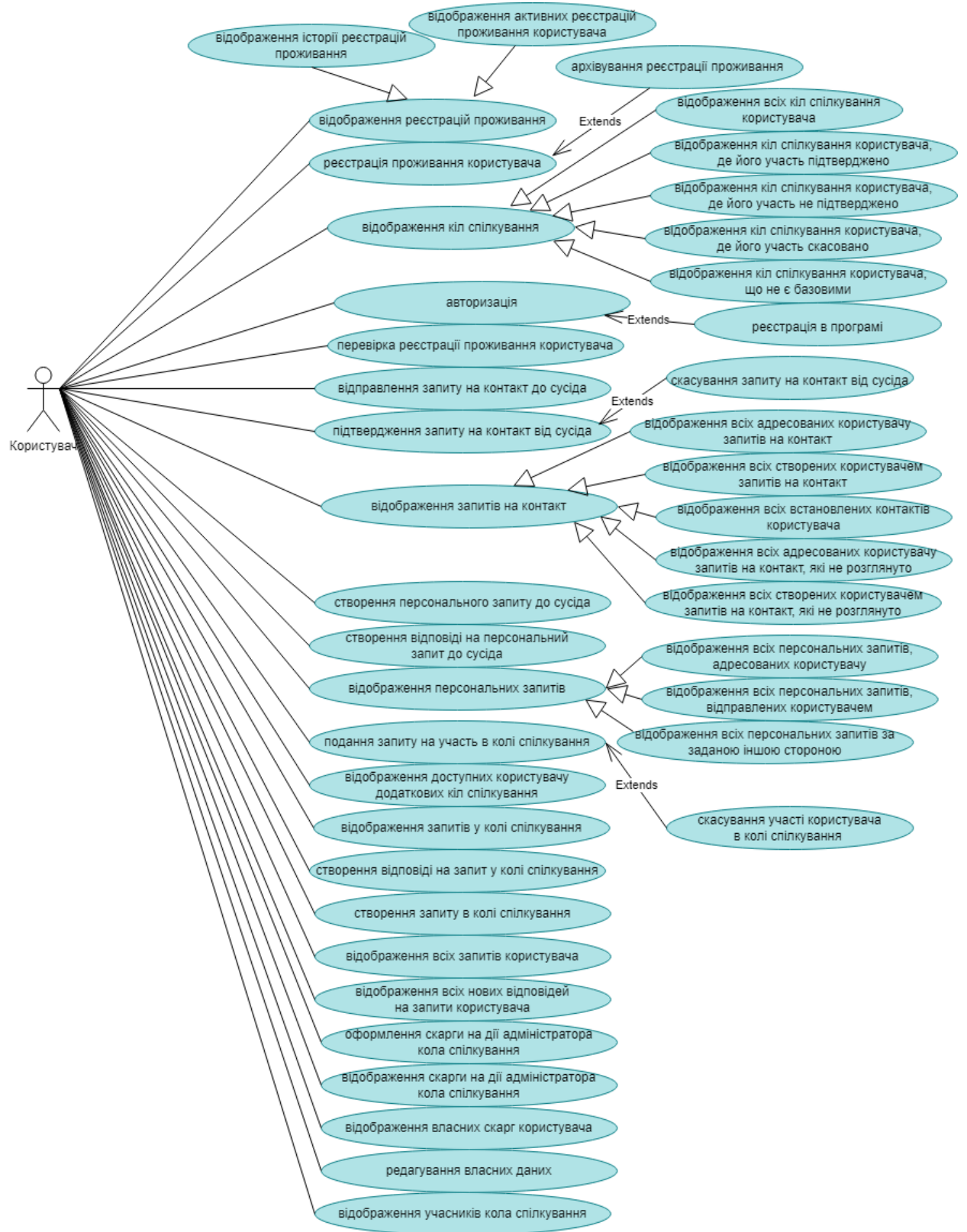


Рисунок 2.1 – ДВВ користувача програми для обміну запитами з сусідами

Сценарії основного потоку, які виділено для адміністратора кола додатково до сценаріїв звичайного користувача:

- створення кола спілкування серед сусідів;
- підтвердження участі користувача в колі спілкування;
- відображення кіл спілкування користувача, де він є адміністратором.

Альтернативні сценарії:

- архівування кола спілкування серед сусідів для сценарію створення кола спілкування серед сусідів, коли потреби в колі спілкування більше немає;
- редагування кола спілкування серед сусідів для сценарію створення кола спілкування серед сусідів, коли потрібно оновити дані кола спілкування (рис. 2.2).

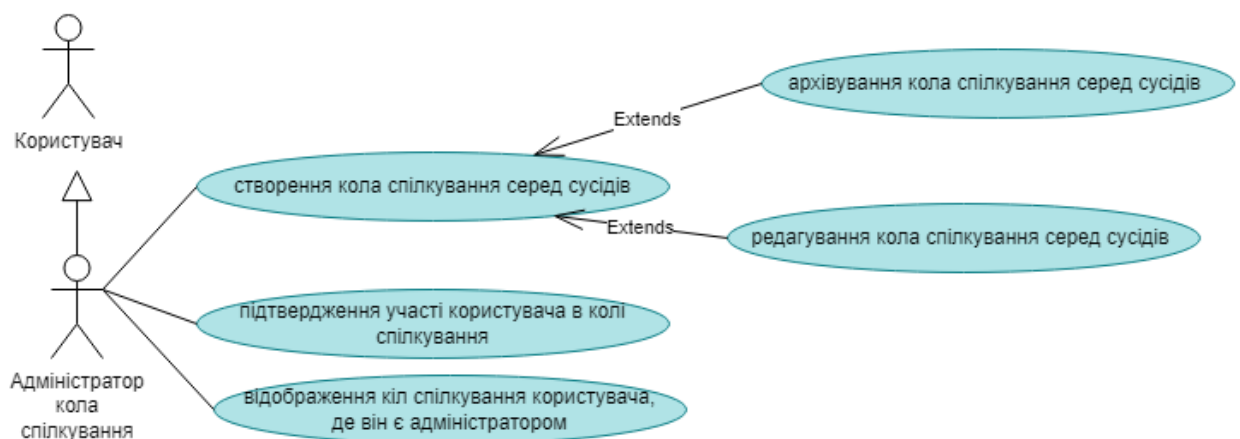


Рисунок 2.2 – ДВВ адміністратора кола спілкування в програмі для обміну запитами з сусідами

Сценарії основного потоку, які виділено для адміністратора програми:

- відображення скарг за адміністратором кола спілкування;
- відображення скарги на дії адміністратора кола спілкування;
- відображення всіх отриманих скарг на дії адміністраторів кіл спілкування;
- пошук серед всіх отриманих скарг на дії адміністраторів кіл спілкування;
- авторизація;

- керування скаргою;
 - відображення повного вмісту запитів у колі для спілкування
- (рис. 2.3).

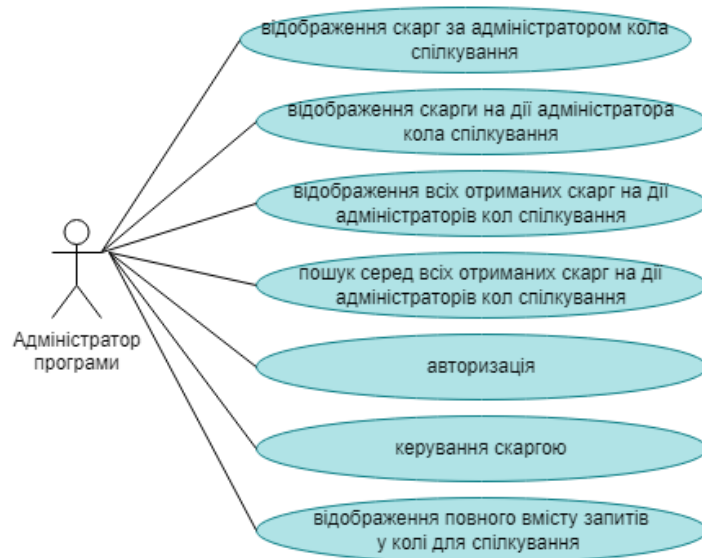


Рисунок 2.3 – ДБВ адміністратора програми для обміну запитами з сусідами

2.3 Визначення структури бази даних

База даних (БД) дозволяє структурувати дані, уникнути їх дублювання, надмірності, забезпечити цілісність, тому саме таким чином визначено порядок роботи з даними в програмі. БД вирішено реалізовувати за допомогою MySQL, безкоштовного, вільнодоступного інструментаря, який відзначається продуктивністю.

Структуризація поняття сусідства звичайно що передбачає під собою визначення структури будівель у місті. Якщо такі будівлі не визначені, то тоді будь-хто може внести будь-яку адресу, в тому числі неіснуючої будівлі, та почати використовувати програму таким чином. А це вже спотворює всю роботу, адже за рахунок неіснуючої будівлі та оформлення проживання в ній можна стати сусідом будь-кому.

Будівлі міст виділені в таблицю БД Building з полями-характеристиками:

- id – номер будівлі міста, де може реєструватися проживання користувача;
- location – місто, де така будівля розташована (програмно задається перелік міст, в якому програма працює);
- street – вулиця, на якій знаходиться будівля в зазначеному місті;
- streetnumber – номер будинку на зазначеній вулиці;
- flats – кількість квартир, яка є в цьому будинку (це фактично максимальний можливий номер квартири, в якому може реєструватися користувач програми).

Користувачі програми, які виступають в подальшому сусідами для інших користувачів, виділені в таблицю БД Neighbor з полями-характеристиками:

- id – номер запису про користувача-сусіда в програмі;
- user – акаунт в Django, за яким реалізується авторизація користувача-сусіда в програмі;
- docname – вірне ім'я за документами людини;
- docsurname – вірне прізвище за документами людини;
- phonenumber – номер телефону, за яким можна з'єднатися з користувачем-сусідом;
- photodata – зображення людини.

Таблиця User є базовою для Django, тобто вона створюється зі створенням самого додатку, тому її окремо не було описано, адже з одного боку вона включає стандартну інформацію (логін, пароль тощо), а з іншого не є частиною тих рішень, які приймалися безпосередньо в даній роботі, а є рішеннями розробників Django. Тому інформацію про людину, яка є користувачем програми і є або претендує на те, щоб стати користувачем-сусідом, виділено в окрему таблицю Neighbor, яка має зв'язок один-до-одного з цією таблицею User. Кожен користувач Django може тільки один запис про себе зробити як про фактичного або потенційного сусіда, але і сусід може мати тільки один акаунт Django, не більше.

Потенційно кожна людина може проживати одночасно в декількох місцях. Можливо певне з них тимчасово, однак якщо людина приїхала, наприклад, в інше місто на декілька місяців, то достатньо обґрунтованим є те, щоб дозволити їй контактувати з сусідами. Тому окремо виділено таблицю БД `Homeregistration` реєстрацій проживання з полями-характеристиками:

- `id` – номер реєстрації проживання людини;
- `neighbor` – користувач-сусід, який зареєстрував це своє проживання в програмі;
- `building` – будівля, де він проживає;
- `flat` – номер квартири, де він проживає (якщо будинок не є багатоквартирним, то значення може бути порожнім);
- `checklong` – довгота, що відповідає місцю перевірки, визначеному під час реєстрації проживання;
- `checklat` – широта, що відповідає місцю перевірки, визначеному під час реєстрації проживання;
- `registertime` – час реєстрації проживання користувача-сусіда;
- `adminchecked` – перевірено адміністратором базового кола спілкування чи ні;
- `archived` – актуальна реєстрація проживання або вже є архівною і не використовується.

Таблиця `Building` відноситься до таблиці `Homeregistration` як один-до-багатьох. Такий тип пояснюється тим, що в одній будівлі може бути зареєстровано декілька проживань (від 0), проте сама по собі реєстрація проживання не може бути множинною, тобто не можна однією реєстрацією оформити проживання в різних будівлях, адже для цього потрібно створювати окремі реєстрації. Це зокрема пов'язано і з тим, що коли виконуються будь-які дії користувачем-сусідом у програмі, то якщо він проживає в декількох місцях, він має обрати ту реєстрацію, яка відповідає тій дії, що він бажає виконати.

Таблиця `Neighbor` відноситься до таблиці `Homeregistration` як один-до-багатьох. Такий тип пояснюється тим, що одна людина може, як і було

зазначено вище, реєструватися в різних будівлях, проживаючи одночасно в декількох місцях (наприклад, в одній постійно, а в іншій тимчасово). Такий підхід зокрема дозволяє взаємодіяти з сусідами і власнику будівлі або квартири, і винаймачу, що надає можливість не втрачати зв'язок і додатково сприяє вирішенню нагальних проблем. Проте кожна реєстрація стосується тільки однієї людини.

Однак реєстрація проживання може бути проблемою для витоку даних про інших людей, тому до допуску до даних сусідів (навіть реальних імені та прізвища) необхідно, щоб користувач був перевірений. Оскільки адміністратор може не одразу перевірити дані користувача, адже це може потребувати і особистого контакту, а до того ж користувач може видати себе певними діями (наприклад, намагаючись вчинити шахрайство), то корисним може бути надання інформації іншими користувачами, які вже є учасниками базового кола спілкування. Такі дані виділено в таблицю БД Registrationcheck з полями-характеристиками:

- id – номер перевірки реєстрації проживання людини користувачем-сусідом базового кола спілкування;
- homeregistration – реєстрація проживання, за якою було виконано перевірку;
- checkerregistration – реєстрація проживання людини, яка виконала перевірку;
- checktime – час перевірки реєстрації проживання;
- checktype – тип результату перевірки реєстрації проживання: підтвердження або спростування.

Реєстрація проживання може мати перевірки від різних користувачів, тому таблиця Homeregistration відноситься до таблиці Registrationcheck як один-до-багатьох. Відповідно користувач, який зареєстрував своє проживання, може перевіряти реєстрацію проживання інших користувачів, при чому виконувати різну кількість таких перевірок. До того ж кожна перевірка може мати тільки одного перевіряльника. Тому другий раз, вже

через поле checkerregistration (а не через поле homeregistration як у першому випадку) таблиця Homeregistration відноситься до таблиці Registrationcheck як один-до-багатьох.

Типи запитів, які можна подати до користувачів-сусідів, виділені в таблицю БД Requesttopic з полями-характеристиками:

- id – номер типу запиту до сусіда;
- topictype – назва типу запиту до сусіда.

Запити можуть адресуватися колам користувачів-сусідів (детальніше описано далі), а можуть бути персонально направленими. Персонально направлені запити можна виконувати тоді, коли є підтвердження на контакт з двох сторін між користувачами. Якщо такого підтвердження немає або якщо інший користувач негативно відповів на спробу встановити контакт, то персональний запит вже відправити не вдасться. Тому важливим елементом є збереження даних про те, як було організовано контакт між сторонами-сусідами. Ці контакти винесені в таблицю БД Personcontact з полями-характеристиками:

- id – номер персонального контакту між користувачами-сусідами;
- srchomeregistration – реєстрація проживання користувача-сусіда, який створив запит на контакт;
- dsthomeregistration – реєстрація проживання користувача-сусіда, якому адресовано запит на контакт;
- resulttype – тип опрацювання запиту на контакт: підтверджено, скасовано або необроблено;
- createtime – час створення запиту на контакт.

Персонально направлені запити між користувачами-сусідами виділені в таблицю Personrequest з полями-характеристиками:

- id – номер персонально направленою запиту між користувачами-сусідами;

– originalrequest – персонально направлений запит, на який є відповіддю цей запит (якщо не задано, то це оригінальний запит, а не відповідь);

– srchomeregistration – реєстрація проживання користувача-сусіда, який створив персонально направлений запит;

– dsthomeregistration – реєстрація проживання користувача-сусіда, якому направлено персональний запит;

– requesttime – час створення персонально адресованого запиту;

– requesttopic – тип персонального запиту до сусіда;

– requesttext – текст персонально адресованого запиту.

До персонально направленого запиту може бути додано зображення, які для цього виділені в таблиці БД Personrequestphoto з полями-характеристиками:

– id – номер зображення, яке було завантажено до персонально направленого запиту;

– personrequest – персонально направлений запит, в який входить зображення;

– photodata – вміст самого зображення, доданого до персонально направленого запиту.

Таблиця Personrequest відноситься до таблиці Personrequestphoto як один-до-багатьох. Такий тип пояснюється тим, що зображення може стосуватися тільки одного персонально направленого запиту і не може поширюватися в різних запитах. Проте кожен персонально направлений запит може мати декілька фотографій.

Кола спілкування виділяються для того, щоб користувачі могли обмінюватися запитами в певній групі. Таким базовим колом є люди, які проживають в тому самому будинку. Таке коло створюється автоматично, коли реєструється хоча би один користувач за адресою відповідної будівлі. Відповідно він автоматично стає адміністратором. Проте якщо з цим виявилось якесь зловживання, то інші користувачі, які реєструються за цим

місцем і наприклад, не можуть через адміністратора отримати доступ, хоча фактично там проживають, можуть звернутися зі скаргою до адміністратора програми, який вже має винести власне рішення щодо надання або позбавлення прав такого адміністратора кола спілкування. Інші кола спілкування створюються за бажанням, але серед сусідів. Тобто це може бути спілкування за якоюсь тематикою, але окрім того це може бути і спілкування з розширеною кількістю людей, адже умовою можливості додавання людини до кола є те, що вона зареєструвала проживання в радіусі 100 метрів (тобто фактично відстань між сусідами може складати 200 метрів). Таким чином до додаткових кіл можуть входити люди, які не проживають в одному й тому самому будинку. Але вони мають входити в базове коло спілкування, яке створюється автоматично.

Для кожного кола має бути визначено, чи всі запити в ньому мають бути видні всім учасникам цього кола спілкування. Якщо ні, то запити та відповіді на них видні тільки тим користувачам, які отримали підтвердження від адміністратора (якщо така необхідність визначена для кола). Якщо ж всі мають доступ, то тоді кожен учасник кола спілкування може визначити за своїм запитом, чи він дозволяє його бачити тим учасникам кола, які не отримали підтвердження самої реєстрації проживання користувача.

Кола спілкування між користувачами виділені в таблицю Neighborcircle з полями-характеристиками:

- id – номер кола спілкування між користувачами;
- circlename – назва кола спілкування між користувачами;
- circletype – тип кола спілкування між користувачами: базовий, який автоматично формується з людей, які зареєстрували і підтвердили проживання в певній будівлі, чи додатково створений;
- adminhomeregistration – реєстрація проживання користувача-сусіда, який є адміністратором кола спілкування;
- createtime – час створення кола спілкування;
- archived – дійсне коло спілкування або заархівоване;

– widetype – дозволяється переглядати запити в колі спілкування до того, як користувач був підтверджений у колі адміністратором, чи ні.

Учасник кіл спілкування виділені в таблицю Neighborcirclemember з полями-характеристиками:

- id – номер учасника кола спілкування;
- neighborcircle – коло спілкування, до якого входить як учасник користувач;
- homeregistration – реєстрація проживання, за якою має намір взяти участь у колі спілкування учасник або за якою бере участь;
- createtime – час створення запиту на участь в даному колі спілкування;
- archived – участь є актуальною або вже заархівованою (вилучено участь);
- blocked – учасника заблоковано в цьому колі для спілкування або ні;
- approved – участь в цьому колі спілкування підтверджено адміністратором або ні.

Таблиця Neighborcircle відноситься до таблиці Neighborcirclemember як один-до-багатьох. Такий тип пояснюється тим, що в коло спілкування можуть входити різні учасники (тут необхідно уточнити, що початково в колі знаходиться хоча б один учасник, але якщо воно є заархівованим і для історії не вилучається, то всі учасники можуть його покинути), але ж і учасник є учасником одного конкретного кола спілкування. Якщо людина входить в різні кола спілкування, то це можливо, але кожна така участь оформлюється окремим записом у таблиці Neighborcirclemember.

Запити, які направляються не персонально, а в коло спілкування, виділені в таблицю БД Neighborcirclerequest з полями-характеристиками:

- id – номер запита, який направляється в коло спілкування;
- neighborcircle – коло спілкування, у яке відправлено учасником запит;
- homeregistration – реєстрація проживання, за якою має намір взяти участь у колі спілкування учасник або за якою бере участь;

– `originalrequest` – якщо вказано порожнє значення, то це означає, що даний запис представляє сам початковий запит учасника кола спілкування, але в іншому випадку це може бути сам запит в колі спілкування, на який є відповіддю даний запис: якщо надається відповідь, то вказується запит, а якщо це сам запит, то значення порожнє;

– `requesttime` – час відправлення цього запиту в коло спілкування;

– `requesttopic` – тип персонального запиту в колі спілкування;

– `requesttext` – текст запиту, відправленого в коло спілкування, що визначає суть проблеми;

– `widetype` – дозволяється переглядати цей запит в колі спілкування користувачу до того, як він був підтверджений у колі адміністратором, чи ні.

Таблиця `Neighborcircle` відноситься до таблиці `Neighborcirclerequest` як один-до-багатьох. Такий тип пояснюється тим, що кожне коло спілкування може включати багато запитів, відправлених в це коло, але з іншого боку кожен запит направляється в конкретне коло спілкування.

Таблиця `Homeregistration` відноситься до таблиці `Neighborcirclerequest` як один-до-багатьох. Такий тип пояснюється тим, що кожна реєстрація проживання може використовуватися багато разів для відправлення за нею запитів у різні (або однакові) кола спілкування, але з іншого боку кожен запит визначається одним авторством, яке і встановлюється через зазначення реєстрації проживання.

Таблиця `Neighborcirclerequest` відноситься до себе ж (до таблиці `Neighborcirclerequest`) як один-до-багатьох. Такий тип пояснюється тим, що кожен запит в коло спілкування може бути або оригінальним і тоді він не має зв'язку з іншим запитом, або бути відповіддю на існуючий запит в колі спілкування і тоді цей запит адресується до іншого запиту, але тільки одного (не може формуватися відповідь одразу на декілька запитів). Проте і кожен запит в коло спілкування може містити різну кількість відповідей у вигляді інших запитів (від 0).

До запиту, відправленого в коло спілкування, може бути додано зображення, які для цього виділені в таблиці БД Neighborcirclerequestphoto з полями-характеристиками:

- id – номер зображення, яке було завантажено до запиту в коло спілкування;
- neighborcirclerequest – запит у коло спілкування, в який входить зображення;
- photodata – вміст самого зображення, доданого до запиту, направлено в коло спілкування.

Таблиця Neighborcirclerequest відноситься до таблиці Neighborcirclerequestphoto як один-до-багатьох. Такий тип пояснюється тим, що зображення може стосуватися тільки одного запиту, направлено в коло спілкування, і не може поширюватися в різних запитах. Проте кожен запит, відправлений в коло спілкування, може мати декілька фотографій.

Для кожного запиту в коло спілкування, в якому можуть брати участь учасники з непідтвердженим (неперевіреним) адміністратором базового кола спілкування реєстраціями проживання, автор запиту може визначити, чи потрібно їм цей запит демонструвати, чи ні. Якщо ні, то ніякої інформації з запиту, а також відповідей на запит з кола спілкування, такі користувачі не зможуть отримати. З іншого боку можна також визначити і конкретних учасників для будь-якого запиту, яким цей запит в колі спілкування не буде видимим.

Видимість запиту в коло спілкування виділяється таблицею Neighborcirclerequestnotseen з полями-характеристиками:

- id – номер визначення видимості запиту в коло спілкування;
- neighborcirclerequest – запит у коло спілкування, для якого визначається видимість;
- homeregistration – реєстрація проживання користувача з кола спілкування, для якого заборонено переглядати цей запит (рис. 2.4).

Таблиця Neighborcirclerequest відноситься до таблиці Neighborcirclerequestnotseen як один-до-багатьох. Такий тип пояснюється тим, що для кожного запиту може бути визначено декілька заборон для інших учасників кола спілкування для перегляду запиту. Але кожна така заборона стосується одного запиту.

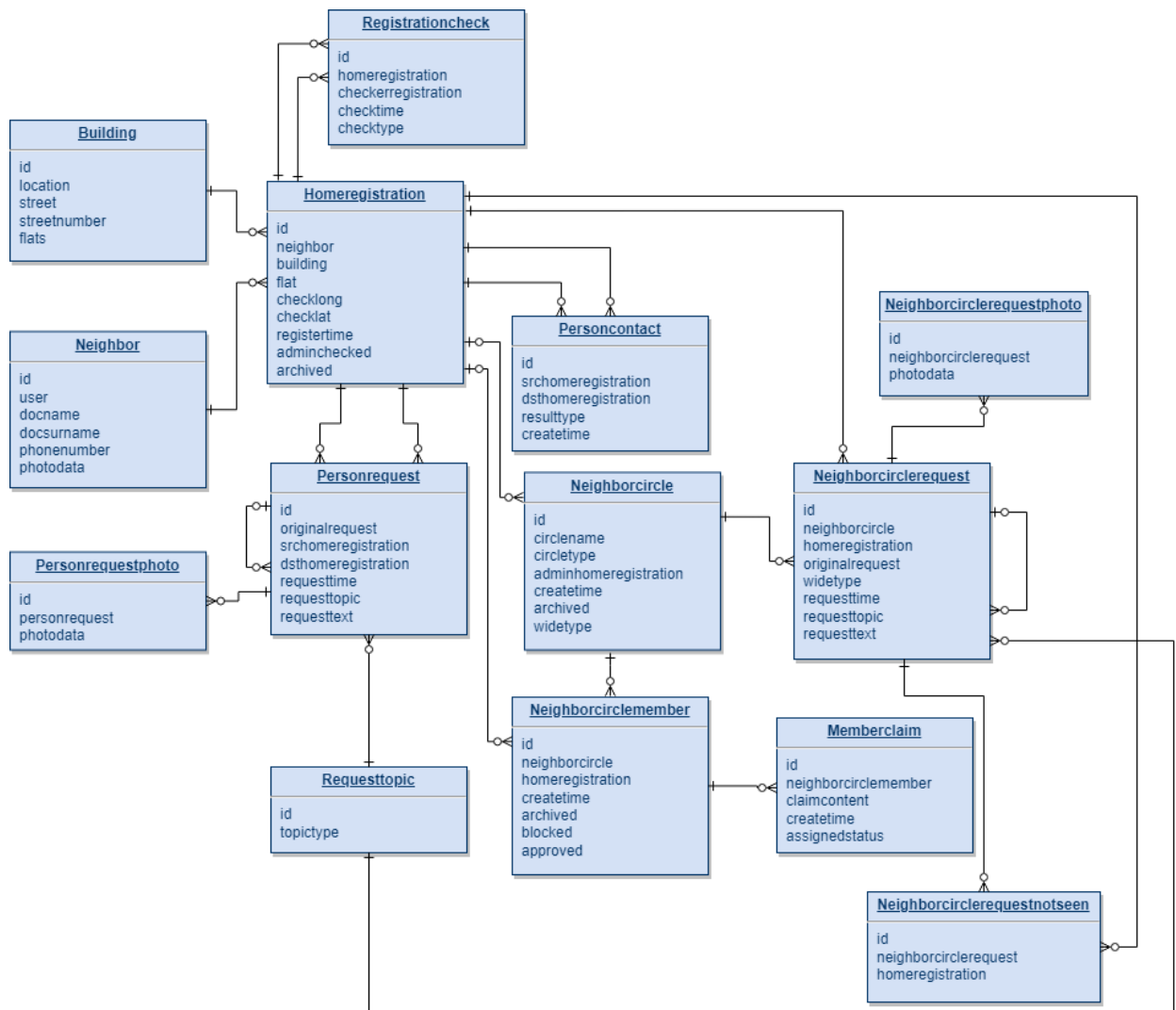


Рисунок 2.4 – Схема БД програми для обміну запитами з сусідами

Таблиця Homeregistration відноситься до таблиці Neighborcirclerequestnotseen як один-до-багатьох. Такий тип пояснюється тим, що за однією реєстрацією проживання може бути визначено багато заборон іншими учасниками кіл спілкування для перегляду їхніх запитів. Проте кожна заборона стосується тільки однієї реєстрації проживання.

Скарги до адміністратора програми на порушення правил адміністраторами кіл спілкування виділяються таблицею Memberclaim з полями-характеристиками:

- id – номер скарги до адміністратора програми на порушення правил адміністратором кола;
- neighborcirclemember – учасник кола спілкування, який подав цю скаргу;
- claimcontent – суть скарги, описана текстом;
- createtime – час створення скарги;
- assignedstatus – стан, який було присвоєно скарзі.

2.4 Висновки за розділом 2

БД вирішено реалізовувати за допомогою MySQL, безкоштовного, вільнодоступного інструментаря, який відзначається продуктивністю. Структуру БД визначено з необхідністю розподілу прав на доступ до запитів, включаючи як персональні, так і запити в колах спілкування. ДБВ створені для звичайного користувача, адміністратора кола спілкування та для адміністратора програми. Вирішено реалізацію провадити мовою Python засобами реалізації вебдодатків Django.

3 РОЗРОБКА ПРОГРАМИ

3.1 Визначення порядку роботи з програмою

При визначенні порядку роботи з програмою було сформовано варіанти меню, які доступні користувачу в залежності від його ролі. Це детальніше описано в розділі 4. Але наріжною проблемою щодо забезпечення правильного доступу є визначення тих повідомлень, які конкретному користувачу мають відображатися в колі спілкування. Для цього було сформовано окрему схему (рис. 3.1).

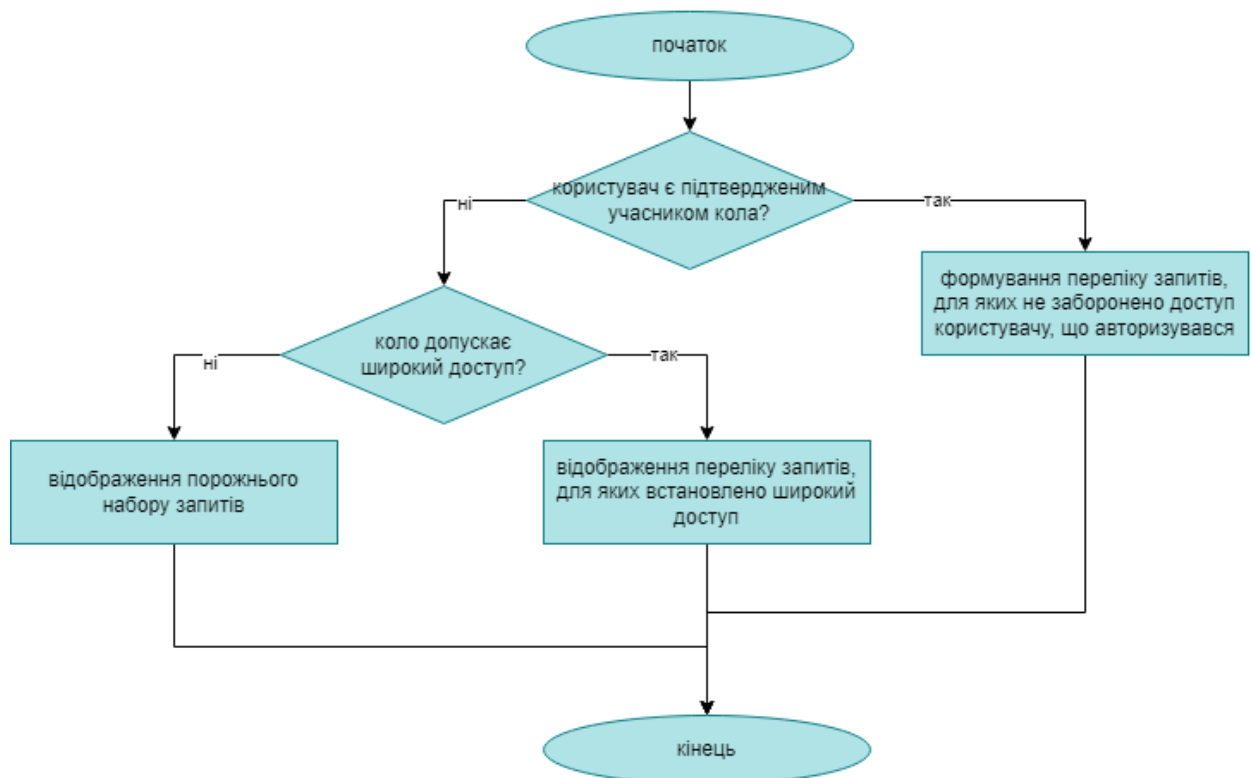


Рисунок 3.1 – Схема порядку формування переліку запитів для відображення користувачу в колі спілкування

Тоді спочатку відбувається перевірка того, чи є користувач після авторизації підтвердженим учасником кола. Якщо так, то для нього формується перелік запитів, які в цьому колі були створені вже, але виключаються ті, які автором запиту були персонально для нього заборонені. Якщо користувач не є підтвердженим учасником кола, то відбувається

перевірка, чи допускає взагалі це коло широкий доступ. Якщо так, то відображаються ті запити з цього кола, які мають широкий доступ. Якщо коло взагалі не допускає широкий варіант доступу, то відображається порожній набір запитів.

Поза межі даного порядку винесено перевірку того, чи не є користувач заблокованим і чи не є його участь заархівованою.

3.2 Особливості програмної реалізації

Враховуючи створення структури даних на основі коду, важливим під час реалізації є визначення моделей додатку, що детально описано нижче. Реалізація ж функцій програми пов'язана з реалізацією пари з шаблонів сторінок та представлень, що описано далі.

Будівлі міст реалізовано моделлю `Building` з полями:

- `location` реалізує місто, де така будівля розташована, через тип `CharField`;
- `street` реалізує вулицю, на якій знаходиться будівля в зазначеному місті, через тип `CharField`;
- `streetnumber` реалізує номер будинку на зазначеній вулиці через тип `IntegerField`;
- `flats` реалізує кількість квартир, яка є в цьому будинку, через тип `IntegerField`.

Користувачів програми, які виступають в подальшому сусідами для інших користувачів, реалізовано моделлю `Neighbor` наслідуванням класу `User` з додатковими полями:

- `docname` реалізує вірне ім'я за документами людини через тип `CharField`;
- `docsurname` реалізує вірне прізвище за документами людини через тип `CharField`;

- `phonenumber` реалізує номер телефону, за яким можна з'єднатися з користувачем-сусідом, через тип `CharField`;

- `photodata` реалізує вміст зображення людини через тип `ImageField`.

Реєстрації проживання реалізовано моделлю `Homeregistration` з полями:

- `neighbor` реалізує користувача-сусіда, який зареєстрував це своє проживання в програмі через тип `ForeignKey` і встановлення зв'язку з моделлю `Neighbor`;

- `building` реалізує будівлю, де користувач проживає, через тип `ForeignKey` і встановлення зв'язку з моделлю `Building`;

- `flat` реалізує номер квартири, де користувач проживає, через тип `IntegerField`;

- `checklong` реалізує довготу, що відповідає місцю перевірки, визначеному під час реєстрації проживання, через тип `DecimalField`;

- `checklat` реалізує широту, що відповідає місцю перевірки, визначеному під час реєстрації проживання, через тип `DecimalField`;

- `registertime` реалізує час реєстрації проживання користувача-сусіда через тип `DateTimeField`;

- `adminchecked` реалізує позначку перевірки адміністратором базового кола спілкування цієї реєстрації проживання (підтверджено правдивість, підтверджено неправдивість, не розглянуто) через тип `PositiveSmallIntegerField`;

- `archived` реалізує позначку актуальності реєстрації проживання через тип `BooleanField`.

Перевірки реєстрацій проживання реалізовано моделлю `Registrationcheck` з полями:

- `homeregistration` реалізує реєстрацію проживання, за якою було виконано перевірку, через тип `ForeignKey` і встановлення зв'язку з моделлю `Homeregistration`;

- checkerregistration реалізує реєстрацію проживання людини, яка виконала перевірку, через тип ForeignKey і встановлення зв'язку з моделлю Homeregistration;

- checktime реалізує час перевірки реєстрації проживання через тип DateTimeField;

- checktype реалізує тип результату перевірки реєстрації проживання (підтвердження або спростування) через тип BooleanField.

Типи запитів, які можна подати до користувачів-сусідів, реалізовано моделлю Requesttopic з єдиним додатковим полем topictype, що реалізує назву типу запиту до сусіда.

Контакт між сторонами-сусідами реалізовано моделлю Personcontact з полями:

- srchomeregistration реалізує реєстрацію проживання користувача-сусіда, який створив запит на контакт, через тип ForeignKey і встановлення зв'язку з моделлю Homeregistration;

- dsthomeregistration реалізує реєстрацію проживання користувача-сусіда, якому адресовано запит на контакт, через тип ForeignKey і встановлення зв'язку з моделлю Homeregistration;

- resulttype реалізує тип опрацювання запиту на контакт (підтверджено, скасовано або необроблено) через тип PositiveSmallIntegerField;

- createtime реалізує час створення запиту на контакт через тип DateTimeField.

Персонально направлені запити між користувачами-сусідами реалізовано моделлю Personrequest з полями:

- originalrequest реалізує персонально направлений запит, на який є відповіддю цей запит, через тип ForeignKey і встановлення зв'язку з моделлю Personrequest;

- srchomeregistration реалізує реєстрацію проживання користувача-сусіда, який створив персонально направлений запит, через тип ForeignKey і встановлення зв'язку з моделлю Homeregistration;

- `dsthomeregistration` реалізує реєстрацію проживання користувача-сусіда, якому направлено персональний запит, через тип `ForeignKey` і встановлення зв'язку з моделлю `Homeregistration`;
- `requesttime` реалізує час створення персонально адресованого запиту через тип `DateTimeField`;
- `requesttopic` реалізує тип персонального запиту до сусіда через тип `ForeignKey` і встановлення зв'язку з моделлю `Requesttopic`;
- `requesttext` реалізує текст персонально адресованого запиту через тип `CharField`.

До персонально направленного запиту може бути додано зображення, які реалізовано моделлю `Personrequestphoto` з полями:

- `personrequest` реалізує персонально направлений запит, в який входить зображення, через тип `ForeignKey` і встановлення зв'язку з моделлю `Personrequest`;
- `photodata` реалізує вміст самого зображення, доданого до персонально направленного запиту, через тип `ImageField`.

Кола спілкування між користувачами реалізовано моделлю `Neighborcircle` з полями:

- `circlename` реалізує назву кола спілкування між користувачами через тип `CharField`;
- `circletype` реалізує тип кола спілкування між користувачами (базовий, який автоматично формується з людей, які зареєстрували і підтвердили проживання в певній будівлі, чи додатково створений) через тип `BooleanField`;
- `adminhomeregistration` реалізує реєстрацію проживання користувача-сусіда, який є адміністратором кола спілкування, через тип `ForeignKey` і встановлення зв'язку з моделлю `Homeregistration`;
- `createtime` реалізує час створення кола спілкування через тип `DateTimeField`;
- `archived` реалізує відмітку того, чи це дійсне коло спілкування, чи заархівоване, через тип `BooleanField`;

– widetype реалізує відмітку про те, чи дозволяється переглядати запити в колі спілкування до того, як користувач був підтверджений у колі адміністратором, чи ні, через тип BooleanField.

Учасники кіл спілкування реалізовано моделлю Neighborcirclemember з полями:

– neighborcircle реалізує коло спілкування, до якого входить як учасник користувач, через тип ForeignKey і встановлення зв'язку з моделлю Neighborcircle;

– homeregistration реалізує реєстрацію проживання, за якою має намір взяти участь у колі спілкування учасник або за якою бере участь, через тип ForeignKey і встановлення зв'язку з моделлю Homeregistration;

– createtime реалізує час створення запиту на участь в даному колі спілкування через тип DateTimeField;

– archived реалізує відмітку про те, чи участь є актуальною, чи вже заархівованою, через тип BooleanField;

– blocked реалізує відмітку про те, що учасника заблоковано в цьому колі для спілкування або ні, через тип BooleanField;

– approved реалізує відмітку про те, чи участь в цьому колі спілкування підтверджено адміністратором, чи ні (так, ні, не розглянуто), через тип PositiveSmallIntegerField.

Запити, які направляються не персонально, а в кола спілкування, реалізовано моделлю Neighborcirclerequest з полями:

– neighborcircle реалізує коло спілкування, у яке відправлено учасником запит, через тип ForeignKey і встановлення зв'язку з моделлю Neighborcircle;

– homeregistration реалізує реєстрацію проживання, за якою має намір взяти участь у колі спілкування учасник або за якою бере участь, через тип ForeignKey і встановлення зв'язку з моделлю Homeregistration;

– originalrequest реалізує зв'язок з попереднім запитом у коло спілкування через тип ForeignKey і встановлення зв'язку з моделлю Neighborcirclerequest;

- requesttime реалізує час відправлення цього запиту в коло спілкування через тип DateTimeField;
- requesttopic реалізує тип персонального запиту в колі спілкування через тип ForeignKey і встановлення зв'язку з моделлю Requesttopic;
- requesttext реалізує текст запиту, відправленого в коло спілкування, що визначає суть проблеми, через тип CharField;
- widetype реалізує дозвіл переглядати цей запит в колі спілкування користувачу до того, як він був підтверджений у колі адміністратором, чи ні, через тип BooleanField;
- notseens реалізує заборону на видимість цього запиту в колі спілкування зазначеним користувачам, які є учасниками цього кола спілкування, через тип ManyToManyField і встановлення зв'язку з моделлю Nomeregistration.

До запиту, відправленого в коло спілкування, може бути додано зображення, які реалізовано моделлю Neighborcirclerequestphoto з полями:

- neighborcirclerequest реалізує запит у коло спілкування, в який входить зображення, через тип ForeignKey і встановлення зв'язку з моделлю Neighborcirclerequest;
- photodata реалізує вміст самого зображення, доданого до запиту, направленого в коло спілкування, через тип ImageField.

Скарги до адміністратора програми на порушення правил адміністраторами кіл спілкування реалізовано моделлю Memberclaim з полями:

- neighborcirclemember реалізує учасника кола спілкування, який подав скаргу, через тип ForeignKey і встановлення зв'язку з моделлю Neighborcirclemember;
- claimcontent реалізує суть скарги, описана текстом, через тип CharField;
- createtime реалізує час створення скарги через тип DateTimeField;

– assignedstatus реалізує стан, який було присвоєно скарзі, через тип PositiveSmallIntegerField.

Представлення вебдодатку реалізовані не класами, а функціями, що дозволяє гнучко формувати у відповідності з логікою виконання функції потрібні дані, щоб наповнити ними шаблон вебсторінки. Шаблони визначають розмітку для кожної сторінки, визначаючи, в яких локаціях які саме дані використовуються. Сама розмітка забезпечена через застосування шаблону [8].

Відображення історії реєстрацій проживання – представлення neighbor_history_homeregistrations, де відбираються реєстрації проживання, виконані користувачем, який авторизувався і має запис у таблиці Neighbor, об'єднуючи такі реєстрації за будівлею проживання. Тобто якщо виявляється, що користувач декілька разів у різний час реєструвався в одному й тому самому місці, то тоді серед історії реєстрацій така будівля буде подаватися тільки один раз, але за нею можна буде переглянути історію змін. Для цього сформований словник, який за ключами, що відповідають номерам будівель, містять реєстрації проживання.

Відображення активних реєстрацій проживання користувача – представлення neighbor_active_homeregistrations, де відбираються реєстрації проживання, виконані користувачем, який авторизувався і має запис у таблиці Neighbor, а самі реєстрації не є архівованими, впорядковуючи їх за часом реєстрації проживання (спершу нові).

Відображення всіх кіл спілкування користувача – представлення all_neighborcircles, де відбираються кола спілкування, учасником яких є користувач, який авторизувався і має запис у таблиці Neighbor, виводячи спершу нові додані.

Реєстрація проживання користувача – представлення homeregistration_make, де визначається геолокація та пропонується зв'язування з найближчими адресами, але не більше ніж на відстані 50 метрів.

Відображення кіл спілкування користувача, де його участь підтверджено, – представлення `approved_neighborcircles`, де відбираються кола спілкування, учасником яких є користувач, який авторизувався і має запис у таблиці `Neighbor`, а участь в яких його підтверджена, виводячи спершу нові додані.

Відображення кіл спілкування користувача, де його участь не підтверджено, – представлення `unapproved_neighborcircles`, де відбираються кола спілкування, учасником яких є користувач, який авторизувався і має запис у таблиці `Neighbor`, а участь в яких його ще не розглянута, виводячи спершу нові додані.

Архівування реєстрації проживання – представлення `homeregistration_archive`. Параметр `homeregistrationid` задає номер реєстрації, яке має обов’язково належати користувачу, який авторизувався.

Відображення кіл спілкування користувача, де його участь скасовано, – представлення `cancelled_neighborcircles`, де відбираються кола спілкування, учасником яких є користувач, який авторизувався і має запис у таблиці `Neighbor`, а участь в яких його скасована адміністраторами, виводячи спершу нові додані.

Перевірка реєстрації проживання користувача – представлення `registrationcheck_make`. Параметр `homeregistrationid` задає номер реєстрації, а вона має знаходитися не більше ніж в 50 метрах від одного з зареєстрованих проживань користувача, який авторизувався.

Відображення кіл спілкування користувача, де він є адміністратором, – представлення `admin_neighborcircles`, де відбираються кола спілкування, адміністратором яких є користувач, який авторизувався і має запис у таблиці `Neighbor`, виводячи спершу нові додані.

Відправлення запиту на контакт до сусіда – представлення `personcontact_make`. Номер реєстрації проживання, за якою реалізується запит, визначається параметром `homeregistrationid`.

Підтвердження запиту на контакт від сусіда – представлення `personcontact_plus`. Номер запиту на контакт від сусіда задається параметром `personcontactid`.

Скасування запиту на контакт від сусіда – представлення `personcontact_minus`. Номер запиту на контакт від сусіда задається параметром `personcontactid`.

Відображення кіл спілкування користувача, що не є базовими, – представлення `nonbase_neighborcircles`, де відбираються кола спілкування, учасником яких є користувач, який авторизувався і має запис у таблиці `Neighbor`, а саме коло не позначено як базове (що формується за будівлею проживання), виводячи спершу нові додані.

Відображення всіх адресованих користувачу запитів на контакт – представлення `to_personcontacts`, де відбираються всі запити, які направлені користувачу, який авторизувався і який є в таблиці `Neighbor`, виводячи спершу нові додані. Відображення всіх адресованих користувачу запитів на контакт, які не розглянуто, – представлення `pending_to_personcontacts`.

Відображення всіх створених користувачем запитів на контакт – представлення `from_personcontacts`, де відбираються всі запити, які створені користувачем, який авторизувався і який є в таблиці `Neighbor`, виводячи спершу нові додані. Відображення всіх створених користувачем запитів на контакт, які не розглянуто, – представлення `pending_from_personcontacts`.

Відображення всіх встановлених контактів користувача – представлення `approved_personcontacts`. Сюди відбираються ті запити на контакти, які були створені або адресовані користувачу, який авторизувався, та стан яких є підтвердженням з іншої сторони, виводячи спершу нові подані.

Відображення всіх персональних запитів, адресованих користувачу, – представлення `to_personrequests`. Це вже не запити на контакт, а самі персональні запити, тому до них відбираються ті, які є запитами, а не відповідями, а вже на кожен запит додаються всі відповіді на нього. Також на кожен персональний запит додаються фотографії, якщо вони входять в такий

запит. Відображення всіх персональних запитів, відправлених користувачем, – аналогічне представлення `from_personrequests`.

Відображення всіх персональних запитів за заданою іншою стороною – представлення `from_to_personrequests`. Інша сторона задається логіном `neighbor_username`, який є параметром функції. Потрібний тип запиту задається параметром `requesttopicid`. Якщо значення -1, то використовуються всі запити за всіма типами, а не тільки заданого типу.

Відображення запитів у колі спілкування – представлення `circle_requests`. Коло спілкування задається номером через параметр `neighborcircleid`. Потрібний тип запиту задається параметром `requesttopicid`. Якщо значення -1, то використовуються всі запити за всіма типами, а не тільки заданого типу. При роботі даного представлення перевіряється порядок, визначений рис. 3.1.

Відображення всіх запитів користувача – представлення `from_requests`. Потрібний тип запиту задається параметром `requesttopicid`. Якщо значення -1, то використовуються всі запити за всіма типами, а не тільки заданого типу. До результатів входять і персональні запити, і запити, які були створені в різних колах спілкування користувачем, який авторизувався. Кожен запит наповнюється також відповідями до нього, зображеннями, доданими до нього та до відповідей.

Відображення всіх нових відповідей на запити користувача – представлення `answer_requests`. Особливістю цього представлення є те, що результати впорядковуються за часом відповідей, тобто останні відповіді розташовуються спочатку, але відповіді групуються за тими запитами, на які вони були оформлені. Тобто за кожною відповіддю відбираються початковий запит та відповіді, дозволяючи відновити ланцюжок. Потрібний тип запиту задається параметром `requesttopicid`. Якщо значення -1, то використовуються всі запити за всіма типами, а не тільки заданого типу.

Відображення скарг за адміністратором кола спілкування – представлення `admin_memberclaims`. Параметром `admin_username` задається логін адміністратора кола спілкування, щодо якого потрібно відібрати скарги.

Відображення скарги на дії адміністратора кола спілкування – представлення `memberclaim_content`. Номер скарги на дії адміністратора кола спілкування визначається параметром `memberclaimid`. Виконується перевірка прав: користувач є автором скарги або користувач є адміністратором програми.

Відображення всіх отриманих скарг на дії адміністраторів кіл спілкування – представлення `all_memberclaims`, впорядковуючи скарги від нових спочатку.

Відображення доступних користувачу додаткових кіл спілкування – представлення `possible_circles`. Доступними вважаються кола спілкування, адміністратори яких зареєстровані в 100 метрах від реєстрації проживання користувача, який авторизувався, та яка має номер `homeregistrationid`. Для перевірки відстані за геолокацією використана бібліотека `haversine`.

Пошук серед всіх отриманих скарг на дії адміністраторів кіл спілкування – представлення `search_memberclaims`. Серед параметрів пошуку враховано як передані з запиту параметри значення потрібного стану скарг, періоду, коли скарги були створені, назви кола спілкування, типу кола спілкування, міста, визначення прямого або зворотного порядку.

3.3 Висновки за розділом 3

Враховуючи створення структури даних на основі коду, важливим під час реалізації є визначення моделей додатку, що детально описано в розділі, а реалізація функцій програми пов'язана з реалізацією пари з шаблонів сторінок та представлень, що також описано.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма для обміну запитами з сусідами надає можливість комунікувати користувачам, які проживають в безпосередній близькості один від одного, обмінюватися інформаційними повідомленнями, відповідати на запитання один одного, звертати увагу один одного, спільно вирішувати проблеми, забезпечуючи для цього як персональну комунікацію з можливістю обмеження її, так і комунікацію в колах спілкування, які враховують принцип сусідства.

4.2 Умови виконання програми

Технічна сторона організації сервера має враховувати обов'язкову відповідність тій кількості запитів на секунду, яка має оброблятися. Чим більшою потрібна кількість є, тим вищими є вимоги, але за умов оброблення 50 запитів ці вимоги мінімально включають процесор не менше Intel Xeon E5440 з частотою 2.83 ГГц, оперативну пам'ять в 12 Гб та жорсткий диск з 40 Гб простору.

4.3 Експлуатація програми

Під час експлуатації програми для обміну запитами з сусідами було виконано перевірку роботи функцій та логіки виконання роботи відповідними користувачами. Для збереження конфіденційності даних після перевірки за допомогою геолокації реєстрації проживання користувача було внесено зміни через адміністративну панель додатку зі зміною адреси для відповідного відображення на скріншотах програми, що подані нижче.

Робота з програмою будь-якого звичайного користувача, навіть якщо він є адміністратором кіл спілкування, є подібною. Для такого користувача

меню поділено на дві частини. У частині, що розташована на панелі зліва, доступні інструменти роботи з колами, контактами та скаргами.

Для роботи з колами спілкування у переліку функцій доступний перехід до всіх кіл, де є учасником користувач, підтверджених, не підтверджених, скасованих, додаткових, в яких він є учасником, та додаткових, які є доступними для нього (знаходяться в радіусі 100 метрів).

Після реєстрації проживання користувач одразу стає учасником того кола, яке є базовим для цієї адреси, проте він не є одразу підтвердженим учасником. До всіх інших кіл він може додаватися за бажанням. Так він може обирати потрібне коле з доступних та подавати заявку.

На рис. 4.1 показано приклад відображення кола спілкування для користувача, який є підтвердженим учасником цього кола.

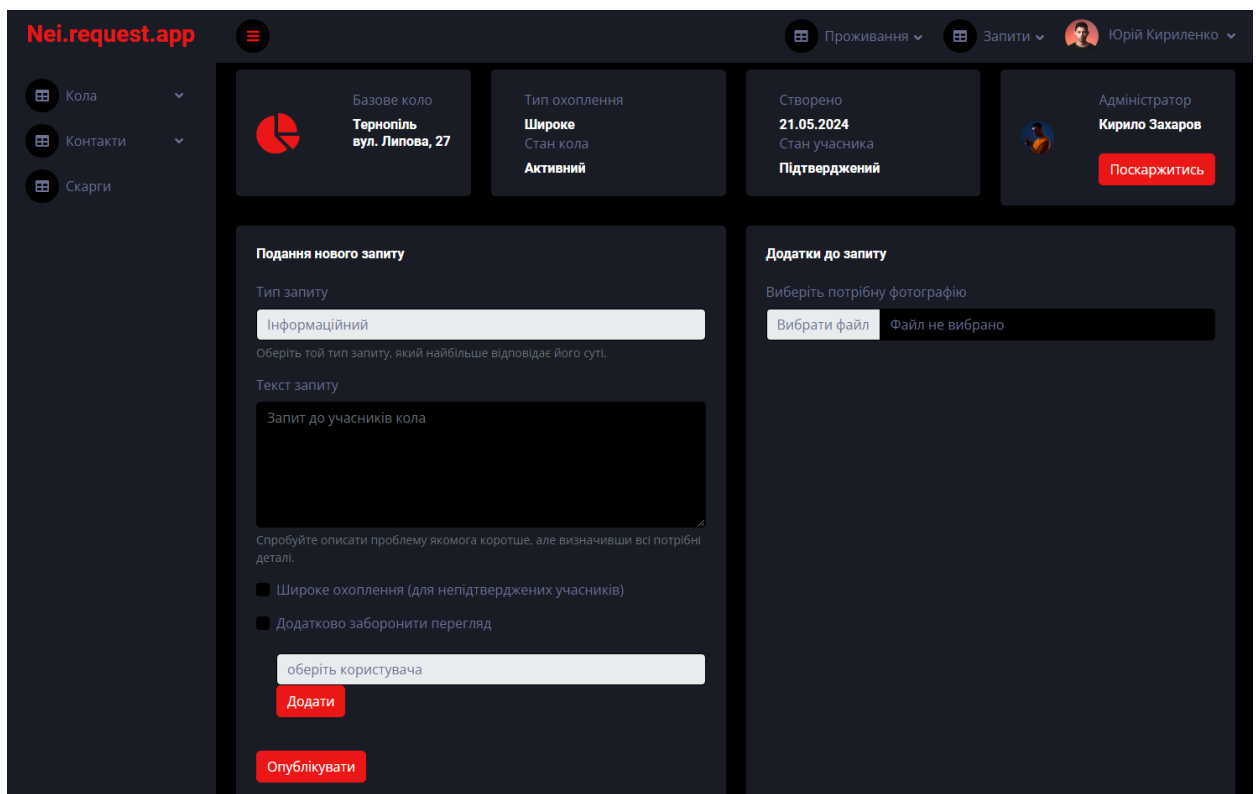


Рисунок 4.1 – Відображення кола для спілкування в частині загальної інформації та подання нового запиту

Відповідна базова інформація відображається згори сторінки. Там же

можна поскаржитись на адміністратора. Оскільки адміністратором базового кола стає користувач, який першим зареєструвався за цією адресою, цей механізм є важливим. Якщо коло передбачає тільки вузьке охоплення (підтверджених користувачів), а користувач не є підтвердженим, то йому відображається тільки ця верхня частина.

Нижче знаходиться поле для створення нового власного запиту. Спочатку потрібно задати категорію запиту: інформаційний, безпековий, розважальний. Далі необхідно ввести текст запиту, визначити, чи буде він доступним широкому охопленню (користувачам, які не є підтвердженими на момент перегляду), а якщо ні, то можна додатково визначити користувачів, яким цей запит не буде видно з тих, хто є підтвердженим. З цих користувачів формується перелік. Кожного можна також вилучити з тих, кого туди додано (з'являється позначка поруч з прізвищем). Також можна додати необхідну кількість фотографій до запиту для ілюстрації.

Далі знаходиться перелік створених запитів у колі (рис. 4.2), а за кожним запитом у правій частині показується перелік відповідей. Кожна відповідь за датою і часом публікації, а потрібно обрати одну з них, натискаючи на відповідній плашці, після чого текст відповіді відображається повністю. Користувач, який є підтвердженим учасником кола, може створити відповідь. Відповіді не відображаються користувачам, для яких не відображається сам початковий запит. Якщо користувач, який є підтвердженим учасником, натисне на фотографію іншого користувача, то він може перейти до перевірки реєстрації цього користувача. З тієї ж сторінки можна перейти до відправлення персонального запиту, якщо між користувачами встановлено контакт.

Якщо користувач не є підтвердженим учасником, але він може бачити певний запит, то йому відображається не повне прізвище та ім'я учасника, а тільки його користувацький логін.

З контактів у лівій панелі можна відкрити функції, які дозволяють відобразити адресовані користувачу запити, створені ним запити та

підтверджені контакти, звідки можна перейти до створення персонального запиту. Верхнє меню дозволяє перейти до проживань та запитів. Проживання включають функції активних проживань, історії проживань та реєстрації нового проживання.

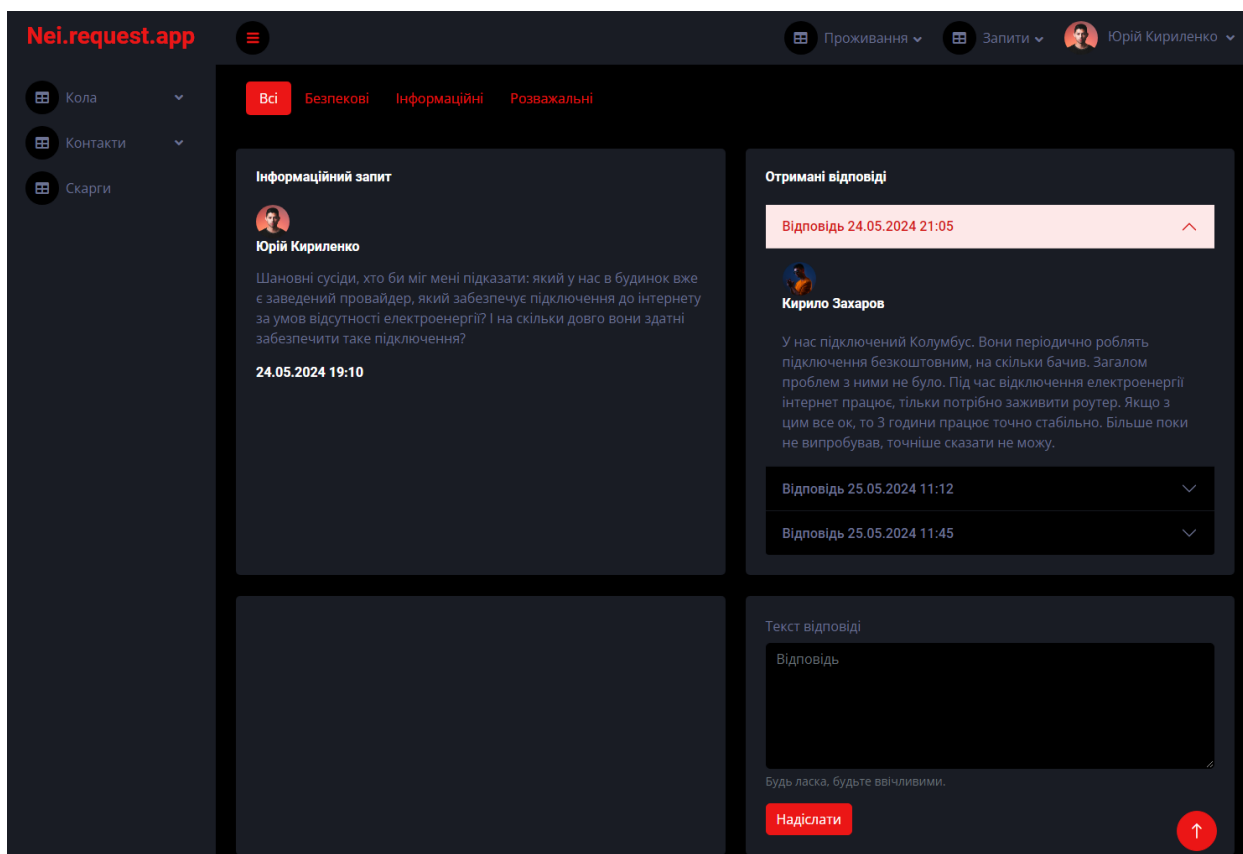


Рисунок 4.2 – Відображення кола для спілкування в частині існуючих запитів та відповідей на них

Активні проживання містять інформацію про самого користувача, про його проживання, дозволяють одразу перейти до реєстрації нового проживання або архівувати те, яке вже було зареєстровано (рис. 4.3).

Користувач, який став адміністратором кола спілкування, має додаткові функції у меню кіл спілкування, що включають відображення його кіл спілкування, де він є адміністратором. З цієї сторінки він може перейти до сторінки всіх учасників кола (рис. 4.4), де може додавати, вилучати учасників, які подали запит, та вилучати підтверджених раніше учасників.

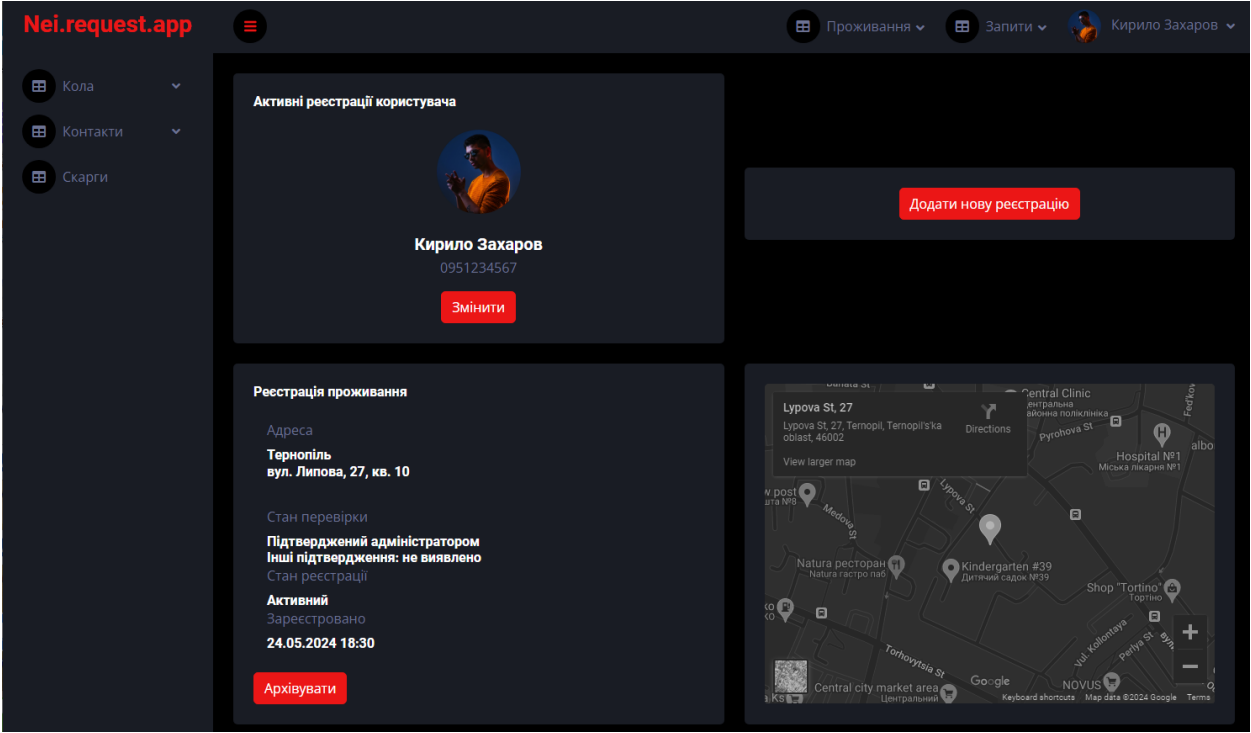


Рисунок 4.3 – Відображення активних реєстрацій проживання користувача

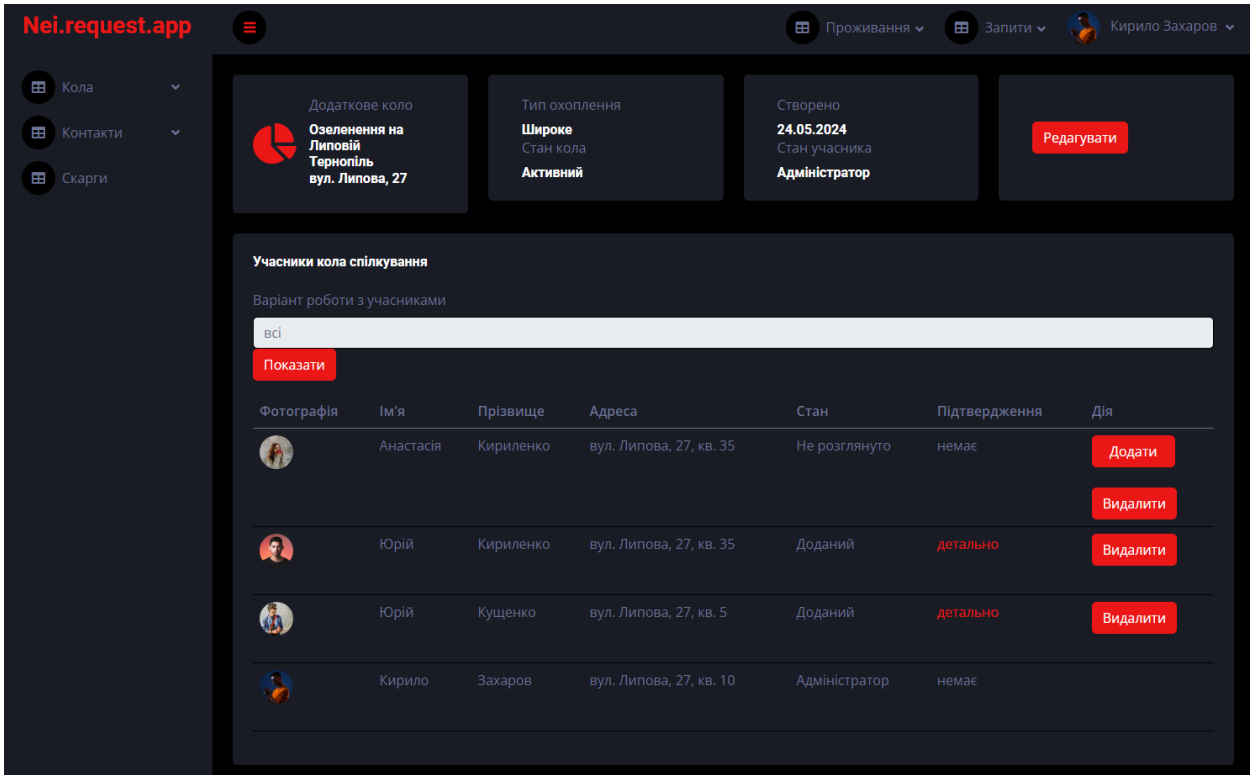
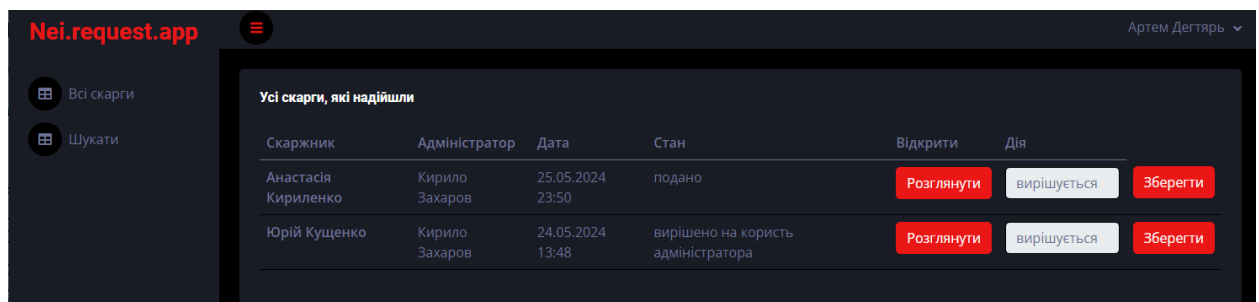


Рисунок 4.4 – Відображення учасників кола спілкування для адміністратора кола

Своє коло спілкування адміністратор може редагувати, звідки може

перейти і до архівації кола спілкування. За кожним учасником адміністратору відображається перехід до переліку перевірок користувача іншими підтвердженими учасниками (посилання «детально»).

Адміністратор програми має інше меню, яке включає функцію відображення всіх скарг (рис. 4.5) та пошуку скарг, де можна деталізувати параметри потрібних скарг. За кожною скаргою адміністратор може кнопкою «Розглянути» відкрити її сторінку з деталізацією вмісту, після чого скарга, якщо вона була у стані поданої, переходить до стану вирішеної. З тієї сторінки адміністратор може перейти до сторінки зі всіма скаргами за цим адміністратором, а також відобразити повний вміст відповідного кола спілкування. На цій та попередніх сторінках можна обрати потрібний стан, в якому знаходиться розгляд скарги, та зберегти його.



Скаржник	Адміністратор	Дата	Стан	Відкрити	Дія
Анастасія Кириленко	Кирило Захаров	25.05.2024 23:50	подано	Розглянути	вирішується Зберегти
Юрій Кущенко	Кирило Захаров	24.05.2024 13:48	вирішено на користь адміністратора	Розглянути	вирішується Зберегти

Рисунок 4.5 – Відображення всіх скарг на адміністраторів кіл спілкування для адміністратора програми

4.4 Тестування програми

Тестування програми для обміну запитами з сусідами забезпечено через перевірку кожної функції, яка була визначена для реалізації і реалізована. Усі функції дозволили в підсумку на фінальному тестуванні отримати результат, який вважається для них коректним (табл. 4.1). Тож тестування є успішним.

Таблиця 4.1 – Результати тестування програми обміну запитами з сусідами

Функціональна можливість	Результат
1	2
Реєстрація проживання користувача	+
Підтвердження правдивості реєстрації проживання користувача адміністратором базового кола спілкування	+
Підтвердження неправдивості реєстрації проживання користувача адміністратором базового кола спілкування	+
Архівування реєстрації проживання користувачем	+
Відображення історії реєстрацій проживання користувача	+
Відображення активних реєстрацій проживання користувача	+
Відображення всіх кіл спілкування користувача	+
Відображення кіл спілкування користувача, де його участь підтверджено	+
Відображення кіл спілкування користувача, де його участь не підтверджено	+
Відображення кіл спілкування користувача, де його участь скасовано	+
Відображення кіл спілкування користувача, де він є адміністратором	+
Відображення кіл спілкування користувача, що не є базовими	+
Реєстрація в програмі користувача	+
Авторизація адміністратора	+
Авторизація користувача	+
Відображення учасників кола спілкування	+
Керування скаргою	+

Продовження таблиці 4.1

1	2
Відправлення запиту на контакт до сусіда	+
Підтвердження запиту на контакт від сусіда	+
Скасування запиту на контакт від сусіда	+
Відображення всіх адресованих користувачу запитів на контакт	+
Відображення всіх створених користувачем запитів на контакт	+
Відображення всіх встановлених контактів користувача	+
Відображення всіх адресованих користувачу запитів на контакт, які не розглянуто	+
Відображення всіх створених користувачем запитів на контакт, які не розглянуто	+
Створення персонального запиту до сусіда	+
Створення відповіді на персональний запит до сусіда	+
Відображення всіх персональних запитів, адресованих користувачу	+
Відображення всіх персональних запитів, відправлених користувачем	+
Відображення всіх персональних запитів за заданою іншою стороною	+
Створення кола спілкування серед сусідів	+
Архівування кола спілкування серед сусідів	+
Підтвердження участі користувача в колі спілкування	+
Скасування участі користувача в колі спілкування	+
Подання запиту на участь в колі спілкування	+
Відображення доступних користувачу додаткових кіл спілкування	+

Продовження таблиці 4.1

1	2
Відображення запитів у колі спілкування	+
Створення відповіді на запит у колі спілкування	+
Створення запиту в колі спілкування	+
Відображення всіх запитів користувача	+
Відображення всіх нових відповідей на запити користувача	+
Оформлення скарги на дії адміністратора кола спілкування	+
Відображення скарг за адміністратором кола спілкування	+
Відображення скарги на дії адміністратора кола спілкування	+
Відображення всіх отриманих скарг на дії адміністраторів кіл спілкування	+
Пошук серед всіх отриманих скарг на дії адміністраторів кіл спілкування	+
Перевірка реєстрації проживання користувача	+
Редагування власних даних	+
Відображення повного вмісту запитів у колі для спілкування	+
Відображення власних скарг користувача	+

4.5 Висновки за розділом 4

За результатами даного розділу виконано перевірку особливостей виконання тих функцій, які були заплановані для програми, особливостей експлуатації програми різними користувачами, як такими, які є звичайними користувачами-сусідами, так і адміністраторами певних кіл спілкування, адміністраторами всієї програми. Результати перевірки оформлені за допомогою чеклистів, що вказують на успішну реалізацію всіх запланованих функцій.

ВИСНОВКИ

Розроблено програмне забезпечення для обміну запитами з сусідами, щоб підвищити інформованість сусідів та забезпечити обмеження доступу на таку інформацію.

Реалізацію виконано мультипарадигмальними засобами програмування мовою Python в комплексі з фреймворком Django, системою підтримки баз даних MySQL. Забезпечення можливостей неправомірного початкового доступу до керування колами спілкування запропоновано вирішувати за допомогою механізму скарг. Для підтримки цього серед користувачів було виділено адміністраторів програми, для яких, як і для звичайних користувачів, адміністраторів кіл спілкування, побудовано ДБВ.

Створена в підсумку програма дозволяє не тільки обмінюватися запитами з сусідами, але і впорядковувати цей процес для забезпечення обмеження доступу на інформацію, уникнення потрапляння такої інформації до осіб, які могли б її використовувати зі шкідливими цілями в подальшому. Програма для обміну запитами з сусідами надає можливість комунікувати користувачам, які проживають в безпосередній близькості один від одного, обмінюватися інформаційними повідомленнями, відповідати на запитання один одного, звертати увагу один одного, спільно вирішувати проблеми, забезпечуючи для цього як персональну комунікацію з можливістю обмеження її, так і комунікацію в колах спілкування, які враховують принцип сусідства.

Виконано перевірку особливостей виконання тих функцій, які були заплановані для програми, особливостей експлуатації програми різними користувачами, як такими, які є звичайними користувачами-сусідами, так і адміністраторами певних кіл спілкування, адміністраторами всієї програми. Результати перевірки оформлені за допомогою чеклистів, що вказують на успішну реалізацію всіх запланованих функцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nextdoor [Electronic resource]. – Access mode : <https://nextdoor.com/>.
2. Anthony, A. Is media platform Nextdoor a friend in need or a vigilante nightmare? [Electronic resource] / A. Anthony. – Access mode : <https://www.theguardian.com/media/2022/mar/20/is-nextdoor-a-friend-in-need-or-vigilante-nightmare>.
3. Using Nextdoor [Electronic resource]. – Access mode : https://help.nextdoor.com/s/topic-page?topicName=Using-Nextdoor&cTopic=Safety&language=en_US.
4. Django – Full Stack Python [Electronic resource]. – Access mode : <https://www.fullstackpython.com/django.html>.
5. Django Web Framework (Python) [Electronic resource]. – Access mode : <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django>.
6. Tornado Web Server – Tornado 6.4 documentation [Electronic resource]. – Access mode : <https://www.tornadoweb.org/en/stable/>.
7. Tornado Web Server [Electronic resource]. – Access mode : <https://github.com/tornadoweb/tornado>.
8. DarkPan – Bootstrap 5 Admin Dashboard Template [Electronic resource]. – Access mode : <https://htmlcodex.com/bootstrap-5-admin-template/>.

ДОДАТОК А
Технічне завдання

Вступ

Соціальні мережі відомі у всьому світі, впливають на розвиток людства, проте не завжди цей вплив є позитивним. Одним з таких негативних факторів є те, що люди стають менше спілкуватися в реальності, а частиною цієї проблеми є зменшення спілкування з сусідами. Це створює проблеми при переїзді, коли людина потребує адаптації, а значить підтримки оточуючих. Така підтримка потрібна через надання додаткової інформації, відповіді на певні запитання (запити). Окрім того часом потрібно розповсюдити інформацію між сусідами, що є важливою. Але в соціальних мережах друзі обираються не за принципом сусідства загалом. Тож достатньо корисним для забезпечення підтримки щодо окреслених проблем є врахування під час забезпечення комунікації принципу сусідства. Більше того загалом для такого спілкування такий принцип потрібно не тільки враховувати, але і використовувати його в центрі забезпечення безпеки. Це пов'язано з тим, що спілкування між сусідами дозволяє розкрити особи, дозволяє отримати інформацію, яка може бути корисною для шахраїв. Тому принцип сусідства має бути не тільки основою для визначення участі в програмі та взаємодії з іншими, але і заборони на доступ до інформації, яка може бути критичною, ідентифікувати осіб. Саме для надання такої концепції та забезпечення супутніх можливостей має бути направлена програма, що потрібно створити в роботі.

A.1 Підстави для розробки

Наказом № 168 від 24 квітня 2024 р. за Національним університетом «Запорізька політехніка» визначено тему «Програмне забезпечення для обміну запитами з сусідами» для виконання дипломної кваліфікаційної роботи.

A.2 Призначення розробки

Програма для обміну запитами з сусідами має надавати можливість комунікувати користувачам, які проживають в безпосередній близькості один від одного, обмінюватися інформаційними повідомленнями, відповідати на запитання один одного, звертати увагу один одного, спільно вирішувати проблеми, забезпечуючи для цього як персональну комунікацію з можливістю обмеження її, так і комунікацію в колах спілкування, які мають враховувати принцип сусідства.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики програми обміну запитами з сусідами повинні включати:

- реєстрація проживання користувача;
- підтвердження правдивості або неправдивості реєстрації проживання користувача адміністратором базового кола спілкування;
- архівування реєстрації проживання користувачем;
- відображення реєстрацій проживання користувача, включаючи історію та активні;
- відображення кіл спілкування користувача, включаючи ті, де його участь підтверджено, непідтверджено, скасовано, де він є адміністратором, що не є базовими;
- реєстрація в програмі, авторизація користувача або адміністратора;
- перевірка реєстрації проживання користувача;
- відправлення запиту на контакт до сусіда;
- підтвердження або скасування запиту на контакт від сусіда;
- відображення всіх адресованих користувачу запитів на контакт або створених користувачем, включаючи нерозглянуті запити та встановлені

контакти;

- створення персонального запиту до сусіда;
- створення відповіді на персональний запит до сусіда;
- відображення всіх персональних запитів, адресованих користувачу, відправлених користувачем, або тільки за заданою іншою стороною;
- створення кола спілкування серед сусідів;
- архівування кола спілкування серед сусідів;
- підтвердження або скасування участі користувача в колі спілкування;
- подання запиту на участь в колі спілкування;
- відображення доступних користувачу додаткових кіл спілкування;
- відображення запитів у колі спілкування;
- створення відповіді на запит у колі спілкування;
- створення запиту в колі спілкування;
- відображення учасників кола спілкування;
- відображення всіх запитів користувача, включаючи персональні та в колі спілкування;
- відображення всіх нових відповідей на запити користувача;
- оформлення скарги на дії адміністратора кола спілкування;
- відображення скарг за адміністратором кола спілкування;
- відображення скарги на дії адміністратора кола спілкування;
- відображення всіх отриманих скарг на дії адміністраторів кіл спілкування;
- редагування власних даних;
- пошук серед всіх отриманих скарг на дії адміністраторів кіл спілкування.

A.3.2 Умови експлуатації

Експлуатація можлива тільки за використання сервера, на якому можливо встановити і в подальшому встановлено засоби, які включають

інтерпретатор Python версії 3, засоби створення вебдодатків на основі Django, забезпечено доступ до системи керування базами даних MySQL з можливістю програмного створення бази даних засобами фреймворку для розробки вебдодатків Django.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічні засоби визначені на рівні, що має забезпечувати приблизно 50 запитів на секунду:

- процесор не менше Intel Xeon E5440 з частотою 2.83 ГГц;
- оперативна пам'ять – 12 Гб;
- жорсткий диск – 40 Гб простору.

A.4 Порядок контролю та приймання

Виконання дипломної кваліфікаційної роботи бакалавра має бути закінчено у восьми тижневий термін з розподіленням за етапами, що визначено завданням на роботу у відповідному календарному плані. Після виконання даного плану реалізується процедура захисту роботи. Захист відбувається перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

from .models import Neighbor
from .models import Homeregistration
from django.shortcuts import render
from .models import Neighborcircle
from .models import Neighborcirclemember
from .models import Building
from .models import Registrationcheck
from .models import Requesttopic
from .models import Personcontact
from .models import Personrequest
from .models import Personrequestphoto
from .models import Neighborcirclerequest
from .models import Neighborcirclerequestphoto
from .models import Memberclaim

def neighbor_active_homeregistrations(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            home_registrations =
Homeregistration.objects.filter(neighbor = neighbor_authorized,
archived =
False).select_related("neighbor").select_related("building").ord
er_by("-registertime").order_by("-registertime")
            for home_registration in home_registrations:
                home_registration.checks =
Registrationcheck.objects.filter(homeregistration =
home_registration)
            return render(request,
"neirequestapp/activehomes.html", home_registrations)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

def neighbor_history_homeregistrations(request):
    if request.user.is_active:

```

```

        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            home_registrations =
Homeregistration.objects.filter(neighbor =
neighbor_authorized).select_related("neighbor").select_related("
building").order_by("-registertime")
            accumulated_home_registrations = {}
            buildings = []
            for home_registration in home_registrations:
                if home_registration.building not in
buildings:

buildings.append(home_registration.building)

accumulated_home_registrations[home_registration.building.id] =
[home_registration]
            else:
                added_value =
accumulated_home_registrations[home_registration.building.id]
                added_value.append(home_registration)

accumulated_home_registrations[home_registration.building.id] =
added_value
            return render(request,
"neirequestapp/historyhomes.html",
accumulated_home_registrations)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

def all_neighborcircles(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]

```

```

        neighborcirclemembers =
Neighborcirclemember.objects.filter(homeregistration__neighbor =
neighbor_authorized).select_related("neighborcircle").select_rel
ated("homeregistration").select_related("homeregistration__build
ing").order_by("-createtime")
        return render(request,
"neirequestapp/approvedcircles.html", neighborcirclemembers)
    return render(
        request,
        "neirequestapp/mistake.html",
        {
            "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
        }
    )

def approved_neighborcircles(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            neighborcirclemembers =
Neighborcirclemember.objects.filter(homeregistration__neighbor =
neighbor_authorized, approved =
1).select_related("neighborcircle").select_related("homeregistra
tion").select_related("homeregistration__building").order_by("-
createtime")
            return render(request,
"neirequestapp/approvedcircles.html", neighborcirclemembers)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

def unapproved_neighborcircles(request):
    if request.user.is_active:

```

```

        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            neighborcirclemembers =
Neighborcirclemember.objects.filter(homeregistration__neighbor =
neighbor_authorized, approved =
0).select_related("neighborcircle").select_related("homeregistra
tion").select_related("homeregistration__building").order_by("-
createtime")

            return render(request,
"neirequestapp/unapprovedcircles.html", neighborcirclemembers)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

def cancelled_neighborcircles(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            neighborcirclemembers =
Neighborcirclemember.objects.filter(homeregistration__neighbor =
neighbor_authorized, approved =
2).select_related("neighborcircle").select_related("homeregistra
tion").select_related("homeregistration__building").order_by("-
createtime")

            return render(request,
"neirequestapp/canapprovedcircles.html", neighborcirclemembers)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

```

```

    )

    def nonbase_neighborcircles(request):
        if request.user.is_active:
            neighbor_authorized = Neighbor.objects.filter(user
= request.user)
            if neighbor_authorized:
                neighbor_authorized = neighbor_authorized[0]
                neighborcirclemembers =
Neighborcirclemember.objects.filter(homeregistration__neighbor =
neighbor_authorized, neighborcircle__circletype =
False).select_related("neighborcircle").select_related("homeregi
stration").select_related("homeregistration__building").order_by
("-createtime")
                return render(request,
"neirequestapp/nonbasecircles.html", neighborcirclemembers)
            return render(
                request,
                "neirequestapp/mistake.html",
                {
                    "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
                }
            )

    def admin_neighborcircles(request):
        if request.user.is_active:
            neighbor_authorized = Neighbor.objects.filter(user
= request.user)
            if neighbor_authorized:
                neighbor_authorized = neighbor_authorized[0]
                neighborcirclemembers =
Neighborcircle.objects.filter(adminhomeregistration__neighbor =
neighbor_authorized).order_by("-createtime")
                return render(request,
"neirequestapp/admincircles.html", neighborcirclemembers)
            return render(
                request,
                "neirequestapp/mistake.html",
                {
                    "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
                }
            )

```

```

    }
)

def to_personcontacts(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            personcontacts =
Personcontact.objects.filter(dsthomeregistration__neighbor =
neighbor_authorized).order_by("-createtime")
            return render(request,
"neirequestapp/tocontacts.html", personcontacts)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

def from_personcontacts(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            personcontacts =
Personcontact.objects.filter(srchomeregistration__neighbor =
neighbor_authorized).order_by("-createtime")
            return render(request,
"neirequestapp/fromcontacts.html", personcontacts)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

```

```

def pending_to_personcontacts(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            personcontacts =
Personcontact.objects.filter(dsthomeregistration__neighbor =
neighbor_authorized, resulttype = 0).order_by("-createtime")
            return render(request,
"neirequestapp/pendtocontacts.html", personcontacts)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

def pending_from_personcontacts(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            personcontacts =
Personcontact.objects.filter(srchomeregistration__neighbor =
neighbor_authorized, resulttype = 0).order_by("-createtime")
            return render(request,
"neirequestapp/pendfromcontacts.html", personcontacts)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

from django.db.models import Q

```

```

def approved_personcontacts(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            personcontacts =
Personcontact.objects.filter(Q(dsthomeregistration__neighbor =
neighbor_authorized) | Q(srchomeregistration__neighbor =
neighbor_authorized), resulttype = 1).order_by("-createtime")
            return render(request,
"neirequestapp/approvedcontacts.html", personcontacts)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

def to_personrequests(request):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            personrequests =
Personrequest.objects.filter(dsthomeregistration__neighbor =
neighbor_authorized, originalrequest__isnull = True).order_by("-
requesttime")

            for personrequest in personrequests:
                personrequest.answers =
Personrequest.objects.filter(originalrequest =
personrequest).order_by("requesttime")
                personrequest.photos =
Personrequestphoto.objects.filter(personrequest = personrequest)
                for answer in personrequest.answers:
                    answer.photos =
Personrequestphoto.objects.filter(personrequest = answer)
            return render(request,
"neirequestapp/topersonrequests.html", personrequests)
        return render(

```

```

        request,
        "neirequestapp/mistake.html",
        {
            "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
        }
    )

def from_personrequests(request, requesttopicid = -1):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            if requesttopicid != -1:
                request_topic =
Requesttopic.objects.filter(id = requesttopicid)
                request_topic = request_topic[0]
                personrequests =
Personrequest.objects.filter(srchomeregistration__neighbor =
neighbor_authorized, originalrequest__isnull = True,
requesttopic = request_topic).order_by("-requesttime")
            else:
                personrequests =
Personrequest.objects.filter(srchomeregistration__neighbor =
neighbor_authorized, originalrequest__isnull = True).order_by("-
requesttime")

            for personrequest in personrequests:
                personrequest.answers =
Personrequest.objects.filter(originalrequest =
personrequest).order_by("requesttime")
                personrequest.photos =
Personrequestphoto.objects.filter(personrequest = personrequest)
                for answer in personrequest.answers:
                    answer.photos =
Personrequestphoto.objects.filter(personrequest = answer)
            return render(request,
"neirequestapp/frompersonrequests.html", personrequests)
        return render(
            request,
            "neirequestapp/mistake.html",
            {

```

```

        "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
    }
)

```

```

def from_to_personrequests(request, neighbor_username,
requesttopicid = -1):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            dst_neighbor =
Neighbor.objects.filter(user__username = neighbor_username)
            if requesttopicid != -1:
                request_topic =
Requesttopic.objects.filter(id = requesttopicid)
                request_topic = request_topic[0]
                personrequests =
Personrequest.objects.filter(Q(srchomeregistration__neighbor =
neighbor_authorized) | Q(dsthomeregistration__neighbor =
dst_neighbor), Q(srchomeregistration__neighbor = dst_neighbor) |
Q(dsthomeregistration__neighbor = neighbor_authorized) ,
originalrequest__isnull = True, requesttopic =
request_topic).order_by("-requesttime")
            else:
                personrequests =
Personrequest.objects.filter(Q(srchomeregistration__neighbor =
neighbor_authorized) | Q(dsthomeregistration__neighbor =
dst_neighbor), Q(srchomeregistration__neighbor = dst_neighbor) |
Q(dsthomeregistration__neighbor = neighbor_authorized) ,
originalrequest__isnull = True).order_by("-requesttime")
                for personrequest in personrequests:
                    personrequest.answers =
Personrequest.objects.filter(originalrequest =
personrequest).order_by("requesttime")
                    personrequest.photos =
Personrequestphoto.objects.filter(personrequest = personrequest)
                    for answer in personrequest.answers:
                        answer.photos =
Personrequestphoto.objects.filter(personrequest = answer)
                return render(request,
"neirequestapp/fromtopersonrequests.html", personrequests)

```

```

        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
            }
        )

    def circle_requests(request, neighborcircleid,
requesttopicid = -1):
        if request.user.is_active:
            neighbor_authorized = Neighbor.objects.filter(user
= request.user)
            if neighbor_authorized:
                neighbor_authorized = neighbor_authorized[0]
                neighbor_circle =
Neighborcircle.objects.filter(id =
neighborcircleid).select_related("adminhomeregistration").select
_related("adminhomeregistration__building").select_related("admi
nhomeregistration__neighbor")
                neighbor_circle = neighbor_circle[0]
                neighbor_member =
Neighborcirclemember.objects.filter(neighborcircle =
neighbor_circle, homeregistration__neighbor =
neighbor_authorized)
                access_to_circle = 0
                circlerequests = []
                if not neighbor_circle.archived:
                    if (neighbor_member.approved == 1) and (not
neighbor_member.blocked) and (not neighbor_member.archived):
                        access_to_circle = 1
                    elif (neighbor_member.approved == 0) and
(not neighbor_member.blocked) and (not neighbor_member.archived)
and neighbor_circle.widetype:
                        access_to_circle = 2
                if access_to_circle == 0:
                    return render(
                        request,
                        "neirequestapp/mistake.html",
                        {
                            "reason": "Ви не маєте прав доступу
до цієї спільноти на перегляд запитів."
                        }
                    )

```

```

        }
    )
    elif access_to_circle == 1:
        if requesttopicid != -1:
            request_topic =
Requesttopic.objects.filter(id = requesttopicid)
            request_topic = request_topic[0]
            personrequests =
Personrequest.objects.filter(Q(srchomeregistration__neighbor =
neighbor_authorized) | Q(dsthomeregistration__neighbor =
dst_neighbor), Q(srchomeregistration__neighbor = dst_neighbor) |
Q(dsthomeregistration__neighbor = neighbor_authorized) ,
originalrequest__isnull = True, requesttopic = request_topic,
requesttopic = request_topic).order_by("-requesttime")
        else:
            personrequests =
Personrequest.objects.filter(Q(srchomeregistration__neighbor =
neighbor_authorized) | Q(dsthomeregistration__neighbor =
dst_neighbor), Q(srchomeregistration__neighbor = dst_neighbor) |
Q(dsthomeregistration__neighbor = neighbor_authorized) ,
originalrequest__isnull = True, requesttopic =
request_topic).order_by("-requesttime")
        else:
            if requesttopicid != -1:
                request_topic =
Requesttopic.objects.filter(id = requesttopicid)
                request_topic = request_topic[0]
                circlerequests =
Neighborcirclerequest.objects.filter(neighborcircle =
neighbor_circle, originalrequest__isnull = True, requesttopic =
request_topic).exclude(notseens__in =
[neighbor_authorized]).order_by("-requesttime")
            else:
                circlerequests =
Neighborcirclerequest.objects.filter(neighborcircle =
neighbor_circle, originalrequest__isnull =
True).exclude(notseens__in = [neighbor_authorized]).order_by("-
requesttime")

    elif access_to_circle == 2:
        if requesttopicid != -1:
            request_topic =
Requesttopic.objects.filter(id = requesttopicid)
            request_topic = request_topic[0]

```

```

        circlerequests =
Neighborcirclerequest.objects.filter(neighborcircle =
neighbor_circle, originalrequest__isnull = True, widetype =
True, requesttopic = request_topic).exclude(notseens__in =
[neighbor_authorized]).order_by("-requesttime")
    else:
        circlerequests =
Neighborcirclerequest.objects.filter(neighborcircle =
neighbor_circle, originalrequest__isnull = True, widetype =
True).exclude(notseens__in = [neighbor_authorized]).order_by("-
requesttime")

        for circlerequest in circlerequests:
            circlerequest.answers =
Neighborcirclerequest.objects.filter(originalrequest =
circlerequest).order_by("requesttime")
            circlerequest.photos =
Neighborcirclerequestphoto.objects.filter(neighborcirclerequest
= circlerequest)

            for answer in circlerequest.answers:
                answer.photos =
Neighborcirclerequestphoto.objects.filter(neighborcirclerequest
= answer)

        return render(request,
"neirequestapp/circlerequests.html", {"circle": neighbor_circle,
"member": neighbor_member, "requests": circlerequests })
    return render(
        request,
        "neirequestapp/mistake.html",
        {
            "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
        }
    )

def from_requests(request, requesttopicid = -1):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            if requesttopicid != -1:
                request_topic =
Requesttopic.objects.filter(id = requesttopicid)

```

```

        request_topic = request_topic[0]
        personrequests =
Personrequest.objects.filter(srchomeregistration__neighbor =
neighbor_authorized, originalrequest__isnull = True,
requesttopic = request_topic).order_by("-requesttime")
    else:
        personrequests =
Personrequest.objects.filter(srchomeregistration__neighbor =
neighbor_authorized, originalrequest__isnull = True).order_by("-
requesttime")

        for personrequest in personrequests:
            personrequest.answers =
Personrequest.objects.filter(originalrequest =
personrequest).order_by("requesttime")
            personrequest.photos =
Personrequestphoto.objects.filter(personrequest = personrequest)
            for answer in personrequest.answers:
                answer.photos =
Personrequestphoto.objects.filter(personrequest = answer)
            if requesttopicid != -1:
                request_topic =
Requesttopic.objects.filter(id = requesttopicid)
                request_topic = request_topic[0]
                circlerequests =
Neighborcirclerequest.objects.filter(homeregistration__neighbor
= neighbor_authorized, originalrequest__isnull = True,
requesttopic = request_topic).order_by("-requesttime")
            else:
                circlerequests =
Neighborcirclerequest.objects.filter(homeregistration__neighbor
= neighbor_authorized, originalrequest__isnull =
True).order_by("-requesttime")
                for circlerequest in circlerequests:
                    circlerequest.answers =
Neighborcirclerequest.objects.filter(originalrequest =
circlerequest).order_by("requesttime")
                    circlerequest.photos =
Neighborcirclerequestphoto.objects.filter(neighborcirclerequest
= circlerequest)
                    for answer in circlerequest.answers:
                        answer.photos =
Neighborcirclerequestphoto.objects.filter(neighborcirclerequest
= answer)

```



```

        answer.photos =
Personrequestphoto.objects.filter(personrequest = answer)
        person_answers.append(personrequest)
        if requesttopicid != -1:
            request_topic =
Requesttopic.objects.filter(id = requesttopicid)
            request_topic = request_topic[0]
            circlerequests =
Neighborcirclerequest.objects.filter(homeregistration__neighbor
= neighbor_authorized, originalrequest__isnull = True,
requesttopic = request_topic).order_by("-requesttime")
        else:
            circlerequests =
Neighborcirclerequest.objects.filter(homeregistration__neighbor
= neighbor_authorized, originalrequest__isnull =
True).order_by("-requesttime")
            circleanswers =
Neighborcirclerequest.objects.filter(originalrequest__in =
circlerequests).order_by("-requesttime")
            circle_answers = []
            for circlerequest in circlerequests:
                if circlerequest.originalrequest not in
circle_answers:
                    circlerequest.answers =
Neighborcirclerequest.objects.filter(originalrequest =
circlerequest).order_by("requesttime")
                    circlerequest.photos =
Neighborcirclerequestphoto.objects.filter(neighborcirclerequest
= circlerequest)
                    for answer in circlerequest.answers:
                        answer.photos =
Neighborcirclerequestphoto.objects.filter(neighborcirclerequest
= answer)
                    circle_answers.append(circlerequest)
            return render(request,
"neirequestapp/newanswers.html", {"person_answers":
person_answers, "circle_answers": circle_answers })
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."

```

```

    }
)

def memberclaim_content(request, memberclaimid):
    if request.user.is_active:
        neighbor_authorized = Neighbor.objects.filter(user
= request.user)
        if neighbor_authorized:
            neighbor_authorized = neighbor_authorized[0]
            memberclaim = Memberclaim.objects.filter(id =
memberclaimid).select_related("neighborcirclemember").select_rel
ated("neighborcirclemember_homeregistration").select_related("ne
ighborcirclemember_homeregistration__neighbor)
            memberclaim = memberclaim[0]
            if
memberclaim.neighborcirclemember.homeregistration.neighbor ==
neighbor_authorized or request.user.is_superuser == True:
                return render(request,
"neirequestapp/memberclaim.html", memberclaim)
            return render(
                request,
                "neirequestapp/mistake.html",
                {
                    "reason": "Ви повинні бути користувачем, який
розглядається як сусід або адміністратором, і спершу
авторизуватися. Інакше ця функція недоступна."
                }
            )

def admin_memberclaims(request, admin_username):
    if request.user.is_active and
request.user.is_superuser:
        neighbor_requested =
Neighbor.objects.filter(user__username = admin_username)
        if neighbor_requested:
            neighbor_requested = neighbor_requested[0]
            memberclaims =
Memberclaim.objects.filter(neighborcirclemember__neighborcircle
__adminhomeregistration__neighbor =
neighbor_requested).select_related("neighborcirclemember").selec
t_related("neighborcirclemember_homeregistration").select_relate
d("neighborcirclemember_homeregistration__neighbor).order_by("-
createtime")

```

```

        return render(request,
"neirequestapp/adminmemberclaims.html", memberclaims)
    return render(
        request,
        "neirequestapp/mistake.html",
        {
            "reason": "Ви повинні бути користувачем, який є
адміністратором програми. Інакше ця функція недоступна."
        }
    )

    def all_memberclaims(request):
        if request.user.is_active and
request.user.is_superuser:
            memberclaims =
Memberclaim.objects.all().select_related("neighborcirclemember")
.select_related("neighborcirclemember_homeregistration").select_
related("neighborcirclemember_homeregistration__neighbor).order_
by("-createtime")
            return render(request,
"neirequestapp/memberclaims.html", memberclaims)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який є
адміністратором програми. Інакше ця функція недоступна."
            }
        )

    def new_memberclaims(request, strict = 0):
        if request.user.is_active and
request.user.is_superuser:
            if strict == 0:
                memberclaims =
Memberclaim.objects.filter(assignedstatus__in =
[0,1]).select_related("neighborcirclemember").select_related("ne
ighborcirclemember_homeregistration").select_related("neighborci
rclemember_homeregistration__neighbor).order_by("-createtime")
            elif strict == 1:
                memberclaims =
Memberclaim.objects.filter(assignedstatus =
0).select_related("neighborcirclemember").select_related("neighb

```

```

orcirclemember_homeregistration").select_related("neighborcircle
member_homeregistration__neighbor).order_by("-createtime")
        elif strict == 2:
            memberclaims =
Memberclaim.objects.filter(assignedstatus =
1).select_related("neighborcirclemember").select_related("neighb
orcirclemember_homeregistration").select_related("neighborcircle
member_homeregistration__neighbor).order_by("-createtime")
            return render(request,
"neirequestapp/newmemberclaims.html", memberclaims)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який є
адміністратором програми. Інакше ця функція недоступна."
            }
        )

def solved_memberclaims(request, strict = 0):
    if request.user.is_active and
request.user.is_superuser:
        if strict == 0:
            memberclaims =
Memberclaim.objects.filter(assignedstatus__in =
[2,3,4]).select_related("neighborcirclemember").select_related("neigh
borcirclemember_homeregistration").select_related("neighborci
rclemember_homeregistration__neighbor).order_by("-createtime")
            elif strict == 1:
                memberclaims =
Memberclaim.objects.filter(assignedstatus__in =
[2,4]).select_related("neighborcirclemember").select_related("neigh
borcirclemember_homeregistration").select_related("neighborci
rclemember_homeregistration__neighbor).order_by("-createtime")
            elif strict == 2:
                memberclaims =
Memberclaim.objects.filter(assignedstatus =
2).select_related("neighborcirclemember").select_related("neighb
orcirclemember_homeregistration").select_related("neighborcircle
member_homeregistration__neighbor).order_by("-createtime")
            elif strict == 3:
                memberclaims =
Memberclaim.objects.filter(assignedstatus =
3).select_related("neighborcirclemember").select_related("neighb

```

```

orcirclemember_homeregistration").select_related("neighborcircle
member_homeregistration__neighbor).order_by("-createtime")
        elif strict == 4:
            memberclaims =
Memberclaim.objects.filter(assignedstatus =
4).select_related("neighborcirclemember").select_related("neighb
orcirclemember_homeregistration").select_related("neighborcircle
member_homeregistration__neighbor).order_by("-createtime")
            return render(request,
"neirequestapp/solvedmemberclaims.html", memberclaims)
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Ви повинні бути користувачем, який є
адміністратором програми. Інакше ця функція недоступна."
            }
        )

def search_memberclaims(request):
    if request.user.is_active and
request.user.is_superuser:
        memberclaims =
Memberclaim.objects.all().select_related("neighborcirclemember")
.select_related("neighborcirclemember_homeregistration").select_
related("neighborcirclemember_homeregistration__neighbor")
        claimstatus = self.request.GET.get('status', None)
        if claimstatus:
            claimstatus = int(claimstatus)
            claimstatus = claimstatus - 1
            memberclaims =
memberclaims.filter(assignedstatus = claimstatus)
            time_restriction = self.request.GET.get('time1',
None)

            if time_restriction:
                time_restriction =
datetime.strptime(time_restriction, '%d/%m/%y')
                memberclaims =
memberclaims.filter(createtime__gte = time_restriction)
            time_restriction = self.request.GET.get('time2',
None)

            if time_restriction:
                time_restriction =
datetime.strptime(time_restriction, '%d/%m/%y')

```

```

        memberclaims =
memberclaims.filter(createtime__lte = time_restriction)
        circle_name = self.request.GET.get('circle', None)
        if circle_name:
            memberclaims =
memberclaims.filter(neighborcirclemember__neighborcircle__circle
name__icontains = circle_name)
            circle_type = self.request.GET.get('type', None)
            if circle_type:
                circle_type = int(circle_type)
                if circle_type == 1:
                    memberclaims =
memberclaims.filter(neighborcirclemember__neighborcircle__circle
type = True)
                    elif circle_type == 2:
                        memberclaims =
memberclaims.filter(neighborcirclemember__neighborcircle__circle
type = False)
                        circle_archived = self.request.GET.get('archived',
None)
                        if circle_archived:
                            circle_archived = int(circle_archived)
                            if circle_archived == 1:
                                memberclaims =
memberclaims.filter(neighborcirclemember__neighborcircle__archiv
ed = True)
                                elif circle_archived == 2:
                                    memberclaims =
memberclaims.filter(neighborcirclemember__neighborcircle__archiv
ed = False)
                                    admin_name = self.request.GET.get('admin', None)
                                    if admin_name:
                                        memberclaims =
memberclaims.filter(neighborcirclemember__neighborcircle__adminh
omeregistration__neighbor__user__username = admin_name)
                                        citylocation = self.request.GET.get('city', None)
                                        if citylocation:
                                            memberclaims =
memberclaims.filter(neighborcirclemember__neighborcircle__adminh
omeregistration__building__location = citylocation)
                                            back_order = self.request.GET.get('order', None)
                                            if back_order:
                                                memberclaims =
memberclaims.order_by("createtime")

```



```

        circle_distance =
hs.haversine((home_registration.checklat,
home_registration.checklong),
(city_circle.adminhomeregistration.checklat,
city_circle.adminhomeregistration.checklong),
unit=Unit.KILOMETERS)

        if circle_distance <= 0.1:
            answer_circles.append(city_circle)
        return render(request,
"neirequestapp/possiblecircles.html", answer_circles)
    else:
        return render(
            request,
            "neirequestapp/mistake.html",
            {
                "reason": "Коло спілкування задано
невірно."
            }
        )
    return render(
        request,
        "neirequestapp/mistake.html",
        {
            "reason": "Ви повинні бути користувачем, який
розглядається як сусід, і спершу авторизуватися. Інакше ця
функція недоступна."
        }
    )

```

ДОДАТОК В
Слайди презентації

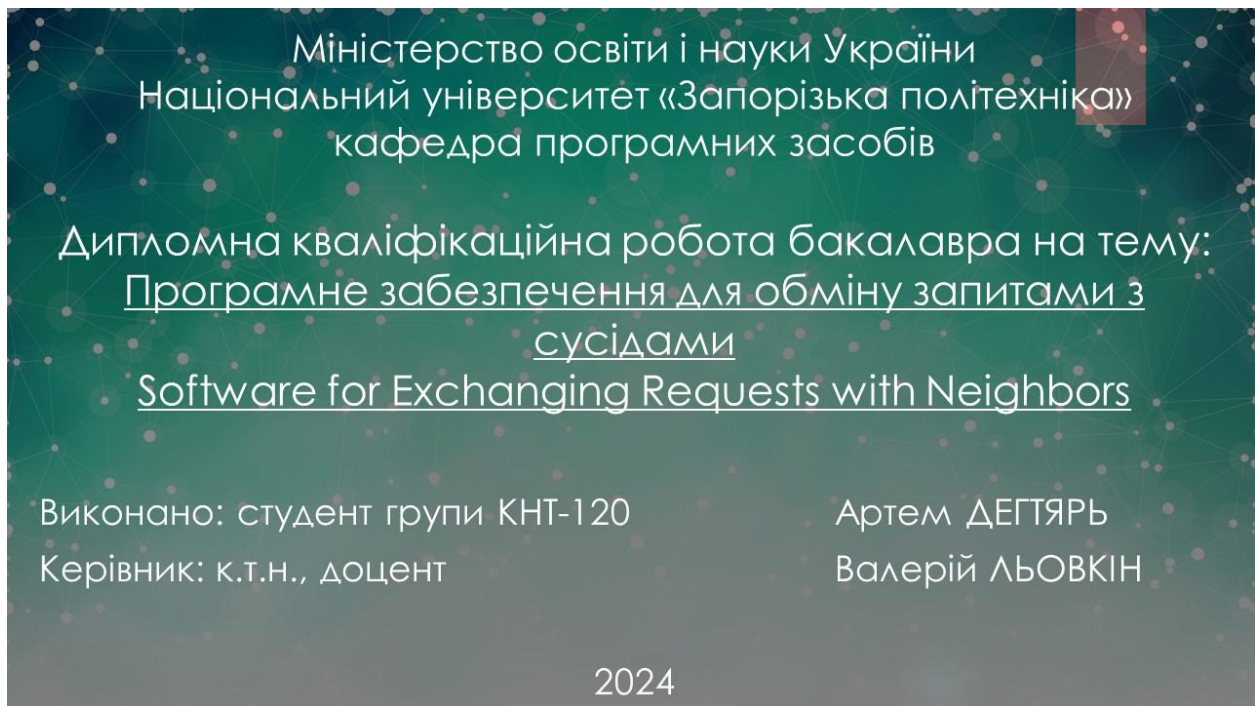


Рисунок В.1 – Перший слайд

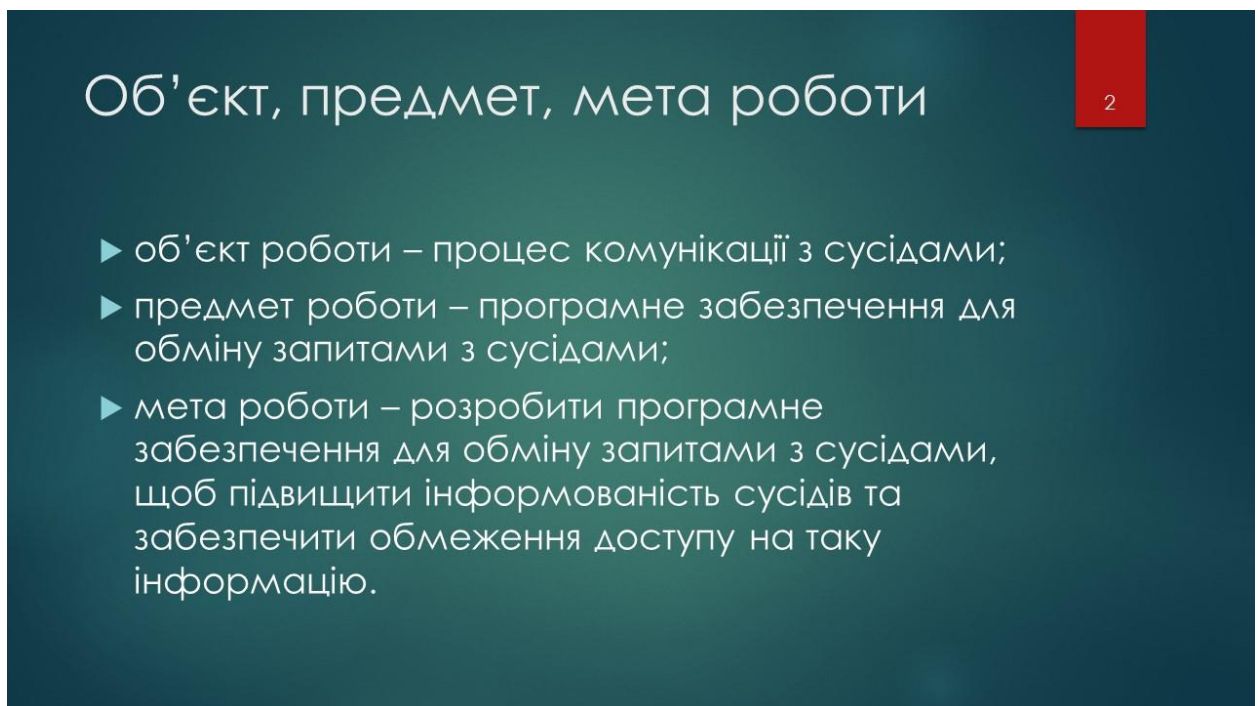


Рисунок В.2 – Об'єкт, предмет, мета роботи

Порівняння існуючих аналогів та розроблюваного програмного забезпечення для обміну запитами з сусідами

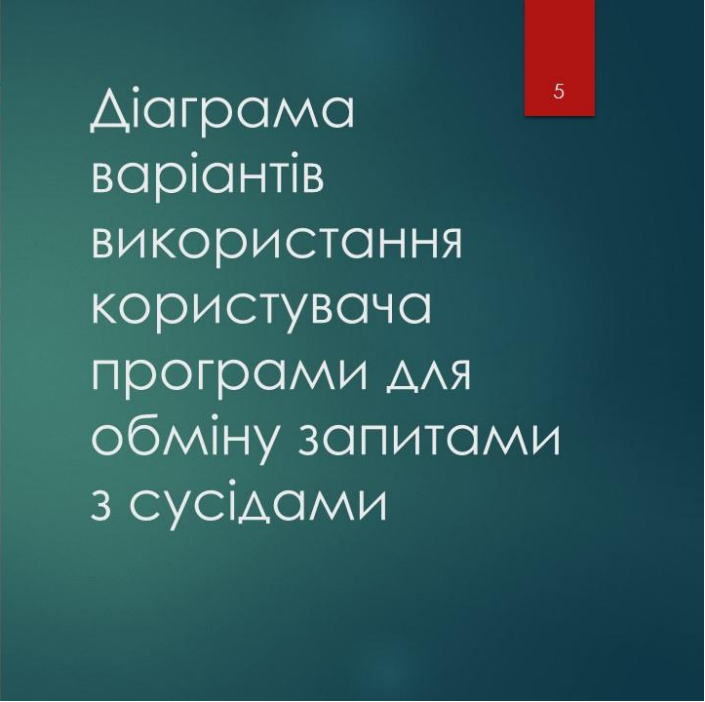
Критерій	Nextdoor	Neirequestapp
Можливість участі в декількох групах сусідів	+	+
Варіанти комунікації	Персональне, в групах	Персональне, в групах
Підхід до визначення меж групи	Адміністратор з врахуванням геолокації, максимально – місто	Геолокація, тільки сусідні будинки
Об'єднання запитів	Гештеги	Типи
Підтвердження доступу користувача адміністратором	–	+
Реклама місцевого бізнесу	+	–

Рисунок В.3 – Порівняння існуючих аналогів та розроблюваного програмного забезпечення для обміну запитами з сусідами

Порівняння фреймворків розробки програми для обміну запитами з сусідами

Критерій	Django	Tornado
Ліцензія	Відкрита, 3 пунктів	Відкрита, Apache License
Підтримка баз даних	Фреймворк об'єктно-реляційного відображення вбудований	Необхідно окремо налаштовувати, доступні фреймворки від інших розробників
Документація та простора застосування	Доступно багато варіантів, докладно описано	Менша наявної документації, окремі проблеми може бути не так легко вирішити
Керування через адміністративну панель	Вбудоване	Проблемне, немає готових для підключення засобів для реалізації

Рисунок В.4 – Порівняння фреймворків розробки програми для обміну запитами з сусідами



Діаграми варіантів використання програми для обміну запитами з сусідами адміністраторами

Рисунок В.6 – Діаграми варіантів використання програми для обміну
запитами з сусідами адміністраторами

Схема бази даних програми для обміну запитами з сусідами

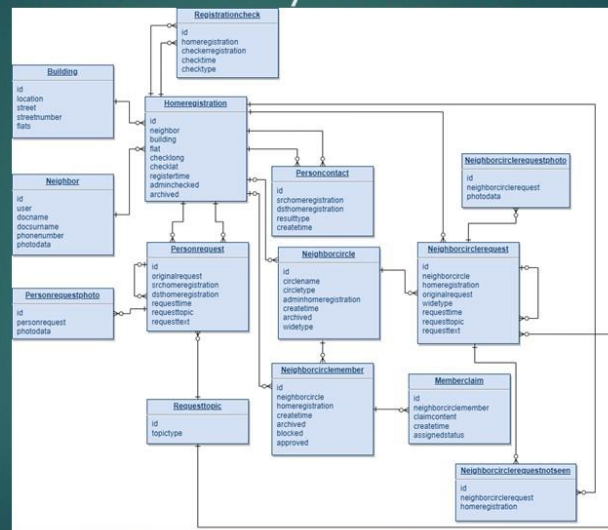


Рисунок В.7 – Схема бази даних програми для обміну запитами з сусідами

Схема порядку формування переліку запитів для відображення користувачу в колі спілкування

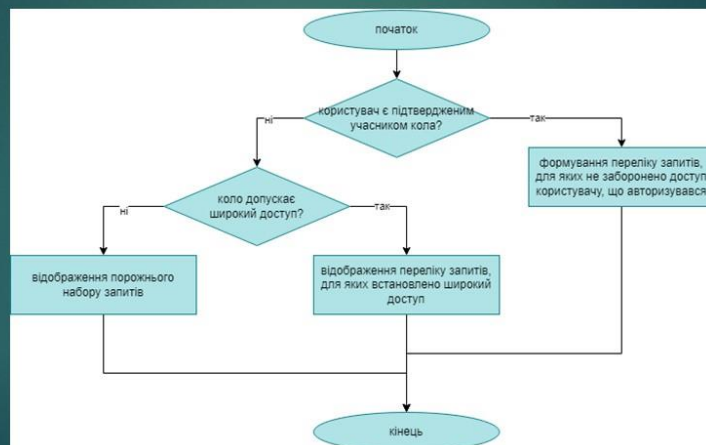


Рисунок В.8 – Схема порядку формування переліку запитів для відображення користувачу в колі спілкування

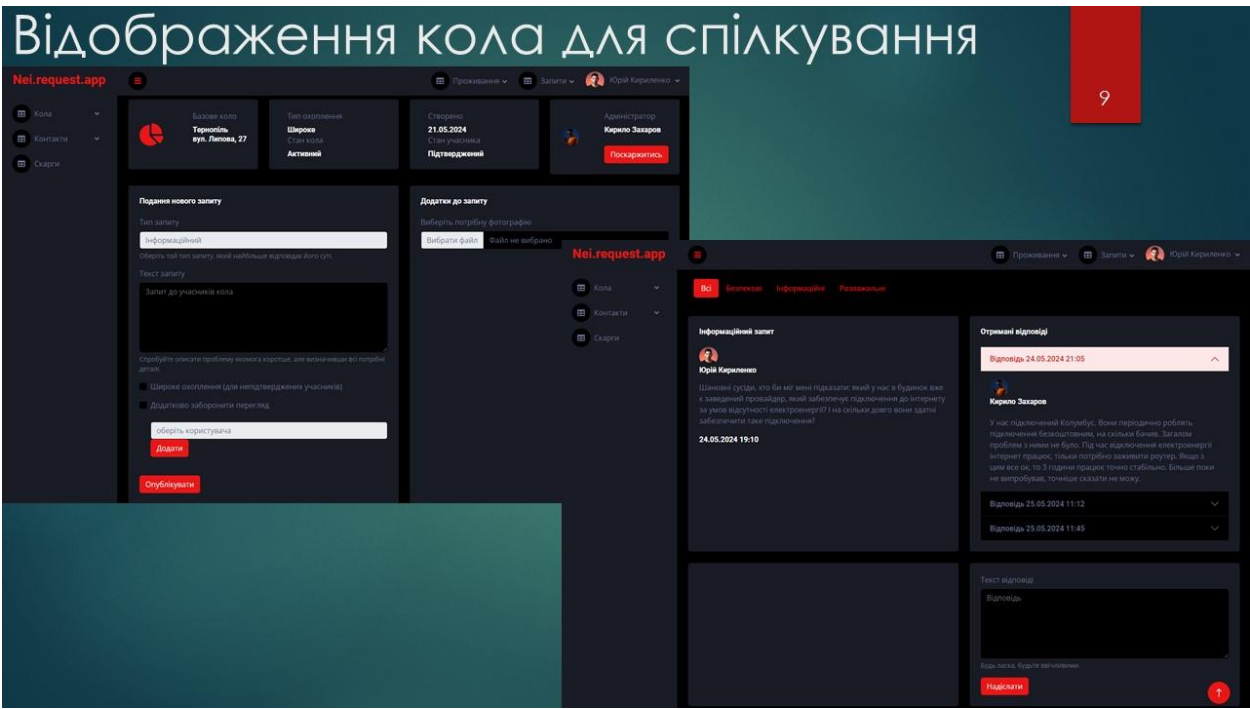


Рисунок В.9 – Відображення кола для спілкування

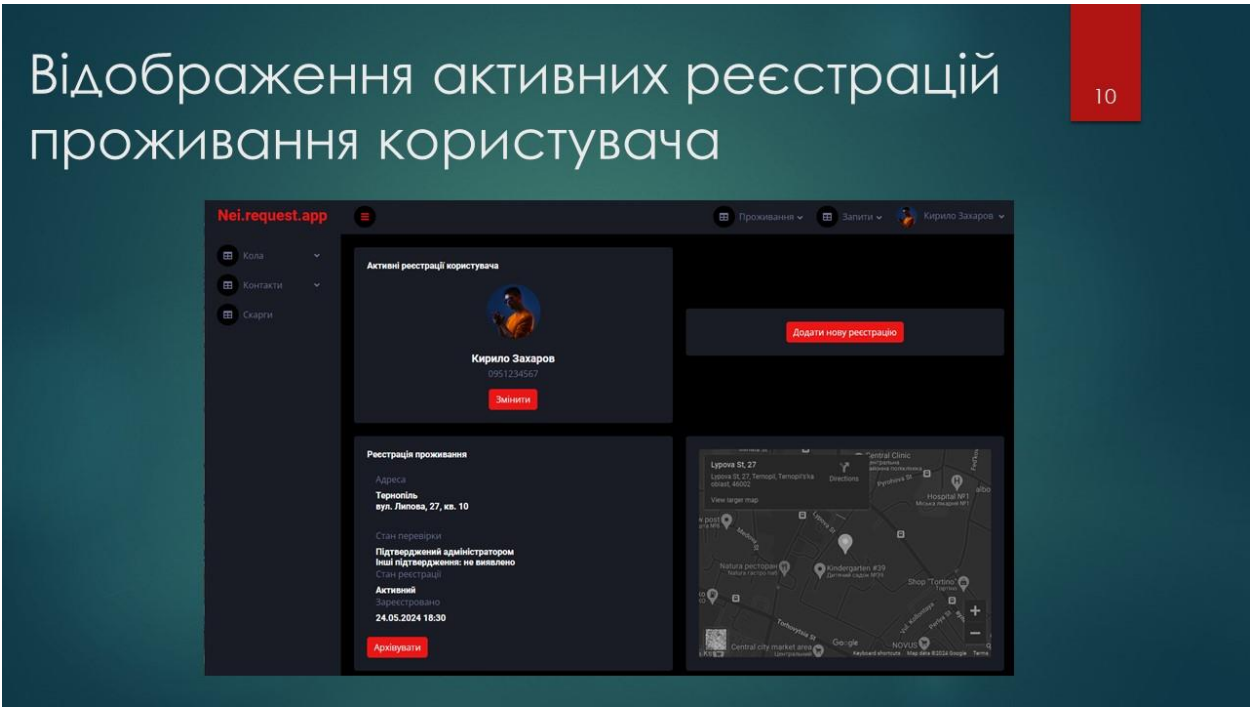


Рисунок В.10 – Відображення активних реєстрацій проживання користувача

Виконання функцій, пов'язаних з адмініструванням

11

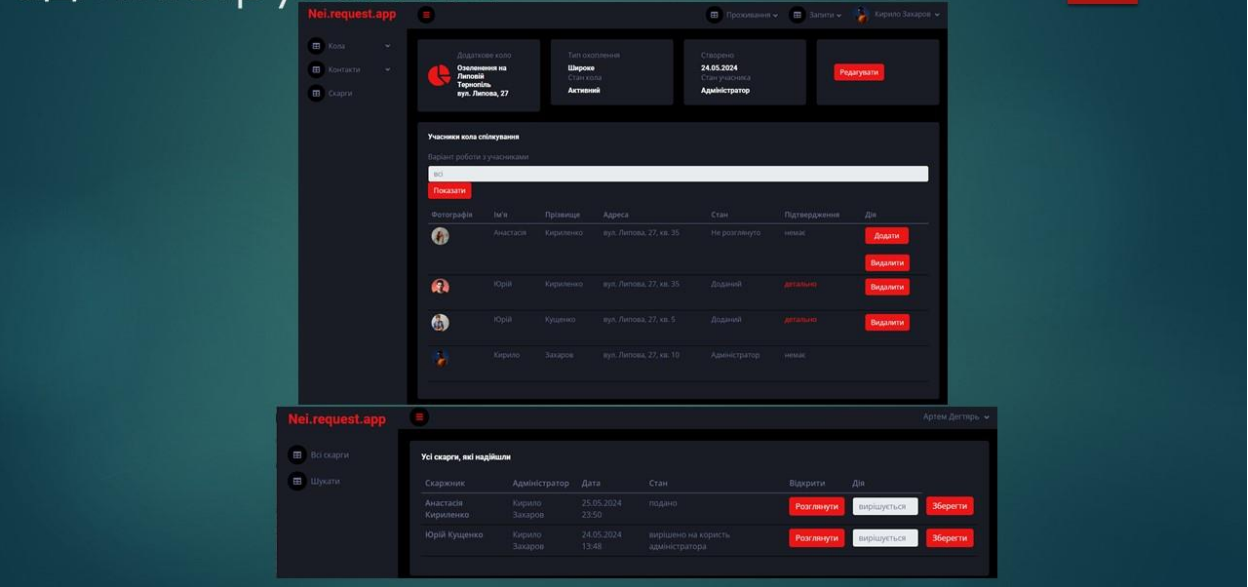


Рисунок В.11 – Виконання функцій, пов'язаних з адмініструванням

Результати тестування програми обміну запитами з сусідами

12

Функціональна можливість	Результат	Функціональна можливість	Результат
Реєстрація проживання користувача	+	Відображення кіл спілкування користувача, що не є базовими	+
Підтвердження правдивості реєстрації проживання користувача адміністратором базового кола спілкування	+	Реєстрація в програмі користувача	+
Підтвердження неправдивості реєстрації проживання користувача адміністратором базового кола спілкування	+	Авторизація адміністратора	+
Архівування реєстрації проживання користувачем	+	Авторизація користувача	+
Відображення історії реєстрацій проживання користувача	+	Відображення учасників кола спілкування	+
Відображення активних реєстрацій проживання користувача	+	Керування скаргю	+
Відображення всіх кіл спілкування користувача	+	Відправлення запиту на контакт до сусіда	+
Відображення кіл спілкування користувача, де його участь підтверджено	+	Підтвердження запиту на контакт від сусіда	+
Відображення кіл спілкування користувача, де його участь непідтверджено	+	Скасування запиту на контакт від сусіда	+
Відображення кіл спілкування користувача, де його участь скасовано	+	Відображення всіх адресованих користувачу запитів на контакт	+
Відображення кіл спілкування користувача, де він є адміністратором	+	Відображення всіх створених користувачем запитів на контакт	+
		Відображення всіх встановлених контактів користувача	+

Рисунок В.12 – Результати тестування програми обміну запитами з сусідами

Результати тестування програми обміну запитами з сусідами

13

Функціональна можливість	Результат	Функціональна можливість	Результат
Відображення всіх адресованих користувачу запитів на контакт, які не розглянуто	+	Скасування участі користувача в колі спілкування	+
Відображення всіх створених користувачем запитів на контакт, які не розглянуто	+	Подання запиту на участь в колі спілкування	+
Створення персонального запиту до сусіда	+	Відображення доступних користувачу додаткових кіл спілкування	+
Створення відповіді на персональний запит до сусіда	+	Відображення запитів у колі спілкування	+
Відображення всіх персональних запитів, адресованих користувачу	+	Створення відповіді на запит у колі спілкування	+
Відображення всіх персональних запитів, відправлених користувачем	+	Створення запиту в колі спілкування	+
Відображення всіх персональних запитів за заданою іншою стороною	+	Відображення всіх запитів користувача	+
Створення кола спілкування серед сусідів	+	Відображення всіх нових відповідей на запити користувача	+
Архівування кола спілкування серед сусідів	+	Оформлення скарги на дії адміністратора кола спілкування	+
Підтвердження участі користувача в колі спілкування	+	Відображення скарг за адміністратором кола спілкування	+
Відображення повного вмісту запитів у колі спілкування	+	Відображення скарги на дії адміністратора кола спілкування	+
		Відображення всіх отриманих скарг на дії адміністраторів кіл спілкування	+
		Пошук серед всіх отриманих скарг на дії адміністраторів кіл спілкування	+
		Перевірка реєстрації проживання користувача	+
		Редагування власних даних	+
		Відображення власних скарг користувача	+

Рисунок В.13 – Результати тестування програми обміну запитами з сусідами

Висновки

14

- ▶ розроблено програмне забезпечення для обміну запитами з сусідами, щоб підвищити інформованість сусідів та забезпечити обмеження доступу на таку інформацію;
- ▶ реалізацію виконано мультипарадигмальними засобами програмування мовою Python в комплексі з фреймворком Django, системою підтримки баз даних MySQL;
- ▶ виконано перевірку особливостей виконання тих функцій, які були заплановані для програми, особливостей експлуатації програми різними користувачами, як такими, які є звичайними користувачами-сусідами, так і адміністраторами певних кіл спілкування, адміністраторами всієї програми. Результати перевірки оформлені за допомогою чеклистів, що вказують на успішну реалізацію всіх запланованих функцій.

Рисунок В.14 – Висновки