

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 МОНІТОРИНГУ

(назва теми)

ПІДОЗРІЛОЇ АКТИВНОСТІ В ЖИТЛОВОМУ РАЙОНІ

SOFTWARE TO MONITOR SUSPICIOUS ACTIVITY IN

RESIDENTIAL AREA

Виконав(ла): студент(ка) 4 курсу, групи КНТ-210сп

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ДЕМЧЕНКО М.Ю.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ТІМЕНКО А.В.

(ПРИЗВИЩЕ та ініціали)

2023_

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ ФКНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ бакалавр _____
Спеціальність _____ 122 Комп'ютерні науки _____
(код і найменування)
Освітня програма (спеціалізація) _____ Комп'ютерні науки _____
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« _____ » _____ 2023 року

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

ДЕМЧЕНКА Микити Юрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення моніторингу підозрілої активності в житловому районі. Software to Monitor Suspicious Activity in Residential Area

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 « 04 » квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 12 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, 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	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 11 » квітня 2023 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)

_____ Микита ДЕМЧЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

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

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 92 с., 22 рис., 3 табл., 3 дод., 10 джерел.

ПІДОЗРІЛА АКТИВНІСТЬ, ЖИТЛОВИЙ РАЙОН, МОНІТОРИНГ, СПОВІЩЕННЯ, ВЕБРОЗРОБКА.

Об'єкт роботи – процес моніторингу підозрілої активності в житловому районі. Предмет роботи – програмне забезпечення моніторингу підозрілої активності в житловому районі.

Мета роботи – розробити програмне забезпечення моніторингу підозрілої активності в житловому районі для допомоги виявлення і вирішення безпекових проблем людям всередині спільноти, з представниками якої вони проживають поруч.

Матеріали, методи та технічні засоби: мова програмування Python, засоби веброзробки, до яких відносяться мови HTML, CSS, JavaScript та фреймворк Django, та система MySQL для керування базою даних.

Результати. Створена веброзробка визначає структуру роботи в цій сфері, включаючи визначені рівні доступу, основні виділені сутності, забезпечені процеси щодо взаємодії між користувачами, виконане проєктування і безпосередньо саму розробку. Усе це дозволяє в підсумку створити певну спільноту всередині програми, в якій люди готові допомагати один одному зокрема стосовно захисту власного майна.

Висновки. Програма призначена для внесення даних про виявлені підозрілі активності, для інформування користувачів з деякої групи про виявлені підозрілі активності, враховуючи рівні доступу, для відслідковування наявної статистики підозрілої активності. Користувачі можуть просити про термінову допомогу, звертатися з проханням перевірити стан будинку.

Галузь використання. Спостереження, безпека власного майна.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 92 pages, 22 figures, 3 tables, 3 appendixes, 10 sources.

SUSPICIOUS ACTIVITY, RESIDENTIAL AREA, MONITORING, NOTIFICATION, WEB DEVELOPMENT.

The object of work is the process of monitoring suspicious activity in a residential area. The subject of the work is software for monitoring suspicious activity in a residential area.

The goal of the work is to develop software for monitoring suspicious activity in a residential area to help identify and resolve security issues for people within the community they live next to.

Materials, methods and technical means: Python programming language, web development tools, which include HTML, CSS, JavaScript and the Django framework, and the MySQL database management system.

The results. The created web application determines the structure of work in this area, including the defined access levels, the main entities, the provided processes for the interaction between users, the completed design and the development itself. All this ultimately allows to create a certain community within the program, in which people are ready to help each other, in particular, with regard to the protection of their own property.

Conclusions. The program is designed to enter data on detected suspicious activities, to inform users from a certain group about detected suspicious activities, taking into account the access levels, to monitor available statistics of suspicious activity. Users can ask for urgent help, ask to check the condition of the house.

Field of use. Observation, security of own property.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Огляд існуючих програмних можливостей для моніторингу підозрілої активності в житловому районі.....	10
1.2 Визначення особливостей програми, що розробляється	12
1.3 Основна ідея програми, що розробляється.....	13
1.4 Висновки за розділом 1	14
2 Проєктування програми.....	16
2.1 Вибір засобів проєктування і розробки	16
2.2 Визначення функціональних вимог до програмного забезпечення	17
2.3 Проєктування структури бази даних.....	20
2.4 Висновки за розділом 2	33
3 Розробка програми	34
3.1 Порядок функціонування програми.....	34
3.2 Реалізація функцій застосунку.....	35
3.3 Висновки за розділом 3	42
4 Експлуатація та тестування програми	43
4.1 Опис програми.....	43
4.2 Експлуатація програми.....	43
4.3 Тестування програми.....	49
4.4 Висновки за розділом 4	51
Висновки	52
Перелік джерел посилення	53
Додаток А Технічне завдання	54
Додаток Б Текст програми	59
Додаток В Слайди презентації.....	85

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Забезпечення безпеки на території місця проживання людини є важливим. При цьому необхідно розуміти, що тільки встановлення факту певної небезпеки власному майну може бути недостатньо. Сучасні камери відеоспостереження лише частково вирішуються цю проблему. Вони можуть допомогти встановити факт проникнення на певну територію, але процес після цього вже не на стільки визначений. Можна викликати поліцію, але поки поліція дістанеться місця пригоди, пройде час. Не всі випадки можуть бути вирішені через цей час. Для деяких це вже може бути запізно, швидкість реагування має бути значно більшою.

З іншого боку, наприклад, сусіди в таких ситуаціях можуть бути набагато оперативнішими. Тому підтримку сусідів або загалом людей, які живуть поруч, можна розглядати як щонайменше корисну. При цьому не всі небезпечні випадки камери відеоспостереження можуть виявити, не всюди камери в багатоквартирних будинках встановлені. А це може призвести до того, що можна визначити, що подія сталася, можна отримати деякі дані про підозрюваного, але ситуація вже сталася, відреагувати на неї не вдалось. При цьому людина, яка проходила повз, помітила щось і має певну підозру, може повідомити швидко сусідів, а інші сусіди, господарі можуть допомогти вирішити або вирішити таку проблему.

При виявленні певної проблемної ситуації варто дізнатися, які дії заплановані, запитати у оточуючих, можна попросити іншого користувача сходити перевірити стан певного приміщення, відправити сповіщення про виявлену небезпеку і необхідність допомоги [1]. За наявності цих засобів користувач може швидко повідомити інших користувачів про те, що сталось, проінформувати всіх завчасно про заплановані дії, співставити підозрілі активності з запланованими діями.

Саме перелічені вимоги і є основними завданнями або визначають ті основні завдання, які мають бути вирішені в процесі розробки програмного

забезпечення (ПЗ) в роботі. Тож метою роботи є розробити ПЗ моніторингу підозрілої активності в житловому районі для допомоги виявлення і вирішення безпекових проблем людям всередині спільноти, з представниками якої вони проживають поруч.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих програмних можливостей для моніторингу підозрілої активності в житловому районі

Даний підрозділ направлений на розгляд і аналіз тих можливостей, які вже існують, надаються в подібному ПЗ.

Однак першим результатом аналізу є те, що прямий відповідник в Україні не був знайдений. Є ПЗ, яке підтримує пошук кримінальних злочинців, надсилає сповіщення потрібним користувачам, коли трапилась проблема (наприклад, напад), але як такої програми для спільного інформування і відслідковування не було безпосередньо виявлено для українських користувачів. Загалом є лише одна програма, яка не доступна в Україні взагалі, – WeAlert [2]-[3]. Тобто фактично це ніяк не вирішує проблему, яка існує, на практиці. Тож це говорить про наявність відповідного простору для забезпечення відсутніх можливостей програмою, яка має бути розроблена в роботі.

WeAlert є застосунком, який дозволяє комунікувати користувачам всередині деякої групи між собою [4]. Доступний пошук груп, що дозволяє підключатися до певної спільноти, однак варто зазначити, що це не сприяє безпеці як такій. Тобто якщо пошук в принципі організований відкрито і якщо будь-який користувач може додатися до певної групи, то це може призводити до того, що крадії, шахраї можуть спершу ознайомитися з програмою перед вчиненням власних діянь. Відповідно програма тоді може нести певну корисну інформацію. Наприклад, може зрозуміти, хто зазвичай є джерелом інформації, як користувачі взаємодіють між собою, коли подібні випадки виявляються частіше. Тож програма є корисною, має багато функцій для підтримки користувачів, але її практичне використання може бути спотворено у випадку наявності зловмисників, які також захочуть її використовувати.

WeAlert надає можливість повідомляти всім учасникам групи про ту подію, яка сталася. Таким чином інформація швидко розповсюджується. Якщо

це пожежа, крадіжка, то звичайно, що будь-яка така інформація у випадку масового розповсюдження може мати користь. Окрім того користувачі можуть безпосередньо контактувати між собою.

При цьому загалом варто відзначити, що такий підхід до роботи з відповідною інформацією є дуже корисним, адже для спільнот, для комунікації в спільнотах можуть використовуватися і інші загальні рішення. Наприклад, месенджери. Але вони мають певну проблему. Так чат об'єднання співвласників багатоквартирних будинків містить великий обсяг інформації, який зазвичай не несе ніякої користі стосовно підозрілої активності. Відповідно, якщо щось стається, то повідомити таким чином про цю інформацію, отримавши швидкий відгук, важко. Коли є окрема програма, яка може надсилати сповіщення тільки стосовно певних проблем, то це звичайно сприяє увазі користувачів, а відповідно призначення щодо моніторингу підозрілої активності розкривається в більш повній мірі.

Також потрібно відкреслити, що загалом подібні застосунки можуть бути достатньо корисними ще й з точки зору певної соціалізації. Кримінальні випадки нерідко трапляються з більшою частотою, ніж це могло би бути, через те, що люди в деякій спільноті не соціалізовані. Тобто не налагоджені зв'язки, люди не комунікують. Тому якщо щось стається або якщо хтось побачив щось, що може викликати підозру, він може пройти повз і можливо навіть не мати можливості повідомити іншим. Таке ПЗ сприяє соціалізації тільки в частині моніторингу підозрілої активності. Тобто якщо людина не хоче отримувати від інших якусь іншу інформацію, читати про їх погляди на життя, то враховуючи, що це не загальний чат деякої спільноти, вона фактично такої інформації отримувати не буде.

MyLocalCrime [5] є ще одним застосунком, який стосується цієї сфери. Він фактично є представником цілої групи таких програм. Однак цей застосунок має абсолютно іншу концепцію. Він передбачає інформування саме про кримінальні ситуації [6]-[7]. Це дозволяє його використовувати як певний довідник стосовно того району, куди наприклад, переміщується

користувач. Звичайно, що корисним може бути і дізнатися про свій власний район, про ті кримінальні ситуації, які в ньому трапляються. Але це фактично вже довідник того, що зафіксувала поліція. Застосунок має достатньо велику потенційну користь для людей, але фактично стосовно статистики того, що сталося. Це підкреслює навіть підхід до інформування про нові події, який в основі використовує інформування на основі електронних листів. Тобто це фактично певні звіти, а не сповіщення про те, що сталося щось термінове.

Тож наявні програми мають достатньо різні підходи, при цьому кожен з них може бути корисним для деякого роду завдань. Проте в роботі в подальшому розглядається логіка, яка базується саме на комунікації людей в житловому районі щодо виявлення, інформування про підозрілу активність або фактичні кримінальні випадки, які спостерігались.

1.2 Визначення особливостей програми, що розробляється

Виділені під час аналізу аналоги були порівняні з розробкою (табл. 1.1).

Таблиця 1.1 – Порівняння розробки з прототипом

Характеристика	MyLocalCrime	WeAlert	Розробка
Можливість зв'язатися з сусідами напряму	Ні	Так	Так
Сповіщення користувачів про ситуацію, яка сталась	Так	Так	Так
Визначення статистики кримінальних випадків, які вже трапилися	Так	Ні	Так
Підтримка і визначення близьких людей	Ні	Ні	Так
Взаємодія муж створеними групами	Ні	Ні	Так

Розробка наслідує ідею MyLocalCrime щодо визначення статистики кримінальних випадків, які вже трапилися. Це може бути корисним для розуміння стану навколишнього середовища в цьому ракурсі. Програма використовує спільну логіку з WeAlert стосовно можливості зв'язатися з сусідами напряду, сповіщення користувачів про ситуацію, яка сталась, але має і свої власні відмінності. Такими відмінностями є підтримка і визначення близьких людей, тобто можливість повідомити про критичну ситуацію одразу і родину або родичів. Це може бути корисним і тоді, коли користувач поїхав у відпустку, а друг або родич допомагає цьому користувачу стосовно догляду, спостереження за будинком. Також програма повинна дозволяти взаємодіяти між створеними групами, а не тільки всередині них. Це фактично разом з особливим підходом щодо включення користувачів до груп, який має базуватися на спільному підтвердженні цієї можливості, відрізняє програму, що розробляється.

1.3 Основна ідея програми, що розробляється

Виходячи з визначених особливостей програми, слід сформулювати основну логіку роботи в програмі.

По-перше усі об'єднання користувачів розділені на три складові:

- сусідства – це об'єднання найнижчого рівня, фактично це може бути об'єднання співвласників багатоквартирного будинку;
- райони – це об'єднання сусідств, коли декілька сусідств хочуть взаємодіяти стосовно передачі інформації про підозрілу активність;
- близькі люди – це користувачі, які можуть або допомагати своїм близьким людям щодо відслідковування активностей, коли людина знаходиться не вдома (у відпустці, наприклад), або ті, які мають отримати сповіщення про критичні ситуації, в які потрапила їх близька людина, коли потребує допомоги.

Райони необхідні зокрема і для того, щоб відділити взаємодію між користувачами. Тобто користувачі на рівні сусідства можуть, наприклад, разом спілкуватися щодо загальних проблем безпеки, надсилаючи повідомлення. Користувачі на рівні району вже таких повідомлень надсилати не будуть, їм важливо отримувати інформацію про ситуації, в яких є підозра на кримінальні випадки, високий рівень небезпеки. Якщо ж це просто певне спостереження, якому ніхто не приділяє увагу, тобто це низький рівень серйозності, то тоді така інформація не є корисною для користувача з іншого сусідства, але з одного району. Окрім того автор може сам одразу визначати, чи варта ця інформація, щоб про неї знали інші представники з району.

Щодо початку роботи з сусідством використовується принцип, що для цього заявка має бути підтверджена всіма менеджерами. Тобто в кожному сусідстві є звичайні користувачі та менеджери. Усі менеджери мають схвалити участь користувача у сусідстві, щоб він почав отримувати інформацію.

1.4 Висновки за розділом 1

У першому розділі проаналізовано ПЗ, що фактично є придатним для певних відстежень підозрілої активності. Фактично в ролі прототипа було обрано застосунок WeAlert, основною проблемою якого є недоступність в Україні. Окрім того було запропоновано додаткові можливості для програми, а також визначено основні ідеї програми, що розробляється, зокрема логіку роботи зі спільнотами різного рівня в програмі, що включають райони і сусідства.

Для виконання у роботі виділений наступний набір завдань:

- аналіз предметної області моніторингу підозрілої активності в житловому районі;
- визначення вимог і особливостей програмного забезпечення моніторингу підозрілої активності в житловому районі;

- проєктування програмного забезпечення моніторингу підозрілої активності в житловому районі;
- реалізація ПЗ моніторингу підозрілої активності в житловому районі і тестування.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Python [8] на даний момент є однією з мов програмування, які активно використовуються для веброзробки, проте при цьому його головною відмінністю є простота і швидкість роботи: він дозволяє створювати перші версії програми дуже швидко за встановлення відповідного фреймворку. Тоді наприклад, за наявності у фреймворку можливостей адміністративної панелі робота з даними може бути розпочата одразу. Відповідно під час порівняння фреймворків для мови Python (табл. 2.1), Django [9] і Tornado, це було враховано як один з критеріїв. Засоби адміністративної панелі є тільки у Django, а також і краща підтримка, кращий контроль безпеки при дещо гіршій продуктивності, хоча для задача на практиці цієї продуктивності вистачає, що підтверджено використанням у багатокористувацьких застосунках з максимальними обсягами популярності у світі на даний момент.

Таблиця 2.1 – Порівняння вебфреймворків для розробки

Показник	Django	Tornado
Механізм керування даними адміністратором	Наявний	Відсутній
Підтримка	Краща	Гірша
Контроль безпеки	Кращий	Середній
Продуктивність	Середня	Відносно краща
Засоби аутентифікації за замовчуванням	Є	Є
Рушій для роботи з шаблонами	Є	Є

2.2 Визначення функціональних вимог до програмного забезпечення

Користувачі програми не мають розбиття на рівні відповідальності. Кожному користувачу доступна кожна функція в програмі. Проте вже стосовно конкретних об'єктів у користувачів виникають певні права. У кожного сусідства є менеджери і звичайні користувачі. Менеджери відповідальні за підтвердження або скасування прав інших користувачів у сусідстві. Загальний перелік функціональних вимог містить такі функції:

- авторизація користувачів;
- реєстрація;
- редагування персональних даних користувача;
- перегляд даних житлового району;
- перегляд даних сусідства;
- перегляд повної інформації підозрілої активності;
- перегляд власних записів про виявлені підозрілі активності;
- перегляд сусідств за районом;
- внесення власного місця проживання;
- редагування даних власного місця проживання;
- перегляд всіх власних місць проживання;
- створення сусідства;
- редагування даних сусідства;
- деактивація сусідства;
- додавання користувача до сусідства;
- вилучення користувача з сусідства;
- підтвердження рівня доступу користувача у сусідстві;
- створення району;
- редагування даних району;
- додавання сусідства до району;
- вилучення сусідства з району;

- підтвердження рівня доступу сусідства у районі;
- перегляд сусідств користувача;
- перегляд поточного стану активностей;
- перегляд історії дій користувача;
- пошук користувачів;
- інформування про виявлену підозрілу активність;
- редагування даних підозрілої активності;
- визначення версії стосовно підозрілої активності;
- голосування за версію стосовно підозрілої активності;
- перегляд повідомлень у спілкуванні за сусідством користувача;
- перегляд переліку спілкувань;
- відправлення повідомлення у спілкування;
- перегляд списку користувачів, з якими були прямі повідомлення;
- відправлення повідомлення користувачу;
- визначення інформації про заплановані дії;
- відправлення сповіщення про необхідність термінової допомоги;
- сповіщення групи користувачів про критичну активність;
- персональне сповіщення користувача;
- перегляд відправлених сповіщень користувачем;
- перегляд історії отриманих сповіщень користувачем;
- визначення близької людини;
- перегляд власних близьких людей;
- активація відповідальності близької людини;
- деактивація відповідальності близької людини;
- перегляд статистики виявлення підозрілих активностей за категоріями;
- встановлення серйозності підозрілої активності;
- перегляд власних версій за виявленими підозрілими активностями

(рис. 2.1).

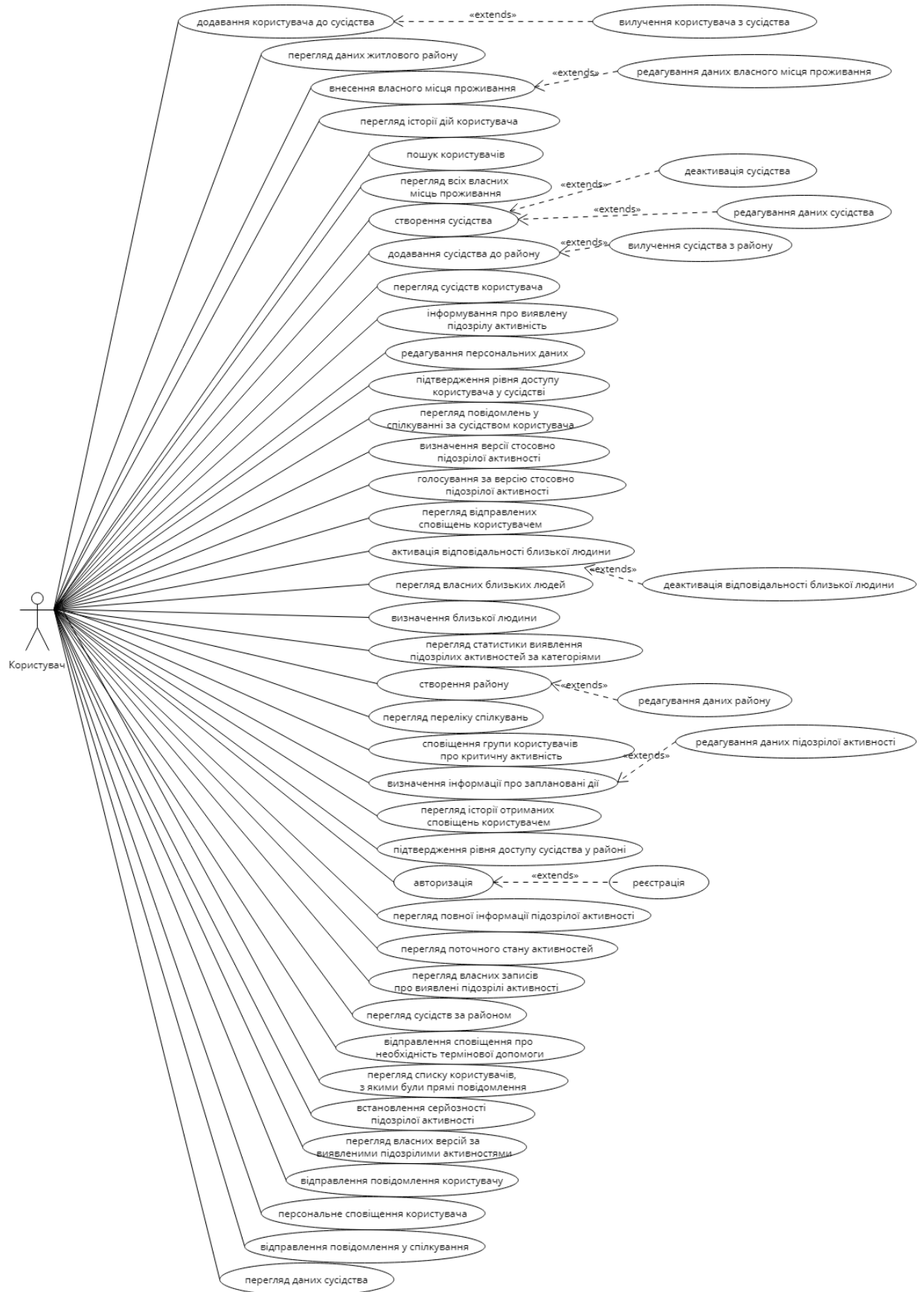


Рисунок 2.1 – Діаграма прецедентів

2.3 Проектування структури бази даних

Базу даних (БД) було створено засоби системи MySQL. Саму структуру БД описано нижче.

Будь-яка людина, яка працює з програмою є її користувачем і обов'язково передбачає наявність запису в БД про цього користувача, адже якщо такого запису немає, то ніякої інформації такій людині надавати не можна, адже це спотворить принцип створення такого застосунку. Тобто якщо злодій буде мати доступ до інформації, якою між собою діляться жителі деякого будинку, то це звичайно є проблемою і цінність такої інформації практично втрачається.

Користувачі програми визначаються записами в таблиці User, що містить сукупність характеристик користувача за полем унікального ідентифікатора користувача id, логіном, що використовується для авторизації, – username, інформацією користувача про себе – bio, за повним іменем людини (персональні прізвище та ім'я) fullname, паролем password, фотографією photo, активністю доступу до програми active.

Користувачі самі по собі важливі, але для функціонування програми не менш важливою характеристикою їх є місце їхнього проживання. Хоча це може бути і не тільки проживання звичайно. Потенційно, коли людина багато років працює на одному місці, якщо це офіс в житловому будинку, то користувач може стати учасником деякої спільноти і на роботі. Тож в загальному випадку виділено сутність проживання, але вона може бути конкретизована і не тільки фактом проживання, а тривалого перебування в якомусь місці, де інформація від користувача може розглядатися як достатньо надійна, надання йому інформації не суперечить правилам безпеки поводження з інформацією, а його послуги можуть бути корисними під час виявлення підозрілої активності.

Постійне проживання користувача в конкретному місці фіксується в БД у таблиці Living, що містить сукупність характеристик проживання за полями:

- id – унікальний ідентифікатор постійного проживання користувача;
- name – коротка назва цього проживання для зручності сприйняття користувачем під час роботи з програмою;
- user_id – користувач, для якого визначено це проживання;
- longitude – довгота, якою визначена географічна координата місця проживання;
- latitude – широта, якою визначена географічна координата місця проживання;
- textaddress – повна адреса проживання, зазначена текстом (з визначенням міста, вулиці, номеру будинку).

Поле user_id у таблиці Living є зв'язуючим з таблицею User, виконуючи роль зовнішнього ключа. Такий зв'язок необхідний через те, що користувач може в різні моменти проживати в різних місцях, тобто з часом фактично переїхати. Звичайно, що в такому випадку дані можна було б і вилучити, але є друга ситуація – коли людина має декілька квартир і часом проживає в декількох. Відповідно для неї важливе інформування про підозрілу активність і щодо тієї квартири або території, де вона проживає в поточний момент, і щодо тієї квартири і території, де вона в поточний момент не проживає, але де з'явиться через якийсь проміжок, який може бути в тому числі дуже коротким. Тож для такого варіанта виходить, що один користувач може мати багато проживань, але при цьому кожне проживання визначається стосовно одного конкретного користувача, тому для реалізації такого зв'язку було введено зовнішній ключ в таблицю Living.

Усі користувачі об'єднуються в певні сусідства. Фактично це може бути те саме об'єднання співвласників багатоквартирного будинку, в якому користувачі бажають домовитися про допомогу один одному у випадку

підозрілої активності. Сусідства знаходяться в БД у таблиці Neighborhood, що містить сукупність характеристик сусідства за полями:

- id – унікальний ідентифікатор сусідства;
- name – назва сусідства, що використовується для швидкого розуміння;
- bio – коротка інформація стосовно цього сусідства: тут може вказуватися певна підказка;
- photo – фотографія, яка ілюструє це сусідство;
- longitude – довгота, якою визначена географічна координата місця розташування сусідства (наприклад, будинку);
- latitude – широта, якою визначена географічна координата місця розташування сусідства;
- textaddress – повна адреса розташування сусідства, зазначена текстом (з визначенням міста, вулиці, номеру будинку);
- confidant – визначає, що з сусідством може працювати стосовно виявлених підозрілих активностей і близька людина користувача, який входить до цього сусідства, якщо він відсутній в даний момент і визначив замість себе помічника;
- active – визначає, чи працює це сусідство, чи вже вилучено фактично (деактивовано).

За кожним проживанням як базове має бути визначено деяке сусідство, але фактично може бути таке, що через деякі внутрішні проблеми люди можуть бути роз'єднані, відповідно створюючи окремі сусідства за однаковою адресою. Тоді може виникнути ситуація, коли один користувач за своїм проживанням буде віднесений до декількох сусідств. При цьому і сусідство містить зазвичай не одного користувача, не одне його проживання, а проживання декількох різних користувачів, тож виходить що багато сусідств співвідносяться з багатьма проживаннями, тому необхідно було створити таблицю NeighborhoodLiving для зв'язування проживань з сусідствами.

Таблиця NeighborhoodLiving містить сукупність характеристик проживань відносно сусідства за полями:

- id – унікальний ідентифікатор входження проживання у певне сусідство;
- neighborhood_id – сусідство, у яке входить дане проживання (зовнішній ключ з таблиці Neighborhood);
- living_id – проживання, яке входить до даного сусідства (зовнішній ключ з таблиці Living);
- manager_level – права, з якими входить користувач за цим проживанням у це сусідство: менеджер або звичайний житель;
- approved – підтвердження всіма менеджерами сусідства входження користувача за цим проживанням до цього сусідства.

Безпосередньо загальне підтвердження в полі approved формується на основі окремих підтверджень, якими кожен менеджер, кожен користувач у сусідстві підтверджує доступ цього користувача до сусідства. Ці окремі підтвердження знаходяться в таблиці NeighborhoodLivingApprove, що містить сукупність характеристик підтвердження входження проживання в сусідство за полями:

- id – унікальний ідентифікатор підтвердження входження проживання в сусідство;
- nbliving_src_id – проживання користувача стосовно певного сусідства, яке підтверджується;
- nbliving_dst_id – проживання користувача стосовно певного сусідства, використовуючи яке було підтверджено це проживання іншого користувача;
- operation_date – час підтвердження цього проживання користувача у сусідстві;
- level – рівень відповідності.

Рівень відповідності може мати наступні значення:

- значення 0 вказує на невідповідність (тобто скасування фактично);

- значення 1 вказує на можливість отримання тільки підтверджених загальних сповіщень;
- значення 2 вказує на можливість отримання всіх загальних сповіщень від цього користувача (проживання якого підтверджується);
- значення 3 вказує на можливість отримання в тому числі персональних сповіщень.

Інформація про підозрілу активність всередині одного сусідства є важливою, але не всю потрібну інформацію так можна отримати. Наприклад, щось можуть виконувати сусіди, представники іншого сусідства (іншого об'єднання співвласників багатоквартирних будинків), а отримання цієї інформації було б важливим для сусідів. Відповідно таке фрагментування виникає в такому випадку. Щоб уникнути цієї проблеми, потрібно об'єднувати сусідства між собою. Для цього використовується сутність району, яка представлена таблицею Region, що містить сукупність характеристик району за полями: унікальний ідентифікатор району – поле id, назва району – поле name.

Безпосередньо входження сусідств до районів знаходиться в таблиці RegionNeighborhood, що містить сукупність характеристик сусідства в районі за полями:

- id – унікальний ідентифікатор входження сусідства до району;
- region_id – район, який входить до цього сусідства (зовнішній ключ з таблиці Region);
- neighborhood_id – сусідство, яке входить в цей район (зовнішній ключ з таблиці Neighborhood);
- approved – підтвердження цього входження сусідства до району.

Кожен район може містити різні входження до нього сусідства, але кожне входження є входженням в один конкретний район. Кожне сусідство може мати входження в різні райони, але кожне входження стосується одного суспільства.

Отримання інформації в межах району також підіймає ті самі проблеми стосовно рівнів доступу, що і під час окремого сусідства. Тому надання доступу певному сусідству до даних іншого сусідства реалізується також через підтвердження. Вони знаходяться в таблиці RegionNeighborhoodApprove, що містить сукупність характеристик підтвердження доступу сусідства до району за полями:

- id – ідентифікатор підтвердження доступу сусідства до району;
- regionneighborhood_id – місце сусідства в певному районі (зовнішній ключ з таблиці RegionNeighborhood);
- user_id – користувач, який підтвердив входження сусідства в район (зовнішній ключ з таблиці User).

Категорії активностей дозволяють одразу користувачу зрозуміти, наприклад, про що саме в сповіщенні йде мова. Вони знаходяться в таблиці ActivityCategory, що містить сукупність характеристик категорій активностей за полями: унікальний ідентифікатор категорії активностей – поле id, назва категорії активностей – поле name.

Виявлена підозріла активність знаходиться в таблиці Suspiciousactivity, що містить сукупність характеристик підозрілої активності за полями:

- id – унікальний ідентифікатор виявленої підозрілої активності;
- title – коротка назва підозрілої активності;
- observation_date – час, коли розпочалося спостереження підозрілої активності;
- observation_long – тривалість цієї підозрілої активності (значення може бути пропущено, якщо це одномоментна активність);
- description – опис підозрілої діяльності;
- nblliving_id – підтвердження входження проживання в сусідство, яким скористався користувач, щоб визначити підозрілу активність (зовнішній ключ з таблиці NeighborhoodLiving);
- longitude – довгота, якою визначена географічна координата місця підозрілої активності (зафіксована за місцем, коли було виконано збереження,

встановлена вручну координата або загальна координата сусідства, вказана у ньому);

- latitude – широта, якою визначена географічна координата місця підозрілої активності;

- place – текстовий опис місця підозрілої активності (наприклад, під'їзд, поверх, де це відбулось);

- activitycategory_id – категорія активності, до якої віднесено цю підозрілу активність;

- severity – ступінь серйозності виявленої діяльності.

Кожна категорія активності може мати різні виявлені підозрілі активності, а кожна підозріла активність співвідноситься тільки з однією категорією активностей.

Кожен користувач може вносити декілька різних записів про підозрілу активність, але створити кожен запис може тільки один користувач.

За кожною підозрілою активністю може встановлюватися ступінь серйозності. Для цього користувачі висловлюють свою думку. Ці думки знаходяться в таблиці SuspiciousactivitySeverity, що містить сукупність характеристик серйозності підозрілої активності за полями:

- id – унікальний ідентифікатор думки про серйозність підозрілої активності;

- suspiciousactivity_id – підозріла активність, якої стосується ця думка (зовнішній ключ з таблиці Suspiciousactivity);

- user_id – користувач, який висловив цю думку про підозрілу активність (зовнішній ключ з таблиці User);

- severity – ступінь серйозності підозрілої активності.

Фотографія, співвіднесена з підозрілою активністю, знаходиться в таблиці SuspiciousactivityPhoto, що містить сукупність характеристик фотографії за полями:

- id – унікальний ідентифікатор фотографії підозрілої активності;

- suspiciousactivity_id – підозріла активність, до якої відноситься ця фотографія (зовнішній ключ з таблиці Suspiciousactivity);
- photo – сама фотографія.

Кожна підозріла активність може мати декілька фотографій для ілюстрації її, при цьому кожна фотографія може бути використана тільки відносно однієї підозрілої активності.

На виявлену підозрілу активність інші користувачі можуть висловлювати свої версії, надавати інформацію про те, що могло статися. Версії користувачів стосовно виявленої підозрілої активності знаходяться в таблиці Version, що містить сукупність характеристик версії стосовно підозрілої активності за полями:

- id – унікальний ідентифікатор версії стосовно виявленої підозрілої активності;
- suspiciousactivity_id – підозріла активність, до якої відноситься ця висловлена версія (зовнішній ключ з таблиці Suspiciousactivity);
- version_text – текст висловленої версії;
- created_date – час створення цієї версії у програмі;
- living_id – проживання користувача, яким він скористався, щоб створити цю версію в програмі (зовнішній ключ з таблиці Living).

Кожен користувач може створювати різні версії стосовно різних підозрілих активностей, але кожна версія має тільки одного автора.

Кожна підозріла активність може мати декілька висловлених версій, а кожна версія може стосуватися тільки однієї активності.

Користувачі можуть голосувати або іншими словами підтримувати висловлені версії. Ці голоси знаходяться в таблиці VersionVote, що містить сукупність характеристик голосів, відданих за версії, за полями:

- id – унікальний ідентифікатор відданого голосу за певну версію стосовно виявленої підозрілої активності;
- user_id – користувач, який висловив цю версію (зовнішній ключ з таблиці User);

– version_id – версія стосовно виявленої активності (зовнішній ключ з таблиці Version).

До висловлених версій також можуть додаватися фотографії. Ці фотографії знаходяться в таблиці VersionPhoto, що містить сукупність характеристик фотографій версій за полями:

– id – унікальний ідентифікатор фотографії висловленої версії стосовно підозрілої активності;

– version_id – версія стосовно виявленої активності (зовнішній ключ з таблиці Version), до якої відноситься фотографія;

– photo – сама фотографія.

Кожна версія про підозрілу активність може мати декілька фотографій для ілюстрації її, при цьому кожна фотографія може бути використана тільки відносно однієї версії про підозрілу активність.

Спілкування може відбуватися у всьому сусідстві, щоб наприклад, в реальному часі проводити обговорення. Такі спілкування знаходяться в таблиці Conversation, що містить сукупність характеристик спілкування за полями:

– id – унікальний ідентифікатор спілкування;

– neighborhood_id – сусідство, до якого належить це спілкування (зовнішній ключ з таблиці Neighborhood);

– user_id – користувач, який написав повідомлення у спілкуванні (зовнішній ключ з таблиці User);

– created_date – час створення повідомлення у спілкуванні;

– text – текст повідомлення.

Кожне сусідство може мати багато спілкувань (повідомлень), а кожне повідомлення відноситься тільки до одного сусідства одночасно.

Кожен користувач може створювати різні повідомлення в спілкуваннях всередині сусідства, але кожне повідомлення має тільки одного автора.

До повідомлення у спілкуванні може додаватися фотографія. Вона знаходиться в таблиці ConversationPhoto, що містить сукупність характеристик фотографій повідомлень у спілкуваннях за полями:

- id – унікальний ідентифікатор фотографії у спілкуванні;
- conversation_id – спілкування, до якого віднесено цю фотографію (зовнішній ключ з таблиці Conversation);
- photo – сама фотографія.

Кожне спілкування може мати багато фотографій, а кожна фотографія може відноситись тільки до одного спілкування.

Прямі повідомлення, тобто повідомлення між окремими користувачами напрямку, знаходяться в таблиці Directmessage, що містить сукупність характеристик швидких повідомлень за полями:

- id – ідентифікатор прямого повідомлення між користувачами;
- user_src_id – користувач, який створив повідомлення (зовнішній ключ з таблиці User);
- user_dst_id – користувач, якому адресовано повідомлення (зовнішній ключ з таблиці User);
- text – текст повідомлення;
- created_date – час створення повідомлення;
- suspiciousactivity_id – підозріла активність, на яку акцентовано увагу в повідомленні (зовнішній ключ з таблиці Suspiciousactivity).

Кожна підозріла активність може мати багато швидких повідомлень у міжособистісному спілкуванні, але кожне повідомлення може співставлятися тільки з однією підозрілою активністю.

До прямого повідомлення може додаватися фотографія. Вона знаходиться в таблиці DirectmessagePhoto, що містить сукупність характеристик фотографій прямих повідомлень за полями:

- id – унікальний ідентифікатор фотографії прямого повідомлення;
- directmessage_id – пряме повідомлення, до якого віднесено цю фотографію (зовнішній ключ з таблиці Directmessage);

– photo – сама фотографія.

Кожне пряме повідомлення може мати багато фотографій, а кожна фотографія може відноситись тільки до одного прямого спілкування.

Окрім пошуку інформації за фактом того, що сталося, користувачам може перед тим надаватися інформація про заплановані дії. Наприклад, роботи, які заплановано проводити. Тоді в момент проведення працівники можуть здаватися підозрілими, розташовуючись, наприклад, у під'їзді. Ось щоб повідомити раніше про заплановані роботи можуть створюватися певні звістки, які в БД знаходяться в таблиці Report, що містить сукупність характеристик звістки за полями:

- id – ідентифікатор звістки про заплановані роботи;
- living_id – проживання користувача, за яким створено звістку (зовнішній ключ з таблиці Living);
- title – назва звістки;
- description – опис суті звістки, тобто наприклад, які саме роботи заплановані;
- place – місце, де заплановано роботи;
- created_date – час створення звістки;
- event_date – час початку події, звістка про яку представлена;
- event_long – тривалість події;
- activitycategory_id – категорія активності, до якої в підсумку може відноситися ця активність, якщо вона буде розглядатися як підозріла (зовнішній ключ з таблиці ActivityCategory);
- longitude – довгота, якою визначена географічна координата місця, де будуть проводитися події (роботи) за звісткою;
- latitude – широта, якою визначена географічна координата місця, де будуть проводитися події (роботи) за звісткою.

Кожна категорія активності може мати декілька звісток, а одна звістка може мати тільки одну категорію.

За проживанням користувача може в різний час створюватися декілька звісток, але кожна звістка може визначатися тільки одним проживанням.

Сповіщення можуть направлятися не тільки індивідуально, але і на певній відстані, щоб усіх учасників спільноти, до якої належить за своїм проживанням користувач, а також всіх учасників спільнот з одного й того самого району повідомити про певну проблему. Дані таких сповіщень знаходяться в таблиці Notification, що містить сукупність характеристик сповіщень за полями:

- id – унікальний ідентифікатор сповіщення;
- living_id – проживання користувача, за яким створено сповіщення (зовнішній ключ з таблиці Living);
- created_date – час створення сповіщення;
- activitycategory_id – категорія активності, щодо якої інформуються цим сповіщенням користувачі (зовнішній ключ з таблиці ActivityCategory);
- text – текст сповіщення;
- level – рівень відповідності сповіщення: всім у сусідстві або всім у всіх сусідствах з районів, до яких входить сусідство, окремий рівень – сповіщення про необхідність термінової допомоги.

Кожне проживання користувача може визначати декілька сповіщень, при цьому кожне сповіщення стосується тільки одного проживання користувача.

Кожна категорія активності може бути застосована в різних сповіщеннях, проте кожне сповіщення може стосуватися тільки однієї категорії активності.

Персональні сповіщення відповідно направлені від одного користувача іншому. Вони знаходяться в таблиці БД PersonalNotification, що містить сукупність характеристик персонального сповіщення за полями:

- id – унікальний ідентифікатор персонального сповіщення;
- user_src_id – користувач, який створив персональне сповіщення (зовнішній ключ з таблиці User);

- user_dst_id – користувач, якому направлено персональне сповіщення (зовнішній ключ з таблиці User);
- text – текст персонального сповіщення;
- created_date – час створення сповіщення (рис. 2.2).

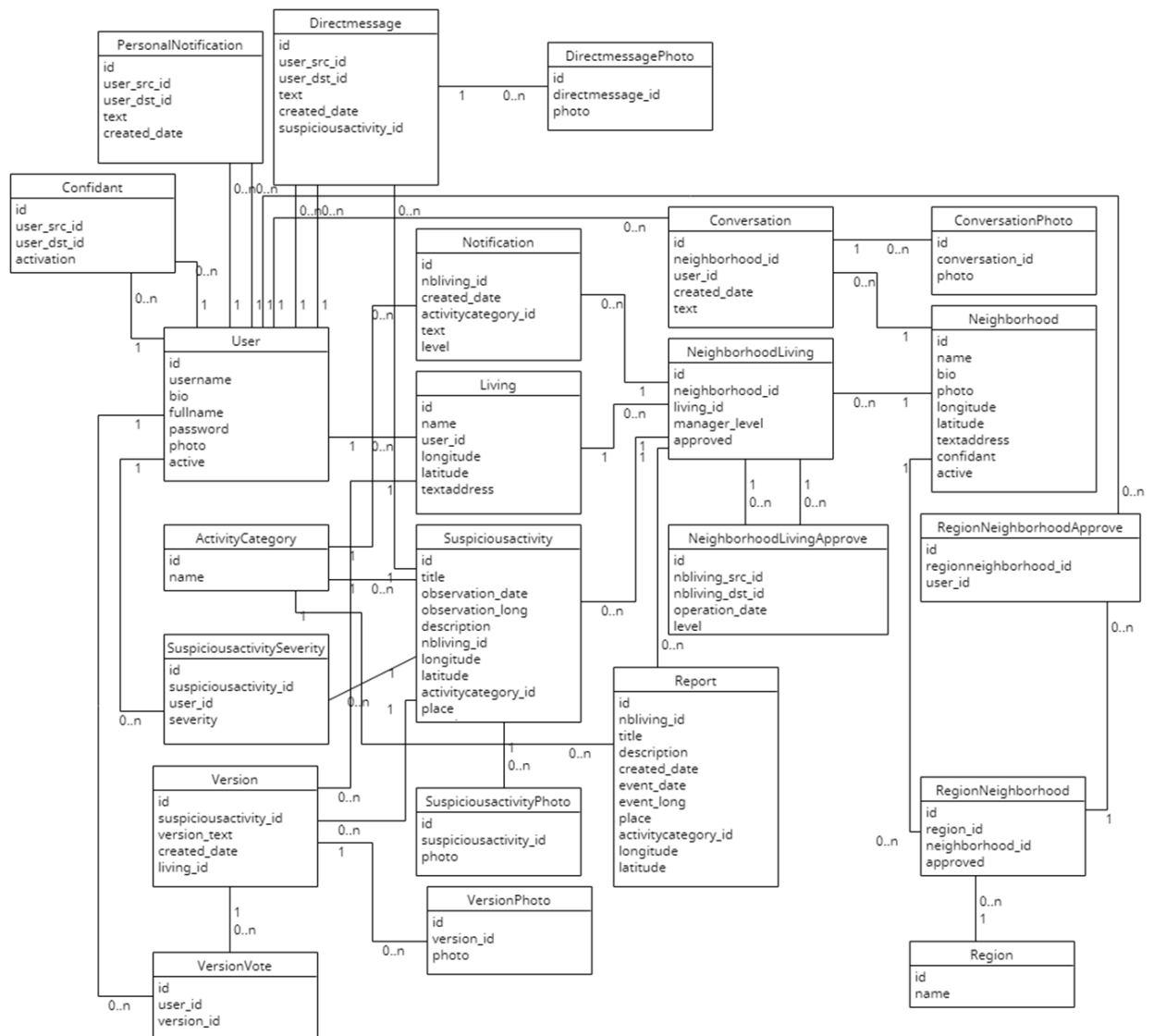


Рисунок 2.2 – Схема БД

Кожен користувач може створювати різні персональні сповіщення, при цьому у конкретного сповіщення може бути тільки один автор. З іншого боку одне персональне сповіщення може бути адресовано тільки одному користувачу, хоча кожен користувач може мати різні персональні сповіщення за час роботи з програмою.

Користувач може визначати у програмі своїх близьких людей. Ця інформація необхідна, щоб у разі необхідності допомоги повідомити близьку людину, де б вона не проживала, до якого б сусідства не входила. Окрім того це необхідно, щоб у випадку, якщо користувач поїхав кудись, і потрібно слідкувати за його будинком, отримувати певну інформацію про підозрілу активність.

Дані про близьких людей знаходяться в таблиці Confidant, що містить сукупність характеристик близьких людей за полями:

- id – унікальний ідентифікатор запису про близьку людину;
- user_src_id – користувач, який визначив свою близьку людину (зовнішній ключ з таблиці User);
- user_dst_id – користувач, який є близькою людиною цього користувача (зовнішній ключ з таблиці User);
- activation – стан активації заміни користувача на близьку людину для контролю і догляду за будинком.

Кожен користувач може мати багато записів про своїх близьких людей, але кожен запис стосується тільки одного автора. Кожен користувач може бути близькою людиною для різних користувачів, але кожен запис стосується тільки одного користувача як близької людини.

2.4 Висновки за розділом 2

Реалізовано вибір технологій, що є засобами розробки, включаючи мову програмування Python, вебфреймворк Django. За визначеними функціональними вимогами виконано моделювання сценаріїв використання програми користувачами. Основними сутностями БД, проєктування якої виконано в цьому розділі, є користувач, проживання, сусідства, райони, підозріла активність, висловлені версії про підозрілу активність, звістки про заплановані події (роботи), масові та індивідуальні сповіщення. Підтримку роботи з БД виконано через MySQL.

3 РОЗРОБКА ПРОГРАМИ

3.1 Порядок функціонування програми

Як було визначено в першому розділі, всі дії мають враховувати рівень доступу. Відповідно однією з особливих дій є підтримка підозрілої активності (рис. 3.1).

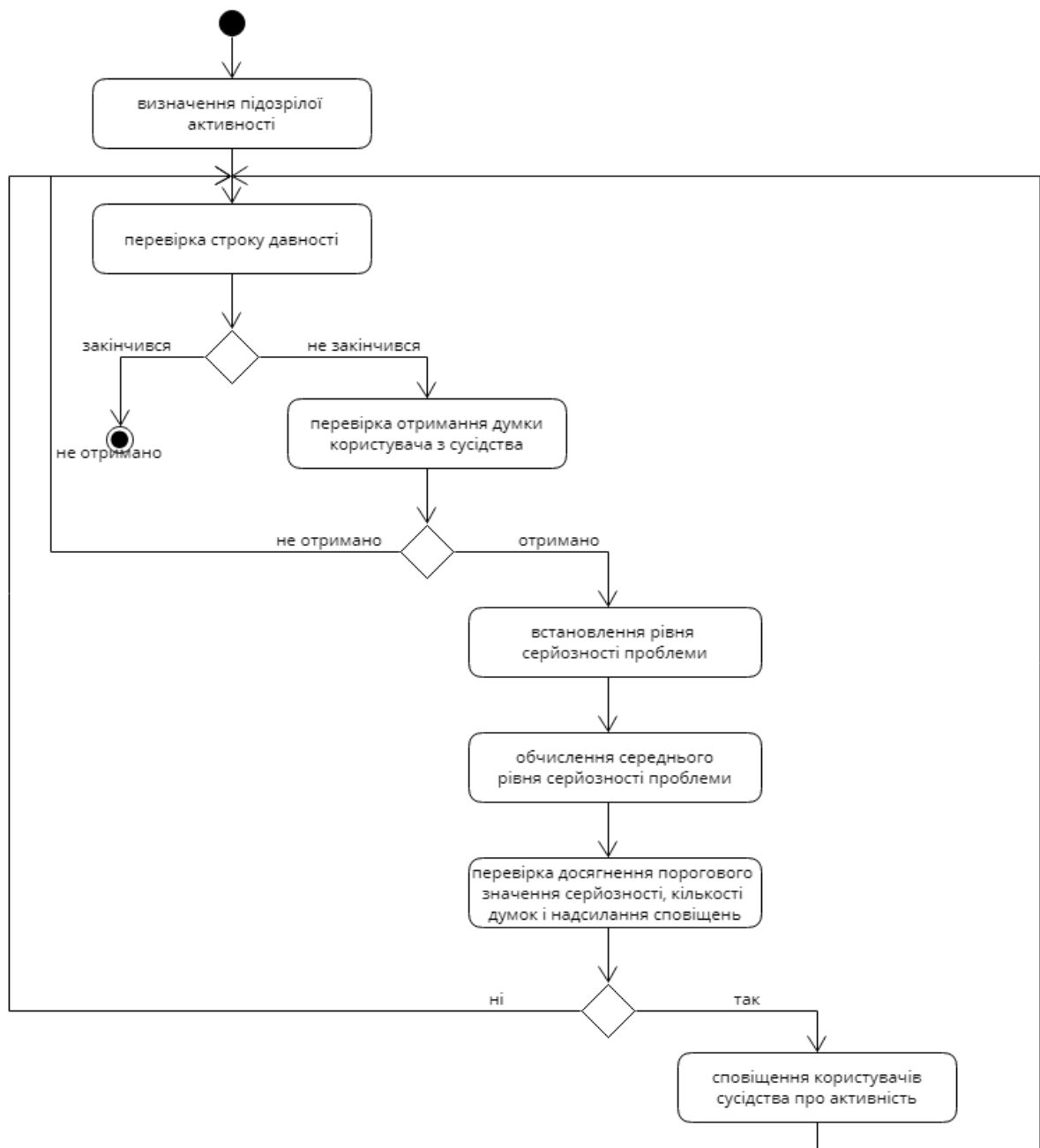


Рисунок 3.1 – Діаграма діяльності підтримки сповіщень підозрілої активності

Вона пов'язана з тим, що кожен користувач може напряму відправити сповіщення іншому користувачу, людям з сусідства, але цим правом звичайно потрібно користатися обмежено. При цьому коли користувач визначив сам підозрілу активність, далі ланцюжок виконується автоматично. Користувач встановлює рівень серйозності проблеми, тобто виявленій активності, але далі інші користувачі мають висловити свою думку, щоб досягти автоматичного відправлення сповіщень. Для цього ще хоча би один користувач має встановити рівень серйозності проблеми. Якщо обчислений середній рівень в якийсь момент виявляється вище порогового значення, то розсилається сповіщення всім користувачам з того самого сусідства.

3.2 Реалізація функцій застосунку

Реалізацію застосунку виконано за допомогою фреймворку Django, використовуючи змішаний підхід. Так моделі застосунку реалізовані за допомогою класів. Представлення створені як функції. Другий варіант обумовлений тим, що він більш гнучкий, передбачає у випадку Django можливість використання декількох моделей одночасно.

Реалізацію функції перегляду власних записів про виявлені підозрілі активності забезпечено створенням функції представлення `my_suspiciousactivities_view`. У ній реалізується впорядкування активностей, які створив користувач, який визначається авторизованим акаунтом.

Реалізацію функції перегляду сусідств за районом забезпечено створенням функції представлення `region_neighborhoods_view`, що має параметр `region`, який встановлює номер району, формуючи набір результатів тільки з тих сусідств, які віднесені до цього району та при цьому це внесення активоване, тобто це означає, що кожний менеджер усіх сусідств, які вже внесені в цей район, підтвердив доступ цього сусідства до району.

Деактивація сусідства фактично замінює вилучення даних, щоб не порушити те, що вже було створено. Реалізовано деактивацію через внесення

відповідного значення в полі `active`. Дозвіл надається тільки в тому випадку, якщо користувач є менеджером цього сусідства.

Пошук користувачів забезпечується функцією представлення `user_search_view`, що має параметри `name` і `login`, які відповідно визначають ім'я та логін користувача. Перший перевіряється також на можливість некоректного порядку, тобто визначається і порядок, коли спочатку визначається ім'я, а потім прізвище або навпаки.

Перегляд сусідств користувача реалізується функцією представлення `user_neighborhoods_view`. Користувач визначається даними авторизації, відбираються всі сусідства, в яких він є учасником, а його участь підтверджена.

Перегляд відправлених сповіщень користувачем забезпечений функцією представлення `user_notifications_view`. Формуються два переліки з результатами: `groupnotifications` містить сповіщення, які були створені і направлені одразу всій групі (учасникам сусідства), `personalnotifications` містить персональні сповіщення. Потім ці два переліки об'єднуються в один, впорядковуючи їх за часом надсилання сповіщення.

Перегляд історії отриманих сповіщень користувачем реалізований функцією `to_user_notifications_view`. Вона також формує єдиний перелік, включаючи і персональні, і групові сповіщення, але вже зі сповіщень, які були адресовані авторизованому користувачу.

Перегляд списку користувачів, з якими були прямі повідомлення, підтримується функцією представлення `direct_users_view`. Спочатку визначаються всі користувачі, яким писав та які писали авторизованому користувачу. Після цього формується загальний перелік з цих двох переліків, при цьому впорядковуючи їх за часом останнього повідомлення.

Перегляд всіх власних місць проживання реалізований функцією `user_livings_view`, яка визначає всі проживання, які стосуються авторизованого користувача.

Перегляд власних близьких людей забезпечений функцією `user_confidants_view`. У ній відбираються з моделі `Confidant` усі користувачі, у яких в полі `user_src` вказаний авторизований користувач.

Активація відповідальності близької людини виконується функцією `confidant_activate_view`, у параметрі якої `confidant` визначається запис про цю близьку людину, який має активуватися. Після виконання цієї дії зі збереженням у БД цей користувач буде бачити на базовій сторінці ті підозрілі активності, заплановані дії, сповіщення, які бачать усі інші користувачі, які відносяться до цього сусідства.

Зворотна дія, деактивація відповідальності близької людини, реалізована функцією `confidant_deactivate_view`, у параметрі якої `confidant` визначається запис про цю близьку людину, який має деактивуватися. У підсумку значення поля змінюється на протилежне, у БД зберігаються внесені зміни, користувач перестає бачити на базовій сторінці відповідні активності.

Внесення власного місця проживання реалізовано функцією `add_living_view`, що підтримує роботу з формою за допомогою створення класу `LivingForm`, виконуючи перевірку і збереження даних при зверненні за методом `POST` і тільки надаючи форму в інших випадках. Дія виконується тільки в тому випадку, якщо користувач має активний акаунт. Інакше користувачу ця дія не доступна.

Редагування даних власного місця проживання забезпечено функцією `edit_living_view`, номер проживання в якій вказано в параметрі `living`. Якщо користувач має активний акаунт, а це проживання належить йому, то виконується або заповнення форми даними, або збереження відповідних даних у БД після виконання перевірки їхньої коректності.

Створення сусідства забезпечено функцією представлення `add_neighborhood_view`. Принцип створення такий самий як і у проживання. Клас `NeighborhoodForm` відповідно тут використовується. При створенні запису в БД створюється запис з менеджерським і підтвердженим рівнем доступу для цього користувача. Тобто і створюється сусідство, і вносяться дані

про участь у цьому сусідстві. Якщо в сусідстві один користувач, то він же і є фактично власником. Далі ролі можуть змінюватися. Але фактично початково далі вирішує цей користувач, хто буде менеджером у сусідстві.

Редагування даних сусідства виконується через функцію `edit_neighborhood_view`. Фактично всі реалізовані функції редагування передбачають роботу з тим самим класом форми, що і створення. У даному випадку це клас `NeighborhoodForm`. Перед виконанням редагування відбувається перевірка прав доступу. Якщо користувач є учасником сусідства, має підтверджений менеджерський рівень доступу, то такі дії йому дозволяється виконати.

Аналогічно створення району забезпечується функцією представлення `add_region_view`, але з параметром `neighborhood`, який визначає сусідство, яке буде внесено одразу до цього району. Тобто сусідство не створюється фактично порожнім. При цьому перевіряється, що користувач має менеджерський підтверджений рівень доступу до цього сусідства.

Редагування даних району реалізовано функцією `edit_region_view`, в якій параметр `region` визначає номер району. Також реалізується перевірка прав доступу.

Інформування про виявлену підозрілу активність здійснюється на основі функції `inform_suspiciousactivity_view`, яка працює на основі класу форми `SuspiciousactivityForm` з параметром `severity`, що визначає серйозність виявленої активності, яку їй присвоює автор.

Редагування даних підозрілої активності реалізовано функцією `edit_suspiciousactivity_view`, яка має параметр `suspiciousactivity`, що визначає номер підозрілої активності, яку має бути змінено через використання форми `SuspiciousactivityForm`. Для виконання цих дій користувач має бути власником цієї активності.

Перегляд власних версій за виявленими підозрілими активностями реалізовано функцією `user_versions_view`. За авторством авторизованого

користувача витягаються всі його версії, які він висловив про підозрілі активності.

Функція представлення `set_severity_view` має параметри `suspiciousactivity` з номером підозрілої активності, `severity` зі ступенем серйозності, `neighborhood` з номером сусідства і визначає встановлення серйозності підозрілої активності. При цьому перевіряється, що користувач має підтверджене проживання в цьому сусідстві, за вказаним рівнем серйозності відбувається переобчислення серйозності підозрілої активності. Переобчислення відбувається шляхом усереднення виставлених значень всіма учасниками, які висловились. Рівнів серйозності всього три (підозріла, висока та критична). Якщо середній рівень вище 2 і вже висловили свої думки хоча би 2 користувачі, то всім користувачам у цьому сусідстві відбувається розсилка сповіщень про цю активність.

Визначення версії стосовно підозрілої активності реалізовано функцією представлення `add_version_view`, в параметрі якої `suspiciousactivity` визначається номер підозрілої активності.

Визначення близької людини забезпечується функцією `add_confidant_view`. У параметрі `user_confidant` має бути вказано номер запису про цю близьку людину. Створюється об'єкт класу `Confidant`, зберігається в БД, при чому неактивований. Активація відбувається через описані вище функції.

Відправлення повідомлення користувачу реалізовано функцією `add_directmessage_view`, параметр якої `user_dst` визначає номер користувача, якому адресовано повідомлення. Відбувається перевірка, що це користувач, який входить до району, в сусідство якого також входить авторизований користувач. Інакше відправити повідомлення неможливо. Підтримка роботи з самим повідомленням реалізується на основі класу форми `DirectmessageForm`. Також відбираються всі повідомлення, яка були надіслані між цими користувачами.

Відправлення повідомлення у спілкування реалізовано функцією представлення `add_conversation_view`, а загальне спілкування визначається номером сусідства з параметра `neighborhood`. Після перевірки того, що користувач має підтверджене проживання в цьому сусідстві, відбувається збереження даних у БД. Також реалізується перегляд повідомлень у спілкуванні за сусідством користувача, тобто за цим сусідством витягаються вже відправлені повідомлення.

Визначення інформації про заплановані дії реалізовано функцією представлення `add_report_view`, яка має параметр `neighborhood`, що визначає номер сусідства, в якому ці заплановані дії мають відбуватися. В основі створена форма `ReportForm`.

Голосування за версію стосовно підозрілої активності з номером `version` виконується через функцію представлення `version_vote_view`. Внесення голосу відбувається тільки тоді, коли його ще не було внесено в БД раніше. Якщо голос вже внесено, він не змінюється.

Підтвердження рівня доступу користувача у сусідстві реалізовано функцією представлення `living_approve_view`. Параметри цієї функції `neighborhoodliving` і `level` визначають відповідно номер сусідства і рівень доступу. Підтвердження може виконувати будь-який користувач, який проживає у цьому сусідстві. Якщо він не підтвердив цей факт, то користувач не зможе йому надсилати персональні сповіщення. Окрім того відбувається перевірка того, чи всі менеджери підтвердили участь користувача у сусідстві. Якщо так, то в БД зберігається, що цей рівень доступу підтверджений.

Підтвердження рівня доступу сусідства у районі реалізовано функцією представлення `region_approve_view`. Функція має параметр `regionneighborhood`, який визначає номер участі сусідства в районі. Підтвердження відбувається тільки в тому випадку, якщо авторизований користувач має менеджерський рівень доступу до хоча би одного сусідства з цього району.

Редагування персональних даних користувача реалізовано функцією представлення `edit_user_view`.

Додавання користувача до сусідства разом з вилученням реалізовано функцією `add_neighborhoodliving_view`, у параметрах якої визначається номер проживання `living`, номер сусідства `neighborhood` та необхідність менеджерського рівня доступу `manager_level`. Якщо такий запис вже внесено, то реалізується вилучення.

Персональне сповіщення користувача реалізується на основі функції представлення `personal_notification_view`, яка має параметри `user_dst`, що визначає адресата персонального сповіщення, та `text`, що визначає текст сповіщення. При цьому всі налаштування передаються на рівень клієнта, а там через `javascript` відбувається відправлення самого сповіщення. Так само влаштовані всі подібні представлення.

Сповіщення групи користувачів про критичну активність реалізовано функцією `group_notification_view`, яка має параметри `neighborhood` з номером сусідства, `text` з текстом сповіщення та `activity` з відповідним номером критичної активності. Функція вносить дані в БД про це сповіщення і відповідно передає всі необхідні налаштування і дані клієнту.

Відправлення сповіщення про необхідність термінової допомоги реалізовано функцією `alert_notification_view`, яка виконує групову розсилку сповіщень всім користувачам, яких авторизований користувач визначив для себе як близьких людей.

Додавання сусідства до району реалізується функцією представлення `add_regionneighborhood_view`, яка має параметри: `region`, що визначає номер району, `neighborhood`, що визначає номер сусідства, яке до цього району має бути внесено. Дозвіл на ці дії отримується тільки тоді, коли авторизований користувач має менеджерський рівень доступу до вказаного сусідства. У випадку, якщо цей доступ вже визначений, то реалізується вилучення сусідства з району.

Обробка відправлення самих сповіщень реалізована функцією `send_notification`. Перегляд даних сусідства реалізовано функцією `neighborhood_view`, яка має параметр `neighborhood` з номером цього сусідства. Перегляд даних житлового району реалізовано функцією `region_view`, параметр якої `region` визначає номер району. Перегляд повної інформації підозрілої активності реалізовано функцією `suspiciousactivity_view`, яка має параметр `suspiciousactivity` з номером цієї активності.

Перегляд поточного стану активностей реалізовано функцією `index_view`, яка підтримує роботу базової сторінки. Вона для авторизованого користувача виконує відбір всіх його районів, сусідств, витягає за ними всі підозрілі активності як за сусідством окремо, так і за районом загалом, звістки про заплановані дії, надіслані групові сповіщення.

Перегляд історії дій користувача виконується функцією представлення `user_actions_view`, яка виконує всі підозрілі активності, звістки про заплановані дії, сповіщення, які було створено авторизованим користувачем.

Перегляд статистики виявлення підозрілих активностей реалізовано функцією `statistics_view`. Дані відбираються за сусідством з номером `neighborhood` або за районом з номером `region`.

Представлення мають шаблони сторінок, за допомогою яких відбувається відтворення даних користувачу. В основі створення цих шаблонів використано макет [10].

3.3 Висновки за розділом 3

Описано процес підтримки підозрілої активності через визначення рівня серйозності проблеми та визначено правила створення автоматичних сповіщень користувачам. Реалізовано функції ПЗ.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис програми

Програма призначена для внесення даних про виявлені підозрілі активності, для інформування користувачів з деякої групи про виявлені підозрілі активності, враховуючи рівні доступу, для відслідковування наявної статистики підозрілої активності. Користувачі можуть просити про термінову допомогу, звертатися з проханням перевірити стан будинку.

4.2 Експлуатація програми

Експлуатація програми користувачем відповідає простим, інтуїтивно зрозумілим інтерфейсам сторінок (рис. 4.1).

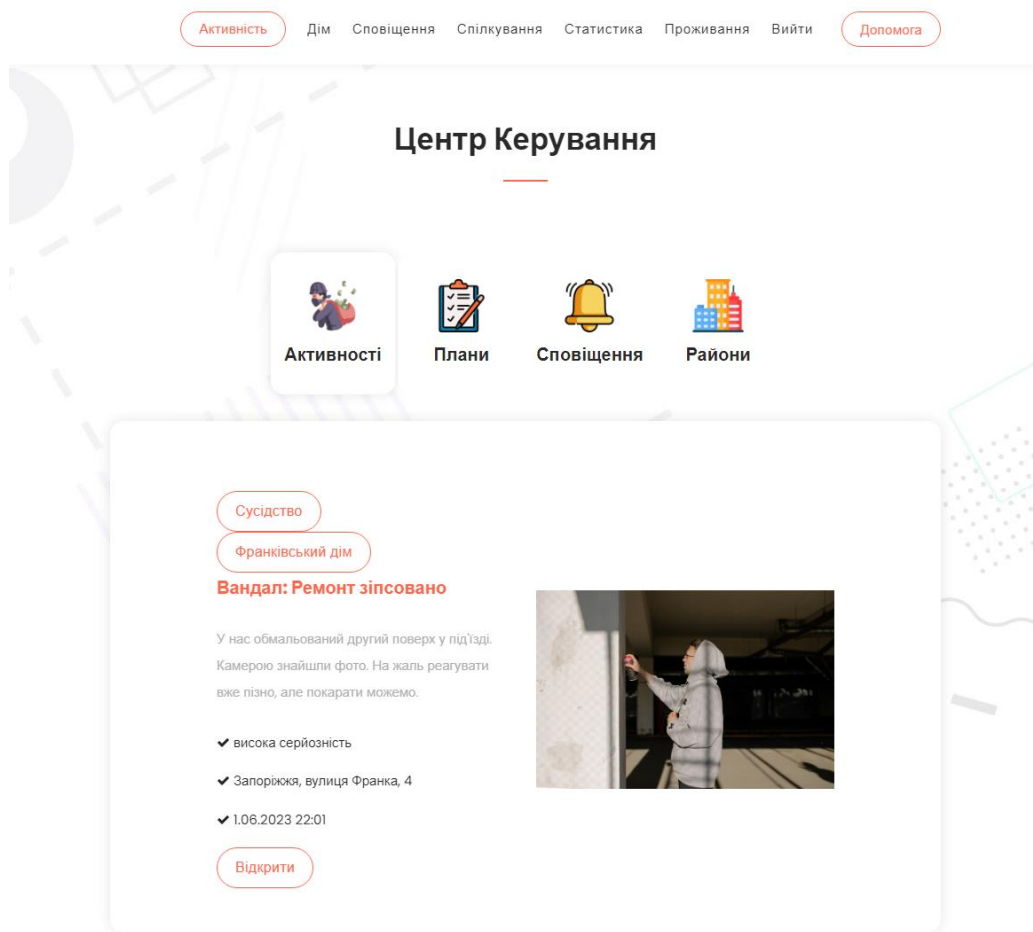


Рисунок 4.1 – Базова сторінка (відображення поточних станів)

Серед всіх сторінок виділено одну, яку названо Дім і яка містить базовий розділ, який відображається за замовчуванням. Наприклад, кожного разу після авторизації у програмі. Звідси можна отримати інформацію про поточний стан спостережень. Містяться підрозділи: «Активності» для перегляду всіх виявлених підозрілих активностей (за кожною приводиться назва, опис, рівень серйозності проблеми, адреса, за якою це констатовано, дата). Натискаючи кнопку «Відкрити», можна перейти до самого детального опису підозрілої активності та обговорення її. Відображення відбувається або за сусідством, або за районом. Для цього використовується посилання з відповідним іменем, яке при натисканні змінює ім'я (район або сусідство). Після натискання на цю кнопку нижче показуються кнопки з назвами сусідств або районів. Натискаючи на відповідну, можна відобразити потрібні повідомлення.

Сторінка району проживання містить перелік сусідств (рис. 4.2).

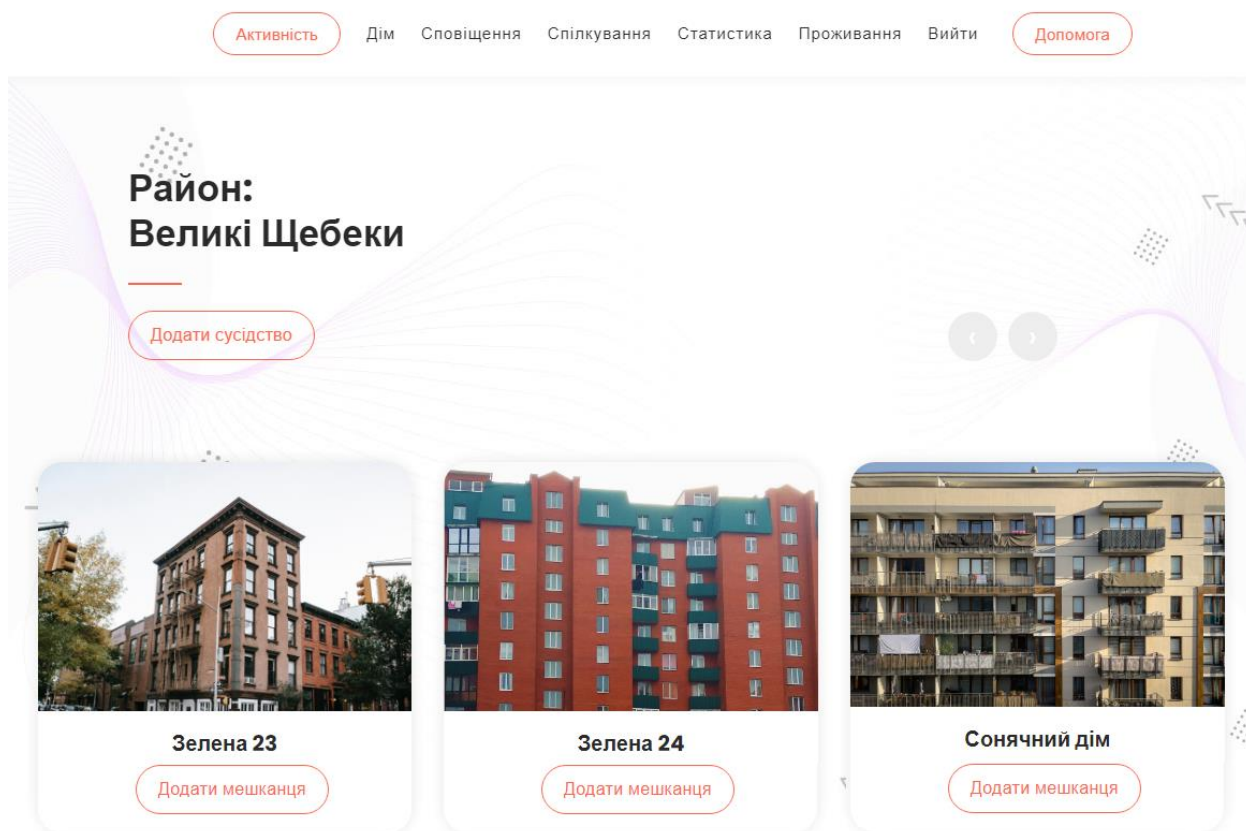


Рисунок 4.2 – Сторінка району проживання

Ця сторінка, як і всі інші, має дещо різний інтерфейс для користувача без додаткових прав і для користувача, який має менеджерську роль у деякому сусідстві (рис. 4.3).

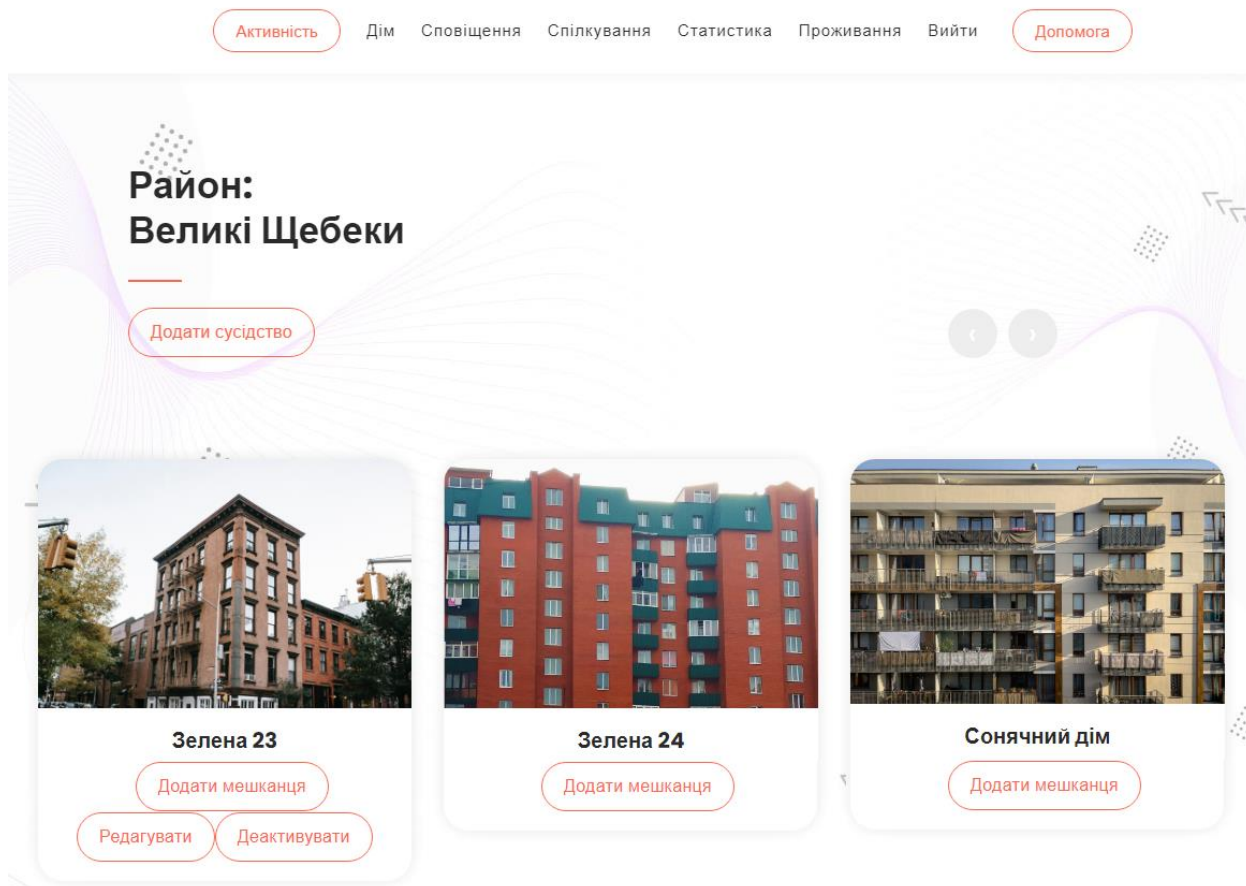


Рисунок 4.3 – Сторінка району проживання для менеджера одного з сусідств

Перемикаючись між розділами, можна подивитись на те, які дії було заплановано, щоб розуміти, наприклад, що підозри некоректні. Також можна отримати доступ до всіх направлених сповіщень та до переліку всіх районів. З тієї сторінки можна відкрити конкретний район, а згори сторінки буде відображено назву, кнопку для додавання нового сусідства до цього району та перелік сусідств, віднесений до цього району.

Для сусідств, у яких користувач є менеджером, дозволяється окрім того, щоб додати нового користувача до сусідства, також перейти до редагування даних та деактивації сусідства, якщо його потрібно фактично вилучити.

Якщо користувач взагалі в жодному сусідстві не виконує ролі менеджера, то йому не відображається і кнопка «Додати сусідство».

Якщо ж користувач відкрив з базової сторінки або зі сповіщення відповідну сторінку підозрілої активності (рис. 4.4), то йому відображається назва, дана активності, кнопка для переходу до створення своєї власної версії того, що відбулось, повного початкового повідомлення про цю активність (зліва разом з фотографією, яку було доєднано), даними автора та текстом повідомлення.

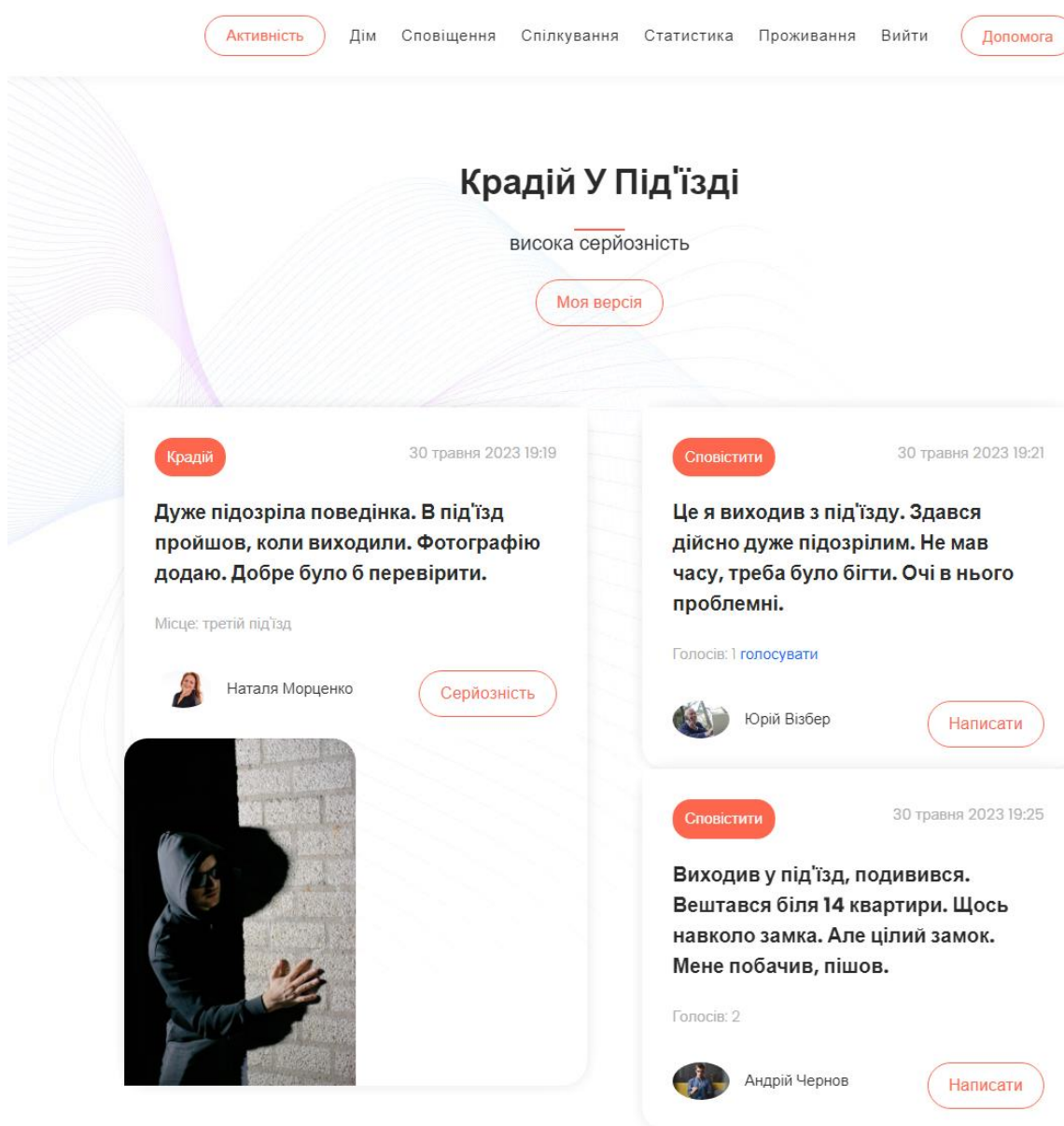


Рисунок 4.4 – Сторінка підозрілої активності

У правій частині сторінки відображаються версії інших користувачів. Біля кожної версії є кнопка «Сповістити» для створення сповіщення персонально цьому користувачу. Також кнопка «Написати» дозволяє написати персональне повідомлення користувачу. Самі ж повідомлення доступні з розділу меню «Сповіщення», який розділено на 2 вкладки, які визначають персональні сповіщення або за самим сусідством. Якщо сусідств декілька, то можна обрати потрібне.

З цієї сторінки також можна перейти до сповіщення конкретного користувача, також це можна зробити зі сторінки «Сповіщення» за відповідним розділом меню.

Розділ меню «Проживання» фактично є центром керування основними даними користувача, в тому числі персональними даними, звідки можна перейти до їх змінювання, переліку проживань, де можна додати нове місце проживання, а також звідти можна перейти до переліку близьких людей (рис. 4.5). Цей перелік містить всіх користувачів, яких для себе визначив як близьких користувач, що авторизувався. Якщо в користувача сталася небезпечна ситуація, він потребує допомоги, то він може натиснути на кнопку «Допомога» згори сторінки, відобразиться сторінка з можливістю вибору зокрема місця проживання, якому відповідає ця проблема. Якщо за 5 секунд можливий вибір не буде зроблений, то дані збережуться за замовчуванням. Але окрім збереження даних також буде відправлено сповіщення близьким людям. Близькі люди також мають можливість додаткового доступу. Для цього під відповідною близькою людиною, якій користувач вважає за потрібне довірити доглядати свій дім, потрібно натиснути на «Зробити відповідальним». У такому випадку на базовій сторінці такого користувача будуть відображатися не тільки його активності, але і будь-які активності, які виконуються за місцем проживання його близької людини.

Розділ меню «Статистика» (рис. 4.6) дозволяє відкрити статистику випадків, які трапилися.

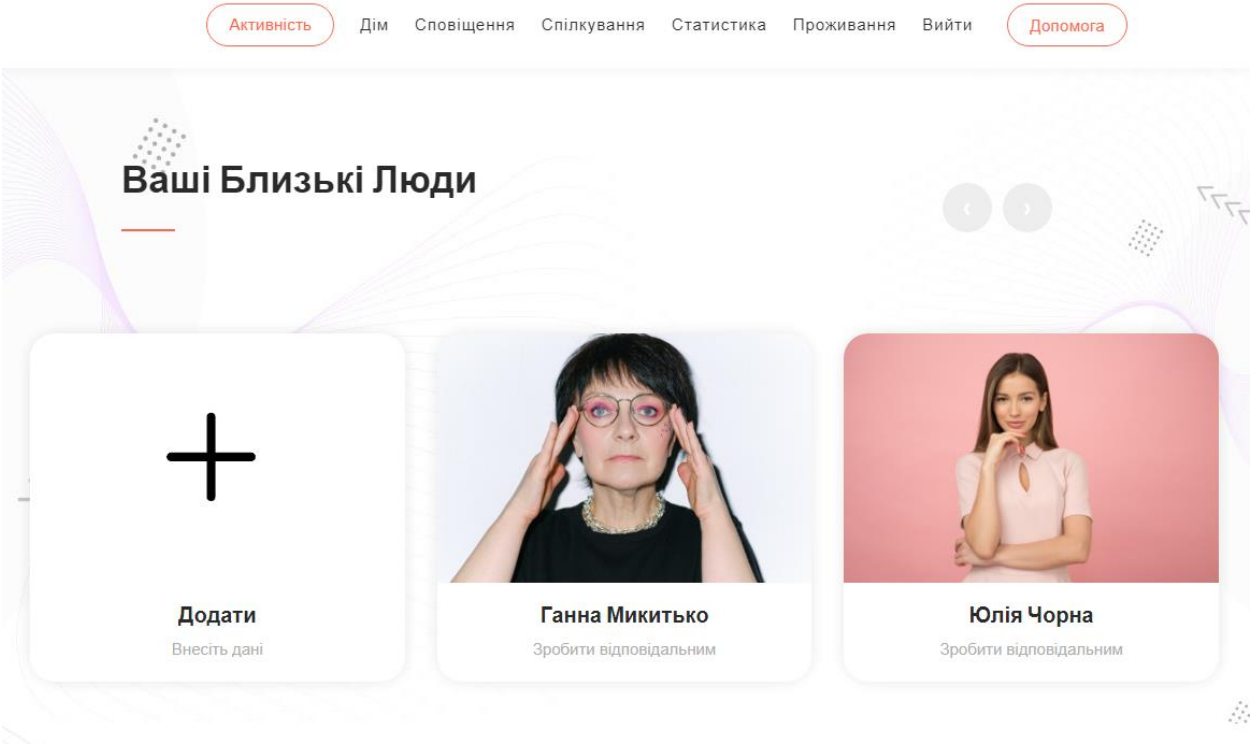


Рисунок 4.5 – Керування переліком близьких людей

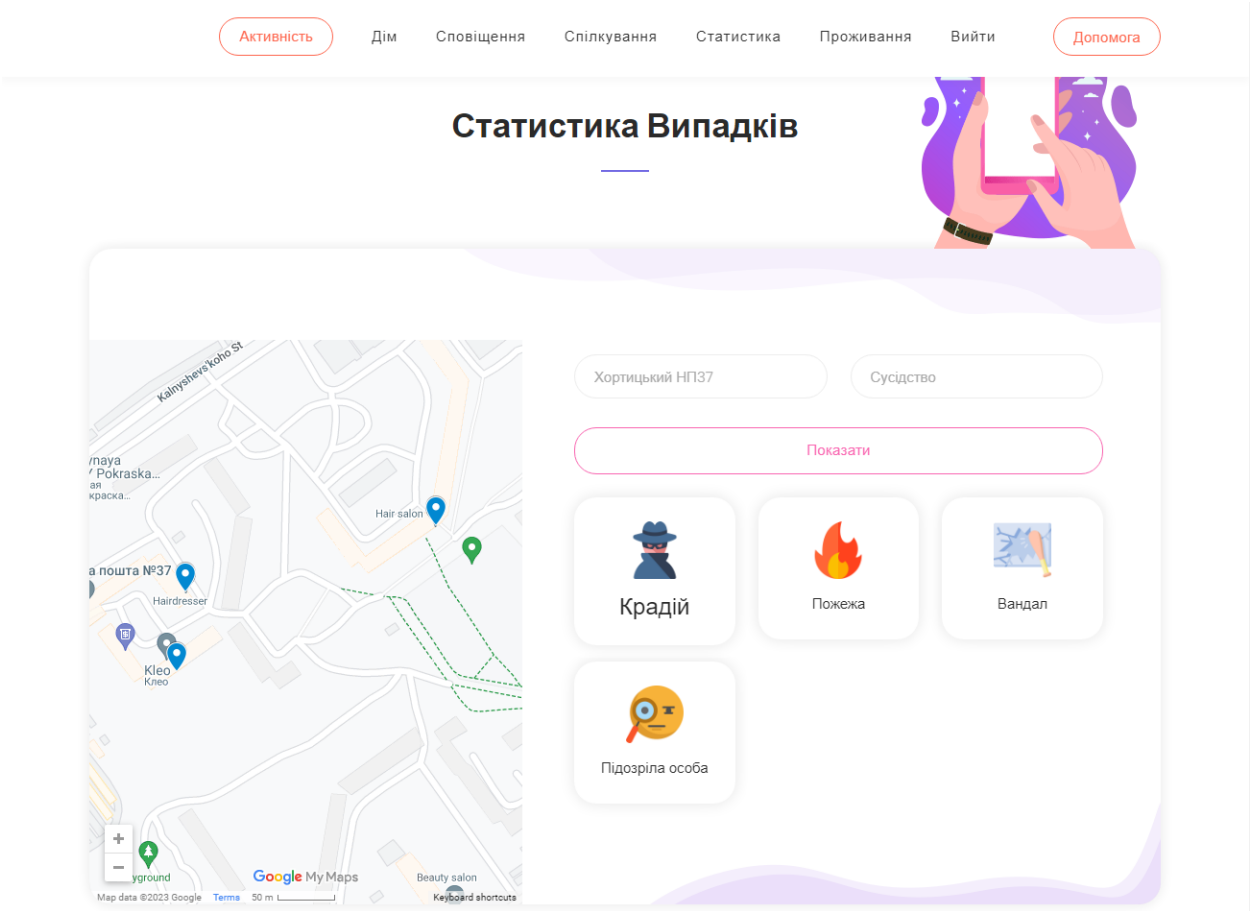


Рисунок 4.6 – Статистика випадків

Це статистика за обраним регіоном або сусідством, якщо воно було встановлено. У результаті на мапі відображаються адреси цього району або сусідства та галочки з активностями, що були зафіксовані. Для вибору активності потрібно натиснути на відповідну іконку. Якщо навести курсор на відповідну позначку з тим, що на цьому місці була певна підозріла активність, то відобразиться кількість таких випадків у вигляді підказки.

Керувати, наприклад, рівнем користувача в сусідстві можна, дійшовши до сторінки сусідства зі сторінки району. Там можна, наприклад, підтвердити участь певного користувача у сусідстві. Якщо користувач особисто цей факт не підтвердив, то персональні сповіщення від такого користувача приходити не будуть. Створити сповіщення також можна з меню «Активність», де можна визначити підозрілу активність або перейти до створення сповіщення.

4.3 Тестування програми

Під час тестування програми було за чеклистом, що відповідає функціональним вимогам, проведено перевірку роботи з програмою щодо отриманих результатів (табл. 4.1). Усі випробування програма пройшла успішно.

Таблиця 4.1 – Чеклист тестування програми

Сценарій	Підсумок
1	2
Перегляд повної інформації підозрілої активності	+
Перегляд власних записів про виявлені підозрілі активності	+
Перегляд сусідств за районом	+
Внесення власного місця проживання	+
Створення сусідства	+
Редагування даних сусідства	+

Продовження таблиці 4.1

1	2
Редагування даних власного місця проживання	+
Редагування персональних даних користувача	+
Авторизація користувачів	+
Реєстрація користувачів	+
Деактивація сусідства	+
Перегляд всіх власних місць проживання	+
Додавання користувача до сусідства	+
Вилучення користувача з сусідства	+
Створення району	+
Редагування даних району	+
Додавання сусідства до району	+
Вилучення сусідства з району	+
Підтвердження рівня доступу користувача у сусідстві	+
Підтвердження рівня доступу сусідства у районі	+
Перегляд сусідств користувача	+
Перегляд історії дій користувача	+
Пошук користувачів	+
Інформування про виявлену підозрілу активність	+
Редагування даних підозрілої активності	+
Перегляд поточного стану активностей	+
Перегляд даних житлового району	+
Визначення версії стосовно підозрілої активності	+
Голосування за версію стосовно підозрілої активності	+
Перегляд повідомлень у спілкуванні за сусідством користувача	+
Перегляд переліку спілкувань	+
Відправлення повідомлення у спілкування	+

Кінець таблиці 4.1

1	2
Перегляд списку користувачів, з якими були прямі повідомлення	+
Відправлення повідомлення користувачу з сусідства або району	+
Визначення інформації про заплановані дії	+
Відправлення сповіщення про необхідність термінової допомоги	+
Сповіщення групи користувачів про критичну активність	+
Персональне сповіщення користувача	+
Перегляд відправлених сповіщень користувачем	+
Перегляд історії отриманих сповіщень користувачем	+
Визначення близької людини	+
Перегляд власних близьких людей	+
Активація відповідальності близької людини	+
Деактивація відповідальності близької людини	+
Перегляд статистики виявлення підозрілих активностей за категоріями	+
Встановлення серйозності підозрілої активності	+
Перегляд власних версій за виявленими підозрілими активностями	+
Перегляд даних сусідства	+

4.4 Висновки за розділом 4

Усі функції реалізовано, інтерфейси роботи з ними забезпечено. Принципи роботи з програмою описано. Проведено тестування працездатності програми. Результати виявились успішними.

ВИСНОВКИ

Мета роботи полягала в розробці ПЗ моніторингу підозрілої активності в житловому районі для допомоги виявлення і вирішення безпекових проблем людям всередині спільноти, з представниками якої вони проживають поруч. Організувати цю допомогу вдалось за рахунок визначення в програмі певних сусідств, які є найнижчим рівнем взаємодії, та районів, які об'єднуються між собою сусідства. Для визначення правил взаємодії встановлено певні правила щодо того, хто і до чого саме який рівень доступу має. Фактично сусідства і райони для розмежовування такого доступу і використовуються.

У процесі спочатку проєктування, а потім і розробки було використано мову програмування Python, засоби веброзробки, до яких відносяться мови HTML, CSS, JavaScript і фреймворк Django, та система MySQL для керування базою даних.

Створена веброзробка визначає структуру роботи в цій сфері, включаючи визначені рівні доступу, основні виділені сутності, забезпечені процеси щодо взаємодії між користувачами, виконане проєктування і безпосередньо саму розробку. Усе це дозволяє в підсумку створити певну спільноту всередині програми, в якій люди готові допомагати один одному зокрема стосовно захисту власного майна.

Програма призначена для внесення даних про виявлені підозрілі активності, для інформування користувачів з деякої групи про виявлені підозрілі активності, враховуючи рівні доступу, для відслідковування наявної статистики підозрілої активності. Користувачі можуть просити про термінову допомогу, звертатися з проханням перевірити стан будинку.

Функції, які були визначені для необхідності забезпечення роботи за цією концепцією в цій предметній області, було реалізовано повністю, після чого проведено тестування, що в результаті завершилось успішно, дозволяючи використовувати створену програму на практиці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bouchrika, I. 15 Best Crime Analytics Software in 2023 [Electronic resource] / I. Bouchrika. – Access mode : <https://research.com/software/best-crime-analytics-software>.
2. WeAlert – safer neighborhoods – Apps on Google Play [Electronic resource]. – Access mode : <https://play.google.com/store/apps/details?id=nl.veiligebuurt.veiligebuurt&hl=en&gl=US>.
3. WeAlert – safer neighbourhoods on the App Store [Electronic resource]. – Access mode : <https://apps.apple.com/gb/app/wealert-safer-neighbourhoods/id1039643970>.
4. 7 Best Apps to Check Crime Rate in Area (Android & iOS) [Electronic resource]. – Access mode : <https://www.appslikethese.com/apps-to-check-crime-rate-in-area/>.
5. MyLocalCrime – Apps on Google Play [Electronic resource]. – Access mode : https://play.google.com/store/apps/details?id=com.spotcrime.mlc&hl=en_US&gl=US.
6. Emchenko, O. 9 Best Crime Tracking Apps & Websites 2023 [Electronic resource] / O. Emchenko. – Access mode : <https://freeappsforme.com/best-crime-trackers/>.
7. 8 Crime Watch Resources For Safer Neighborhoods [Electronic resource]. – Access mode : <https://www.prizmalarms.com/other/8-crime-watch-resources-for-safer-neighborhoods/>.
8. Bernard, J. Python Recipes Handbook [Text] : A Problem-Solution Approach / J. Bernard. – New York : Apress, 2016. – 155 p.
9. Forcier, J. Python Web Development with Django [Text] / J. Forcier. – Boston : Addison-Wesley Professional, 2008. – 402 p.
10. Digimedia Free Website Template. Free CSS Templates. Free CSS [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page290/digimedia>.

ДОДАТОК А
Технічне завдання

Вступ

Забезпечення безпеки на території місця проживання є важливим. При виявленні певної проблемної ситуації варто дізнатися, які дії заплановані, запитати у оточуючих, попросити іншого користувача сходити перевірити стан певного приміщення, відправити сповіщення про виявлену небезпеку і необхідність допомоги. За наявності цих засобів користувач може швидко повідомити інших користувачів про те, що сталося, проінформувати всіх завчасно про заплановані дії, співставити підозрілі активності з запланованими діями. Саме це і є основними завданнями, які мають бути вирішені в процесі розробки програмного забезпечення.

A.1 Підстави для розробки

Завдання на дипломну кваліфікаційну роботу сформульовано у відповідності з наказом № 87 від 4 квітня 2023 р. за Національним університетом «Запорізька політехніка», за яким було визначено тему «Програмне забезпечення моніторингу підозрілої активності в житловому районі».

A.2 Призначення розробки

Програмне забезпечення має бути в підсумку призначено для внесення даних про виявлені підозрілі активності, для інформування користувачів з деякої групи про виявлені підозрілі активності, враховуючи при цьому рівні доступу, для відслідковування наявної статистики підозрілої активності.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Забезпечення призначення програми виконується через реалізацію сценаріїв використання програми на основі наступних функцій:

- авторизація користувачів та реєстрація;
- редагування персональних даних користувача;
- перегляд даних житлового району;
- перегляд даних сусідства;
- перегляд повної інформації підозрілої активності;
- перегляд власних записів про виявлені підозрілі активності;
- перегляд сусідств за районом;
- внесення власного місця проживання;
- редагування даних власного місця проживання;
- перегляд всіх власних місць проживання;
- створення сусідства;
- редагування даних сусідства;
- деактивація сусідства;
- додавання користувача до сусідства;
- вилучення користувача з сусідства;
- підтвердження рівня доступу користувача у сусідстві;
- створення району;
- редагування даних району;
- додавання сусідства до району;
- вилучення сусідства з району;
- підтвердження рівня доступу сусідства у районі;
- перегляд сусідств користувача;
- перегляд поточного стану активностей;
- перегляд історії дій користувача;
- пошук користувачів;

- інформування про виявлену підозрілу активність;
- редагування даних підозрілої активності;
- визначення версії стосовно підозрілої активності;
- голосування за версію стосовно підозрілої активності;
- перегляд повідомлень у спілкуванні за сусідством користувача;
- перегляд переліку спілкувань;
- відправлення повідомлення у спілкування;
- перегляд списку користувачів, з якими були прямі повідомлення;
- відправлення повідомлення користувачу;
- визначення інформації про заплановані дії;
- відправлення сповіщення про необхідність термінової допомоги;
- сповіщення групи користувачів про критичну активність;
- персональне сповіщення користувача;
- перегляд відправлених сповіщень користувачем;
- перегляд історії отриманих сповіщень користувачем;
- визначення близької людини;
- перегляд власних близьких людей;
- активація відповідальності близької людини;
- деактивація відповідальності близької людини;
- перегляд статистики виявлення підозрілих активностей за категоріями;
- встановлення серйозності підозрілої активності;
- перегляд власних версій за виявленими підозрілими активностями.

A.3.2 Умови експлуатації

Експлуатація програми на сервері потребує підтримки, що включає навички роботи з БД, зокрема MySQL, виконання міграцій для Django-застосунків та внесення даних через адміністративний інтерфейс при необхідності розширення або зміни місць, типів розмірів сховищ або інших

каталогів стандартних даних.

А.3.3 Вимоги до складу та параметрів технічних засобів

Набір мінімальних технічних засобів, що мають забезпечуватися на сервері для підтримки роботи вебзастосунку, включає:

- процесор не менше ніж Xeon сімейства 3000 або Core 2 Duo;
- оперативна пам'ять обсягом у 4 Гб;
- простір на жорсткому диску з доступним вільним обсягом не менше ніж 3 Гб, але для можливості збереження файлів зображень бажано не менше 20 Гб;
- мережна карта і підключення до мереж Інтернет.

А.4 Порядок контролю та приймання

Для виконання дипломної кваліфікаційної роботи критичним є виконання визначеного завдання, що включає безпосередньо завдання на роботу і технічне завдання, за темою, яка зафіксована наказом, у строки, які визначені завданням на роботу.

У підсумку виконується захист роботи за оприлюдненим наперед графіком перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

from .models import Suspiciousactivity, Directmessage, Region,
RegionNeighborhood, NeighborhoodLiving, Neighborhood, Notification,
PersonalNotification, Confidant, NeighborhoodLiving, Version,
SuspiciousactivitySeverity, Report, User, Living,
NeighborhoodLivingApprove, RegionNeighborhoodApprove,
ActivityCategory, VersionVote, Conversation

from django.db.models import Q
from django.views.decorators.csrf import csrf_exempt
from django.conf import settings
from django.views.decorators.http import require_POST
from django.shortcuts import render, redirect
import json

from .forms import NeighborhoodForm, LivingForm, RegionForm,
SuspiciousactivityForm, VersionForm, DirectmessageForm, ReportForm,
UserForm

from webpush import send_user_notification


def my_suspiciousactivities_view(request):
    user = request.user
    if user.active:
        suspiciousactivities =
Suspiciousactivity.objects.prefetch_related("photos").filter(user =
user).order_by("-observation_date")
        return render(request, "suspicionmonitor/activities.html",
{"suspiciousactivities": suspiciousactivities,})


def region_neighborhoods_view(request, region):
    user = request.user
    if user.active:
        try:
            region = Region.objects.get(name = region)
        except Region.DoesNotExist:
            region = None
        if user.active and region:
            neighborhoods =
NeighborhoodLiving.objects.select_related("neighborhood").select_relat
ed("living").filter(living__user = user, manager_level = True,
approved = True).distinct("neighborhood")
            if neighborhoods:
                manager_region_neighborhoods =
RegionNeighborhood.objects.select_related("region").select_related("ne
ighborhood").filter(region = region, neighborhood__in = neighborhoods,
approved = False).distinct("neighborhood")
                region_neighborhoods =
RegionNeighborhood.objects.select_related("region").select_related("ne

```

```

neighborhood").filter(approved = True, region =
region).distinct("neighborhood")
        return render(request,
"suspicionmonitor/region_neighbors.html", {"region": region,
"neighborhoods": region_neighborhoods, "manager_neighborhoods ":
manager_region_neighborhoods })

def neighborhood_deactivate_view(request, neighborhood):
    user = request.user
    if user.active:
        try:
            neighborhood = Neighborhood.objects.get(id =
neighborhood)
        except Neighborhood.DoesNotExist:
            neighborhood = None
        if user.active and neighborhood:
            neighborhood =
NeighborhoodLiving.objects.select_related("neighborhood").select_relat
ed("living").filter(living__user = user, manager_level = True,
approved = True, neighborhood = neighborhood).distinct("neighborhood")
            if neighborhood:
                neighborhood.active = False
                neighborhood.save()
                redirect("neighborhood", neighborhood.pk)

def user_search_view(request, name = "", login = ""):
    if request.user.active:
        users = User.objects.all()
        if name:
            name_split = name.split(" ")
            surname_detected = name_split[0]
            name_detected = name_split[1]
            name_changed = name_detected + " " +
surname_detected
            users = users.objects.filter(Q(fullname__icontains =
name) | Q(fullname__icontains = name_changed))
        if login:
            users = users.objects.filter(username = login)
            users = users.objects.filter(active = True)
            users = users.values_list("username", "bio", "photo",
"fullname")
        return render(request, "suspicionmonitor/user_search.html",
{"users": users })

def user_neighborhoods_view(request):

```

```

        if request.user.active:
            user = request.user
            neighborhoods =
NeighborhoodLiving.objects.select_related("neighborhood").filter(livin
g__user = user, approved = True).distinct("neighborhood")
            return render(request,
"suspicionmonitor/region_neighbors.html", {"neighborhoods": neighborhoods
})

def user_notifications_view(request):
    user = request.user
    if user.active:
        groupnotifications =
Notification.objects.select_related("nbliving").select_related("nblivi
ng__living").select_related("activitycategory").filter(living__user =
user).order_by("-created_date")
        personalnotifications =
PersonalNotification.objects.select_related("user_dst").filter(user_sr
c = user).order_by("-created_date")
        notifications = []
        group = 0
        person = 0
        while group < len(groupnotifications) and person <
len(personalnotifications):
            if group >= len(groupnotifications):
                for notification in
personalnotifications[person:]:
                    notifications.append(notification)
                    person = person + 1
                break
            if person >= len(personalnotifications):
                for notification in groupnotifications[group:]:
                    notifications.append(notification)
                    group = group + 1
                break
            if groupnotifications[group].created_date >
personalnotifications[person].created_date:
                notifications.append(groupnotifications[group])
                group = group + 1
            else:
notifications.append(personalnotifications[person])
                person = person + 1

```

```

        return render(request,
"suspicionmonitor/user_notifications.html", {"notifications":
notifications })

def to_user_notifications_view(request):
    user = request.user
    if user.active:
        groupnotifications =
Notification.objects.select_related("nbliving").select_related("nblivi
ng__living").select_related("activitycategory").filter(living__user =
user).order_by("-created_date")
        personalnotifications =
PersonalNotification.objects.select_related("user_src").filter(user_ds
t = user).order_by("-created_date")
        notifications = []
        group = 0
        person = 0
        while group < len(groupnotifications) and person <
len(personalnotifications):
            if group >= len(groupnotifications):
                for notification in
personalnotifications[person:]:
                    notifications.append(notification)
                    person = person + 1
                break
            if person >= len(personalnotifications):
                for notification in groupnotifications[group:]:
                    notifications.append(notification)
                    group = group + 1
                break
            if groupnotifications[group].created_date >
personalnotifications[person].created_date:
                notifications.append(groupnotifications[group])
                group = group + 1
            else:
                notifications.append(personalnotifications[person])
                person = person + 1
        return render(request,
"suspicionmonitor/to_user_notifications.html", {"notifications":
notifications })

def direct_users_view(request):
    user = request.user
    if user.active:

```

```

        src_users = Directmessage.objects.filter(user_dst =
user).order_by("-created_date").distinct("user_src")
        dst_users = Directmessage.objects.filter(user_src =
user).order_by("-created_date").distinct("user_dst")
        users = []
        src_user = 0
        dst_user = 0
        while src_user < len(src_users) and dst_user <
len(dst_users):
            if src_user >= len(src_users):
                for user in dst_users[dst_user:]:
                    users.append(user)
                    dst_user = dst_user + 1
                break
            if dst_user >= len(dst_users):
                for user in src_users[src_user:]:
                    users.append(user)
                    src_user = src_user + 1
                break
            last_message_src_1 =
Directmessage.objects.filter(user_src_id =
src_users[src_user]).order_by("-created_date")[0]
            last_message_dst_1 =
Directmessage.objects.filter(user_dst_id =
src_users[src_user]).order_by("-created_date")[0]
            if last_message_dst_1.created_date >
last_message_src_1.created_date:
                last_message_1 = last_message_dst_1.created_date
            else:
                last_message_1 = last_message_src_1.created_date
            last_message_src_2 =
Directmessage.objects.filter(user_src_id =
dst_users[dst_user]).order_by("-created_date")[0]
            last_message_dst_2 =
Directmessage.objects.filter(user_dst_id =
dst_users[dst_user]).order_by("-created_date")[0]
            if last_message_dst_2.created_date >
last_message_src_2.created_date:
                last_message_2 = last_message_dst_2.created_date
            else:
                last_message_2 = last_message_src_2.created_date
            if last_message_1 > last_message_2:
                users.append(src_users[src_user])
                src_user = src_user + 1
            else:

```

```

        users.append(dst_users[dst_user])
        dst_user = dst_user + 1
    return render(request,
"suspicionmonitor/direct_users.html", {"users": users })

def user_livings_view(request):
    user = request.user
    if user.active:
        livings =
Living.objects.select_related("user").filter(user = user)
        return render(request,
"suspicionmonitor/user_livings.html", {"livings": livings})

def user_confidants_view(request):
    user = request.user
    if user.active:
        confidants =
Confidant.objects.select_related("user").filter(user_src = user)
        return render(request,
"suspicionmonitor/user_confidants.html", {"confidants": confidants })

def confidant_activate_view(request, confidant):
    user = request.user
    if user.active:
        try:
            confidant = Confidant.objects.get(id = confidant)
        except Confidant.DoesNotExist:
            confidant = None
        if confidant:
            confidant.activation = True
            confidant.save()
            redirect("confidants")

def confidant_deactivate_view(request, confidant):
    user = request.user
    if user.active:
        try:
            confidant = Confidant.objects.get(id = confidant)
        except Confidant.DoesNotExist:
            confidant = None
        if confidant:
            confidant.activation = False
            confidant.save()
            redirect("confidants")

```

```

def add_living_view(request):
    user = request.user
    if request.user.active:
        if request.method == "POST":
            living_form = LivingForm(request.POST)
            if living_form.is_valid():
                living = living_form.save(commit=False)
                living.user = user
                living.save()
                redirect("livings")
        else:
            living_form = LivingForm()
    else:
        living_form = None
    return render(request, "suspicionmonitor/living_process.html",
{"form": living_form})

def edit_living_view(request, living):
    user = request.user
    try:
        living = Living.objects.get(id = living)
    except Living.DoesNotExist:
        living = None
    if request.user.active and living.user == user:
        if request.method == "POST":
            living_form = LivingForm(request.POST, instance =
living)

            if living_form.is_valid():
                living = living_form.save(commit=False)
                living.user = user
                living.save()
                redirect("livings")
        else:
            living_form = LivingForm(None, instance = living)
    else:
        living_form = None
    return render(request, "suspicionmonitor/living_process.html",
{"form": living_form })

def add_neighborhood_view(request, living):
    user = request.user
    try:
        living = Living.objects.get(id = living)
    except Living.DoesNotExist:
        living = None

```

```

        if request.user.active and living.user == user:
            if request.method == "POST":
                neighborhood_form = NeighborhoodForm(request.POST)
                if neighborhood_form.is_valid():
                    neighborhood =
neighborhood_form.save(commit=False)
                    neighborhood_form.save()
                    neighborhood_living =
NeighborhoodLiving(neighborhood = neighborhood, living = living,
manager_level = True, approved = True)
                    neighborhood_living.save()
                    redirect("neighborhoods")
            else:
                neighborhood_form = NeighborhoodForm()
        else:
            neighborhood_form = None
            livings = Living.objects.filter(user = user)
            return render(request,
"suspicionmonitor/neighborhood_process.html", {"form": neighborhood_form,
"livings": livings, "new_messages"})

def edit_neighborhood_view(request, neighborhood):
    user = request.user
    try:
        neighborhood = Neighborhood.objects.get(id =
neighborhood)
    except Neighborhood.DoesNotExist:
        neighborhood = None
        rights = NeighborhoodLiving.objects.filter(neighborhood =
neighborhood, living__user = user, manager_level = True, approved =
True)
        if rights and request.user.active:
            if request.method == "POST":
                neighborhood_form = NeighborhoodForm(request.POST,
instance = neighborhood)
                if neighborhood_form.is_valid():
                    neighborhood_form.save()
                    redirect("neighborhoods")
            else:
                neighborhood_form = NeighborhoodForm(None, instance
= neighborhood)
        else:
            neighborhood_form = None

```

```

        return render(request,
"suspicionmonitor/neighborhood_process.html", {"form": neighborhood_form
})

```

```

def add_region_view(request, neighborhood):
    user = request.user
    try:
        neighborhood = Neighborhood.objects.get(id =
neighborhood)
    except Neighborhood.DoesNotExist:
        neighborhood = None
    neighborhoodlivings =
NeighborhoodLiving.objects.filter(neighborhood = neighborhood,
living__user = user, manager_level = True, approved = True)
    if neighborhoodlivings and request.user.active:
        if request.method == "POST":
            region_form = RegionForm(request.POST)
            if region_form.is_valid():
                region = region_form.save(commit=False)
                region.save()
                region_neighborhood =
RegionNeighborhood(neighborhood = neighborhood, region = region,
approved = True)
                region_neighborhood.save()
                redirect("region", region.pk)
        else:
            region_form = RegionForm()
    else:
        region_form = None
    return render(request, "suspicionmonitor/region_process.html",
{"form": region_form })

```

```

def edit_region_view(request, region):
    user = request.user
    try:
        region = Region.objects.get(id = region)
    except Region.DoesNotExist:
        region = None
    neighborhoods =
NeighborhoodLiving.objects.prefetch_related("neighborhood").filter(liv
ing__user = user, manager_level = True, approved =
True).distinct("neighborhood")
    rights = RegionNeighborhood.objects.filter(region = region,
neighborhood__in = neighborhoods, approved = True)
    if rights and request.user.active:

```

```

        if request.method == "POST":
            region_form = RegionForm(request.POST, instance =
region)

            if region_form.is_valid():
                region_form.save()
                redirect("region", region.pk)
            else:
                region_form = RegionForm(None, instance = region)
        else:
            region_form = None
        return render(request, "suspicionmonitor/region_process.html",
{"form": region_form })

def inform_suspiciousactivity_view(request, severity):
    user = request.user
    if request.user.active:
        if request.method == "POST":
            suspiciousactivity_form =
SuspiciousactivityForm(request.POST)
            if suspiciousactivity_form.is_valid():
                nbliving =
suspiciousactivity_form.cleaned_data['living']
                if nbliving.living.user == user:
                    suspiciousactivity =
suspiciousactivity_form.save(commit=False)
                    suspiciousactivity.save()
                    suspiciousactivity_form.save_m2m()
                    suspiciousactivity_severity =
SuspiciousactivitySeverity(suspiciousactivity = suspiciousactivity, user =
user, severity = severity)
                    suspiciousactivity_severity.save()
                    redirect("mysuspiciousactivities")
                else:
                    suspiciousactivity_form = SuspiciousactivityForm()
            else:
                suspiciousactivity_form = None
        return render(request,
"suspicionmonitor/suspiciousactivity_process.html", {"form":
suspiciousactivity_form })

def edit_suspiciousactivity_view(request, suspiciousactivity):
    user = request.user
    try:

```

```

        suspiciousactivity =
Suspiciousactivity.objects.select_related("living").select_related("us
er").filter(id = suspiciousactivity).get()
        except Suspiciousactivity.DoesNotExist:
            suspiciousactivity = None
            if suspiciousactivity.nbliving.living.user == user and
request.user.active:
                if request.method == "POST":
                    suspicactivity_form =
SuspiciousactivityForm(request.POST, instance = suspiciousactivity)
                    if suspicactivity_form.is_valid():
                        suspicactivity_form.save()
                        redirect("suspicactivity",
suspiciousactivity.pk)
                    else:
                        suspicactivity_form = SuspiciousactivityForm(None,
instance = suspiciousactivity)
                    else:
                        suspicactivity_form = None
                return render(request,
"suspicionmonitor/suspicactivity_process.html", {"form":
suspicactivity_form })

def user_versions_view(request):
    user = request.user
    if user.active:
        versions =
Version.objects.select_related("suspiciousactivity").filter(user =
user).order_by("-created_date")
        return render(request,
"suspicionmonitor/user_versions.html", {"versions": versions })

def set_severity_view(request, suspiciousactivity, severity,
neighborhood = None):
    user = request.user
    try:
        suspiciousactivity =
Suspiciousactivity.objects.select_related("living").select_related("us
er").filter(id = suspiciousactivity).get()
        except Suspiciousactivity.DoesNotExist:
            suspiciousactivity = None
    try:
        neighborhood = Neighborhood.objects.filter(id =
neighborhood).get()
    except Neighborhood.DoesNotExist:

```

```

        neighborhood = None
    if user.active:
        neighborhoods =
NeighborhoodLiving.objects.filter(living__user = user, approved =
True).distinct("neighborhood")
        if not neighborhood:
            neighborhood = neighborhoods[0]
            living = NeighborhoodLiving.objects.filter(neighborhood
= neighborhood, living__user = user, approved = True).get()
            suspicseverity =
SuspiciousactivitySeverity.objects.filter(suspiciousactivity =
suspiciousactivity, user = user)
            if suspicseverity:
                suspicseverity.severity = severity
                suspicseverity.save()
                severities =
SuspiciousactivitySeverity.objects.filter(suspiciousactivity =
suspiciousactivity)
                sevetiry_sum = 0
                severity_cases = severities.count()
                for severity in severities:
                    sevetiry_sum = sevetiry_sum + severity.severity
                sevetiry_sum = sevetiry_sum / severity_cases
                suspiciousactivity.severity = sevetiry_sum
                suspiciousactivity.save()
                if severity_cases > 1 and sevetiry_sum > 2:
                    psettings = getattr(settings,
'WEBPUSH_SETTINGS', {})
                    key = psettings.get('VAPID_PUBLIC_KEY')
                    if sevetiry_sum > 2.9:
                        notification = Notification(nbliving =
living, activitycategory = suspiciousactivity.activitycategory, text =
suspiciousactivity.title, level = 2)
                        notification.save()
                    else:
                        notification = Notification(nbliving =
living, activitycategory = suspiciousactivity.activitycategory, text =
suspiciousactivity.title, level = 1)
                        notification.save()
                return render(request,
"suspicionmonitor/set_severity.html", {"key": key, "severity":
sevetiry_sum, "neighborhoods": neighborhoods, "notification":
notification })
        else:

```

```

        suspicseverity = SuspiciousactivitySeverity(severity
= severity, suspiciousactivity = suspiciousactivity, user = user)
        suspicseverity.save()
        return render(request,
"suspicionmonitor/user_versions.html", {"key": None, "severity": 0,
"neighborhoods": []})

def add_version_view(request, suspiciousactivity):
    user = request.user
    try:
        suspiciousactivity =
Suspiciousactivity.objects.select_related("living").select_related("us
er").filter(id = suspiciousactivity).get()
    except Suspiciousactivity.DoesNotExist:
        suspiciousactivity = None
    if request.user.active:
        if request.method == "POST":
            version_form = VersionForm(request.POST)
            if version_form.is_valid():
                living = version_form.cleaned_data['living']
                if living.user == user:
                    living.save()
                    redirect("suspicionactivity",
suspiciousactivity.pk)
            else:
                version_form = VersionForm()
        else:
            version_form = None
    return render(request, "suspicionmonitor/version_process.html",
{"form": version_form })

def add_confidant_view(request, user_confidant):
    user = request.user
    try:
        user_confidant = User.objects.get(id = user_confidant)
    except User.DoesNotExist:
        user_confidant = None
    if user.active:
        confidant = Confidant(user_src = user, user_dst =
user_confidant, activation = False)
        confidant.save()
        redirect("confidants")

def add_directmessage_view(request, user_dst):
    user = request.user

```

```

        my_neighborhoods =
NeighborhoodLiving.objects.select_related("neighborhood").filter(livin
g__user = user, approved = True).distinct("neighborhood")
        my_regions =
RegionNeighborhood.objects.select_related("neighborhood").filter(neigh
borhood__in = my_neighborhoods, approved = True).distinct("region")
        try:
            user_dst = User.objects.get(id = user_dst)
        except User.DoesNotExist:
            user_dst = None
        if user_dst:
            my_neighborhoods_dst =
NeighborhoodLiving.objects.select_related("neighborhood").filter(livin
g__user = user_dst, approved = True).distinct("neighborhood")
            my_regions_dst =
RegionNeighborhood.objects.select_related("neighborhood").filter(neigh
borhood__in = my_neighborhoods_dst, approved =
True).distinct("region")
            rights = False
            for my_region in my_regions:
                if my_region in my_regions_dst:
                    rights = True
                    break
            if rights and request.user.active:
                if request.method == "POST":
                    directmessage_form = DirectmessageForm(request.POST)
                    if directmessage_form.is_valid():
                        directmessage = directmessage_form.save(commit =
False)

                        directmessage.user_src = user
                        directmessage.save()
                        redirect("suspiciousactivity",
suspiciousactivity.pk)
                    else:
                        directmessage_form = DirectmessageForm()
                else:
                    directmessage_form = None
                    directmessages = Directmessage.objects.filter(Q(user_src =
user, user_dst = user_dst) | Q(user_dst = user, user_src =
user_dst)).order_by("-created_date")
                    return render(request, "suspicionmonitor/direct_message.html",
{"form": directmessage_form, "messages": directmessages })

def add_conversation_view(request, neighborhood):
    user = request.user

```



```

        report.nbliving = nbliving
        report.save()
        redirect("index")
    else:
        report_form = ReportForm()
    else:
        report_form = None
    return render(request, "suspicionmonitor/report_process.html",
{"form": report_form, "new_messages"})

def version_vote_view(request, version):
    user = request.user
    if user.active:
        try:
            version =
Version.objects.select_related("version").select_related("version__sus
piciousactivity").filter(id = version).get()
        except Version.DoesNotExist:
            version = None
        if version:
            vote = VersionVote.objects.filter(user = user,
version = version)
            if not vote:
                version_vote = VersionVote(user = user, version
= version)

                version_vote.save()
                redirect("suspiciousactivity",
version.suspiciousactivity.pk)

def living_approve_view(request, neighborhoodliving, level):
    user = request.user
    if user.active:
        try:
            neighborhoodliving =
NeighborhoodLiving.objects.select_related("neighborhood").filter(id =
neighborhoodliving).get()
        except NeighborhoodLiving.DoesNotExist:
            neighborhoodliving = None
        my_neighborhoodliving =
NeighborhoodLiving.objects.filter(neighborhood =
neighborhoodliving.neighborhood, living__user = user)
        approve = NeighborhoodLivingApprove(nbliving_src =
my_neighborhoodliving, nbliving_dst = neighborhoodliving, level =
level)

        approve.save()

```

```

        if not neighborhoodliving.approved:
            managers =
NeighborhoodLiving.objects.filter(neighborhood =
neighborhoodliving.neighborhood, manager_level = True, approved =
True)

            approves =
NeighborhoodLivingApprove.objects.filter(nbliving_dst =
neighborhoodliving, nbliving_src__in = managers)
            if managers.count() == approves.count():
                neighborhoodliving.approved = True
                neighborhoodliving.save()
            redirect("neighborhood",
neighborhoodliving.neighborhood.pk)

def region_approve_view(request, regionneighborhood):
    user = request.user
    if user.active:
        try:
            regionneighborhood =
RegionNeighborhood.objects.select_related("neighborhood").select_relat
ed("region").filter(id = regionneighborhood).get()
        except RegionNeighborhood.DoesNotExist:
            regionneighborhood = None
        my_neighborhoods =
NeighborhoodLiving.objects.filter(neighborhood =
neighborhoodliving.neighborhood, living__user = user, manager_level =
True, approved = True).distinct("neighborhood")
        my_regions =
RegionNeighborhood.objects.select_related("region").filter(neighborhoo
d__in = my_neighborhoods, approved = True).distinct("region")
        if user.active and (regionneighborhood.region in
my_regions):
            approve =
RegionNeighborhoodApprove.objects.filter(user = user,
regionneighborhood = regionneighborhood)
            if not approve:
                approve =
RegionNeighborhoodApprove.objects.filter(regionneighborhood =
regionneighborhood, user = user)
                approve.save()
            approves =
RegionNeighborhoodApprove.objects.filter(regionneighborhood =
regionneighborhood)

```

```

        neighborhoods =
RegionNeighborhood.objects.filter(approved = True, region =
regionneighborhood.region)
        managers =
NeighborhoodLiving.objects.filter(manager_level = True, approved =
True, neighborhood__in = neighborhoods)
        if managers.count() == approves.count():
            regionneighborhood.approved = True
            regionneighborhood.save()
        redirect("neighborhood",
regionneighborhood.neighborhood.pk)

def edit_user_view(request):
    user = request.user
    if user.active:
        if request.method == "POST":
            user_form = UserForm(request.POST, instance = user)
            if user_form.is_valid():
                user_form.save()
                redirect("index")
        else:
            user_form = UserForm(None, instance = user)
    else:
        user_form = None
    return render(request, "suspicionmonitor/user.html", {"form":
user_form})

def add_neighborhoodliving_view(request, living, neighborhood,
manager_level = False):
    user = request.user
    if user.active:
        try:
            neighborhood = Neighborhood.objects.get(id =
neighborhood)
        except Neighborhood.DoesNotExist:
            neighborhood = None
        try:
            living = Living.objects.get(id = living)
        except Living.DoesNotExist:
            living = None
        if neighborhood and living:
            now_living =
NeighborhoodLiving.objects.filter(neighborhood = neighborhood, living
= living)
            if now_living:

```

```

        now_living.delete()
        redirect("index")
    else:
        now_living = NeighborhoodLiving(neighborhood =
neighborhood, living = living, approved = False, manager_level =
manager_level)

        now_living.save()
        redirect("index")

def personal_notification_view(request, user_dst, text):
    user = request.user
    try:
        user_dst = User.objects.get(id = user_dst)
    except Neighborhood.DoesNotExist:
        user_dst = None
    if user.active and user_dst:
        my_neighbors =
NeighborhoodLiving.objects.select_related("living__user").filter(livin
g__user = user, approved = True).distinct("living__user")
        if user_dst in my_neighbors:
            psettings = getattr(settings, 'WEBPUSH_SETTINGS',
{})

            key = psettings.get('VAPID_PUBLIC_KEY')
            notification =
PersonalNotification.objects.filter(user_src = user, user_dst =
user_dst, text)

            notification.save()
            return render(request,
"suspicionmonitor/personal_notification.html", {"key": key,
"notification": notification })

def personal_notification_view(request, user_dst, text):
    user = request.user
    try:
        user_dst = User.objects.get(id = user_dst)
    except User.DoesNotExist:
        user_dst = None
    if user.active and user_dst:
        my_neighbors =
NeighborhoodLiving.objects.select_related("living__user").filter(livin
g__user = user, approved = True).distinct("living__user")
        approve =
NeighborhoodLivingApprove.objects.filter(nbliving_src__living__user =
user)

        if user_dst in my_neighbors and approve:

```

```

        psettings = getattr(settings, 'WEBPUSH_SETTINGS',
    {}))

        key = psettings.get('VAPID_PUBLIC_KEY')
        notification =
PersonalNotification.objects.filter(user_src = user, user_dst =
user_dst, text)

        notification.save()
        return render(request,
"suspicionmonitor/personal_notification.html", {"key": key,
"notification": notification })

    def group_notification_view(request, neighborhood, text,
activity = None):
        user = request.user
        try:
            neighborhood = Neighborhood.objects.get(id =
neighborhood)
        except Neighborhood.DoesNotExist:
            neighborhood = None
        if user.active and user_dst:
            try:
                nbliving =
NeighborhoodLiving.objects.get(neighborhood = neighborhood,
living__user = user, approved = True)
            except NeighborhoodLiving.DoesNotExist:
                nbliving = None
            psettings = getattr(settings, 'WEBPUSH_SETTINGS', {})
            key = psettings.get('VAPID_PUBLIC_KEY')
            try:
                activitycategory = ActivityCategory.objects.get(id =
activity)
            except NeighborhoodLiving.DoesNotExist:
                activitycategory = None
            notification = Notification.objects.filter(nbliving =
nbliving, activitycategory = activitycategory, text = text, level = 1)
            notification.save()
            return render(request,
"suspicionmonitor/group_notification.html", {"key": key, "notification":
notification })

    def alert_notification_view(request):
        user = request.user
        users = Confidant.objects.filter(user_src =
user).values_list("user_dst")
        if user.active and users:

```

```

        psettings = getattr(settings, 'WEBPUSH_SETTINGS', {})
        key = psettings.get('VAPID_PUBLIC_KEY')
        notifications = []
        neighborhoods =
NeighborhoodLiving.objects.get(living__user = user, approved =
True).distinct('neighborhood')
        for user_to in users:
            notification =
PersonalNotification.objects.filter(user_src = user, user_dst =
user_to, text = 'Потрібна допомога!')
            notification.save()
            notifications.append(notification)
        users =
NeighborhoodLiving.objects.select_related("living").select_related("li
ving__user").filter(neighborhood__in =
neighborhoods).values_list("living__user")
        for user_to in users:
            notification =
PersonalNotification.objects.filter(user_src = user, user_dst =
user_to, text = 'Потрібна допомога!')
            notification.save()
            notifications.append(notification)
        return render(request, "suspicionmonitor/alert.html",
{"key": key, "notifications": notifications })

def add_regionneighborhood_view(request, region, neighborhood):
    user = request.user
    if user.active:
        try:
            neighborhood = Neighborhood.objects.get(id =
neighborhood)
        except Neighborhood.DoesNotExist:
            neighborhood = None
        try:
            region = Region.objects.get(id = region)
        except Region.DoesNotExist:
            region = None
        if neighborhood and region:
            approve =
NeighborhoodLiving.objects.filter(living__user = user, neighborhood =
neighborhood, manager_level = True, approved = True)
            if approve:
                now_region =
RegionNeighborhood.objects.filter(region = region, neighborhood =
neighborhood)

```

```

        if now_region:
            now_region.delete()
            redirect("index")
        else:
            now_region = RegionNeighborhood(region = region,
neighborhood = neighborhood, approved = False)
            now_region.save()
            redirect("index")

def neighborhood_view(request, neighborhood):
    user = request.user
    if user.active:
        try:
            neighborhood = Neighborhood.objects.get(id =
neighborhood)
        except Neighborhood.DoesNotExist:
            neighborhood = None
        neighborhoods =
NeighborhoodLiving.objects.select_related("neighborhood").select_relat
ed("living").select_related("living__user").filter(neighborhood =
neighborhood)
        return render(request,
"suspicionmonitor/neighborhood.html", {"neighborhoods": neighborhoods,
"neighborhood": neighborhood})

@require_POST
@csrf_exempt
def send_notification(request):
    try:
        notification_parameters = json.loads(request.body)
        user = User.objects.get(id =
notification_parameters['id'])
        ready_part = {'head': notification_parameters['head'],
'body': notification_parameters['body']}
        send_user_notification(user = user, payload =
ready_part, ttl = 1000)

def region_view(request, region):
    user = request.user
    if user.active:
        try:
            region = Region.objects.get(id = neighborhood)
        except Region.DoesNotExist:
            region = None

```

```

        regions =
RegionNeighborhood.objects.select_related("neighborhood").select_relat
ed("region").filter(region = region)
        return render(request, "suspicionmonitor/neighborhood.html",
{"regions": regions, "region": region })

def suspiciousactivity_view(request, suspiciousactivity):
    user = request.user
    if user.active:
        try:
            suspiciousactivity =
Suspiciousactivity.objects.prefetch_related("photos").prefetch_related
("severity").filter(id = suspiciousactivity).get()
        except Suspiciousactivity.DoesNotExist:
            suspiciousactivity = None
        versions =
Version.objects.select_related("living").filter(suspiciousactivity =
suspiciousactivity)
        return render(request,
"suspicionmonitor/suspiciousactivity.html", {"suspiciousactivity":
suspiciousactivity, "versions": versions})

def index_view(request):
    user = request.user
    users = [user]
    confidant = Confidant.objects.filter(user_dst = user,
activation = True).value("user_src")
    if confidant:
        users.append(confidant)
    if user.active:
        my_neighborhoods =
NeighborhoodLiving.objects.filter(living__user__in = users, approved =
True)

        my_neighborhoodlivings = my_neighborhoods.
distinct("neighborhood")
        my_regions =
RegionNeighborhood.objects.select_related("region").filter(neighborhoo
d__in = my_neighborhoods, approved = True).distinct("region")
        my_region_neighborhoods =
RegionNeighborhood.objects.select_related("neighborhood").filter(regio
n__in = my_regions, approved = True).distinct("neighborhood")
        suspiciousactivities =
Suspiciousactivity.objects.filter(nbliving__in =
my_neighborhoodlivings).order_by("-observation_date")

```

```

        region_suspiciousactivities =
Suspiciousactivity.objects.filter(nbliving__neighborhood__in =
my_region_neighborhoods).order_by("-observation_date")
        reports = Report.objects.filter(nbliving__in =
my_neighborhoodlivings).order_by("-event_date")
        region_reports =
Suspiciousactivity.objects.filter(nbliving__neighborhood__in =
my_region_neighborhoods).order_by("-event_date")
        notifications = Notification.objects.filter(nbliving__in
= my_neighborhoodlivings).order_by("-created_date")
        region_notifications =
Notification.objects.filter(nbliving__neighborhood__in =
my_region_neighborhoods).order_by("-created_date")
        return render(request, "suspicionmonitor/index.html",
{"neighborhoods": my_neighborhoods, "regions": my_regions,
"suspiciousactivities": suspiciousactivities,
"region_suspiciousactivities": region_suspiciousactivities, "reports":
reports, "region_reports": region_reports, "notifications":
notifications, "region_notifications": region_notifications })

```

```

def user_actions_view(request):
    user = request.user
    if user.active:
        suspiciousactivities =
Suspiciousactivity.objects.filter(nbliving__living__user =
user).order_by("-observation_date")
        reports = Report.objects.filter(nbliving__living__user =
user).order_by("-event_date")
        notifications =
Notification.objects.filter(nbliving__living__user = user).order_by("-
created_date")
        personal_notifications =
PersonalNotification.objects.filter(user_src = user).order_by("-
created_date")
        return render(request, "suspicionmonitor/history.html",
{"suspiciousactivities": suspiciousactivities, "reports": reports,
"notifications": notifications, "personal_notifications":
personal_notifications})

```

```

def statistics_view(request, neighborhood = None, region =
None):
    user = request.user
    if user.active:
        if neighborhood:
            try:

```

```

        neighborhoods = [Neighborhood.objects.get(id =
neighborhood)]
    except Neighborhood.DoesNotExist:
        neighborhoods = []
    if region:
        try:
            region = Region.objects.get(id = region)
        except Region.DoesNotExist:
            region = None
        if region:
            neighborhoods =
RegionNeighborhood.objects.filter(region = region, approved =
True).distinct("neighborhood")
            suspiciousactivities =
Suspiciousactivity.objects.select_related("activitycategory").filter(n
bliving__neighborhood__in = neighborhoods).order_by("-
observation_date")
        return render(request, "suspicionmonitor/statistics.html",
{"suspiciousactivities": suspiciousactivities })

```

ДОДАТОК В
Слайди презентації

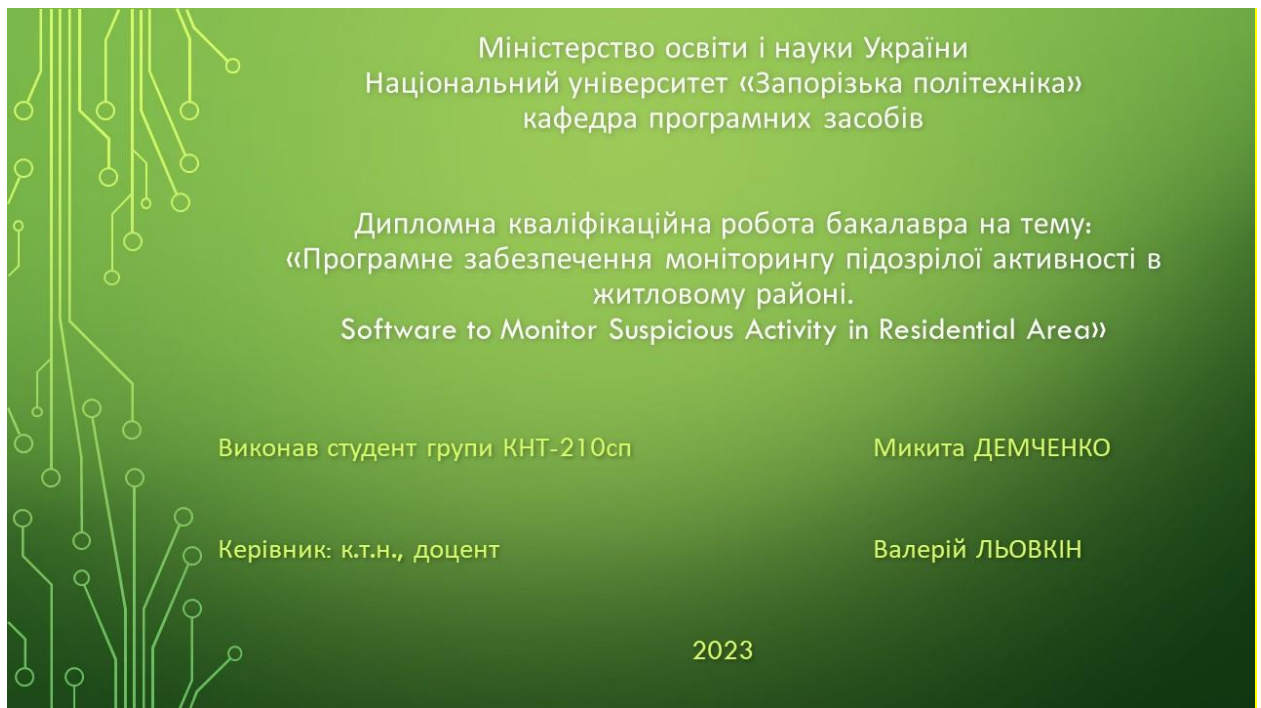


Рисунок В.1 – Початковий слайд

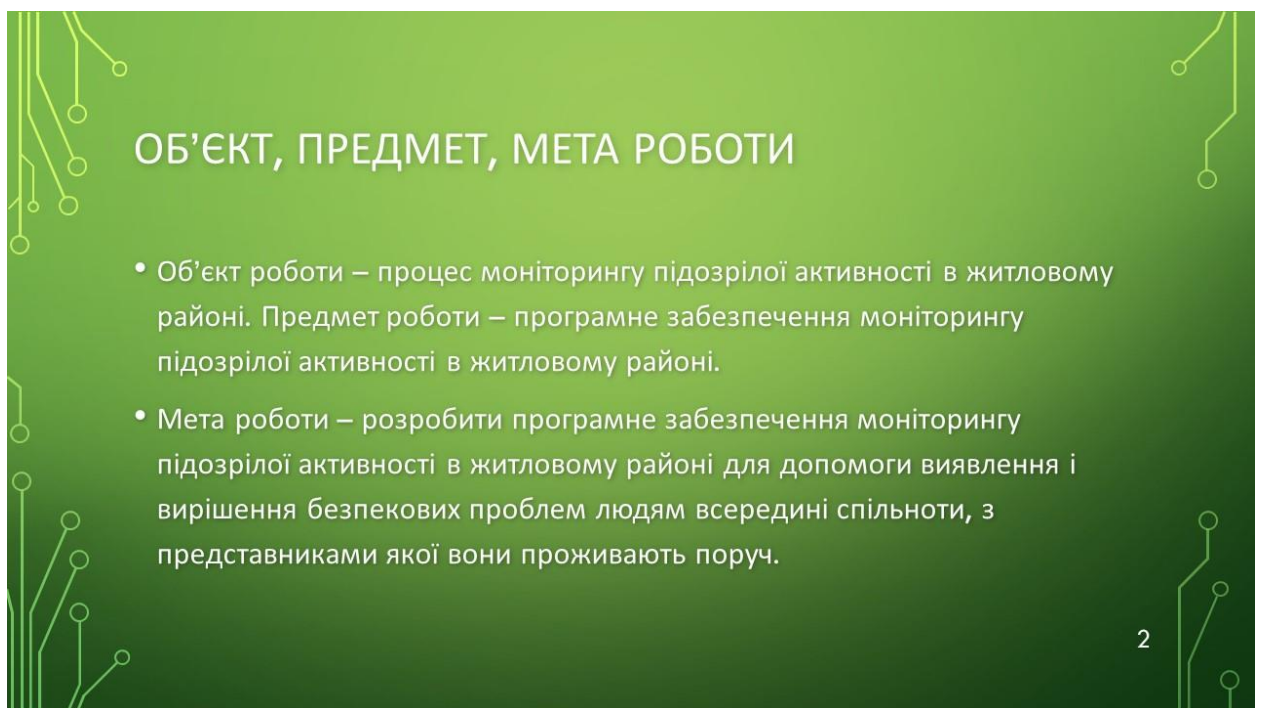


Рисунок В.2 – Мета роботи

ЗАВДАННЯ РОБОТИ

- аналіз предметної області моніторингу підозрілої активності в житловому районі;
- визначення вимог і особливостей програмного забезпечення моніторингу підозрілої активності в житловому районі;
- проектування програмного забезпечення моніторингу підозрілої активності в житловому районі;
- реалізація програмного забезпечення моніторингу підозрілої активності в житловому районі і тестування.

3

Рисунок В.3 – Завдання роботи

ПОРІВНЯННЯ РОЗРОБКИ З ПРОТОТИПОМ

Характеристика	MyLocalCrime	WeAlert	Розробка
Можливість зв'язатися з сусідами напряму	Ні	Так	Так
Сповіщення користувачів про ситуацію, яка сталась	Так	Так	Так
Визначення статистики кримінальних випадків, які вже трапилися	Так	Ні	Так
Підтримка і визначення близьких людей	Ні	Ні	Так
Взаємодія між створеними групами	Ні	Ні	Так

4

Рисунок В.4 – Порівняння розробки з прототипом

Рисунок В.5 – Порівняння вебфреймворків для розробки

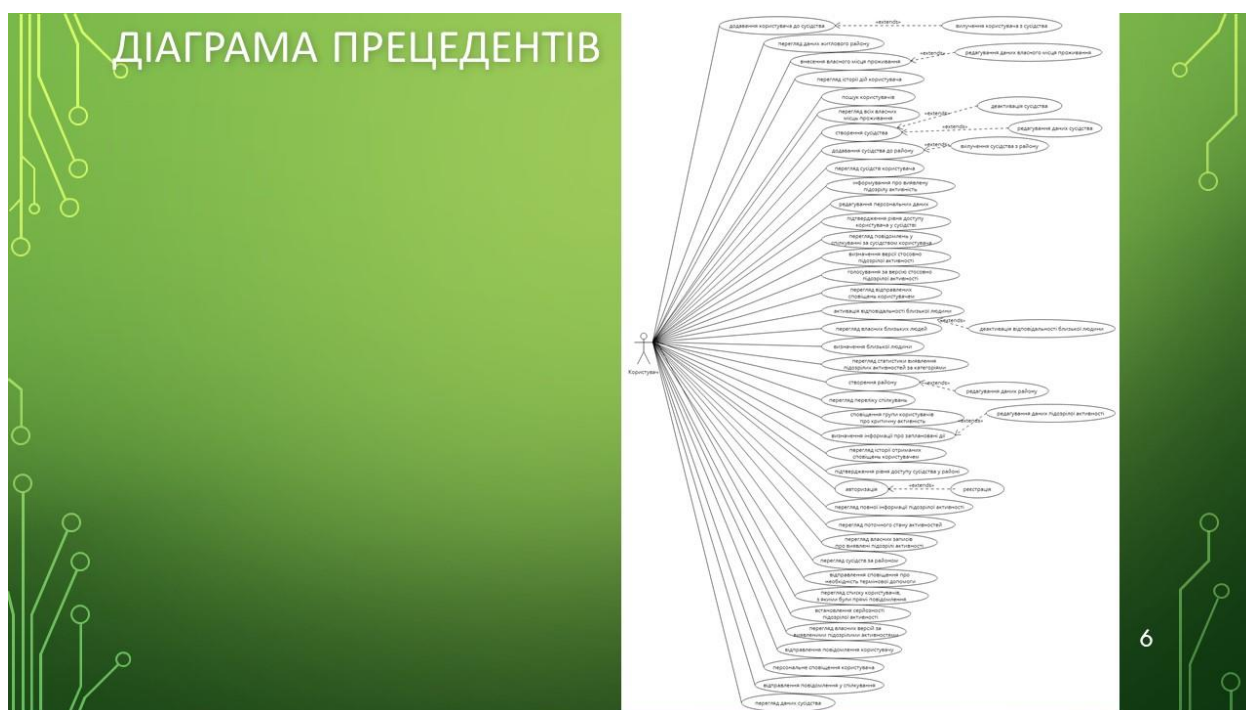


Рисунок В.6 – Діаграма прецедентів



Рисунок В.7 – Схема БД

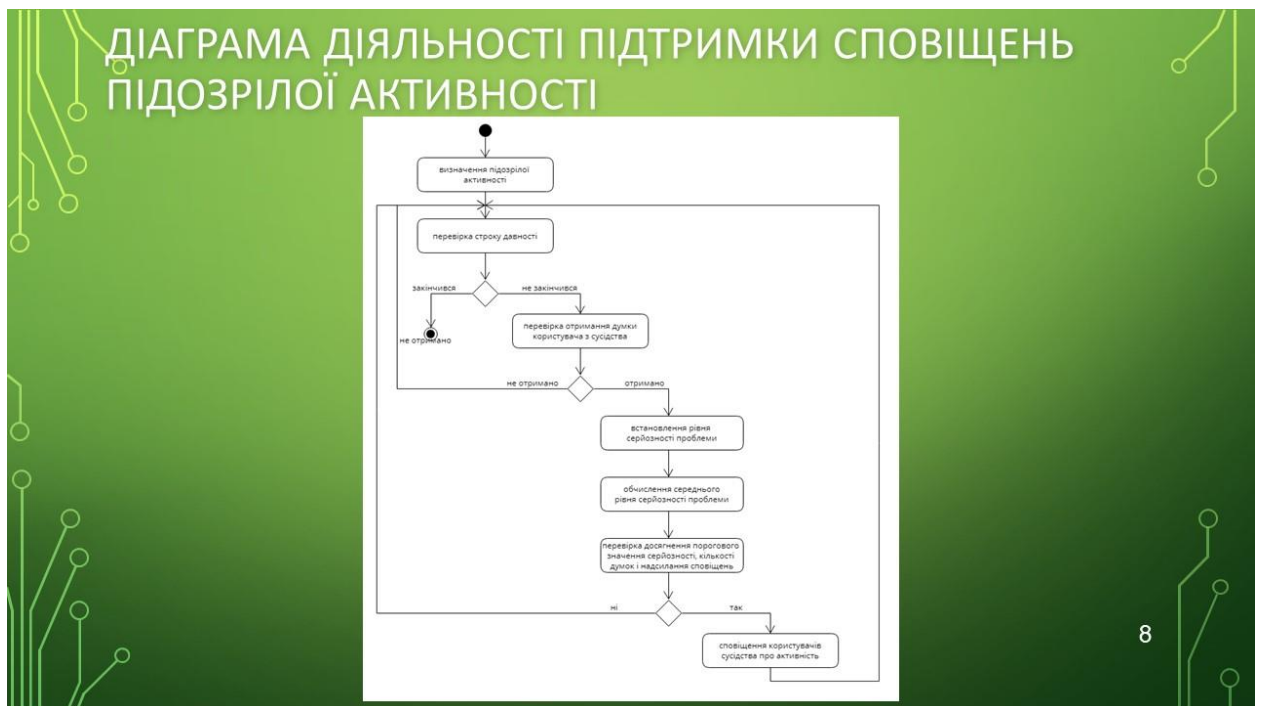


Рисунок В.8 – Діаграма діяльності підтримки сповіщень підозрливої активності

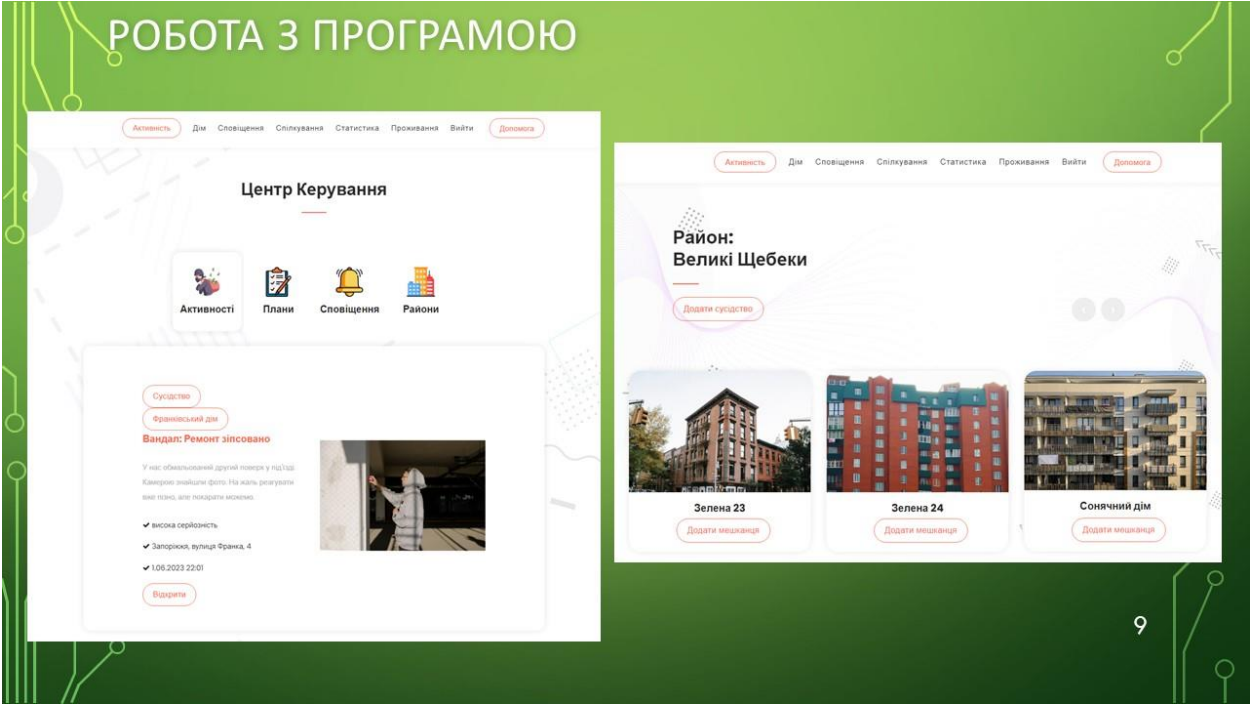


Рисунок В.9 – Робота з програмою

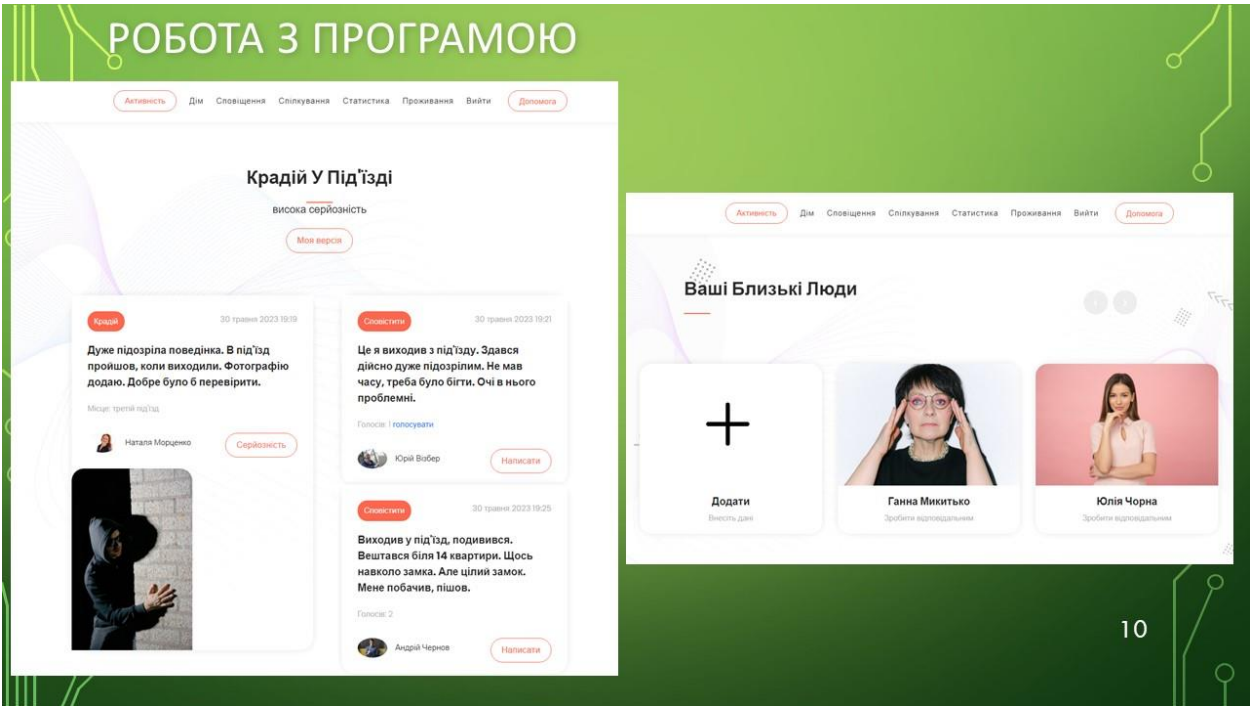


Рисунок В.10 – Робота з програмою

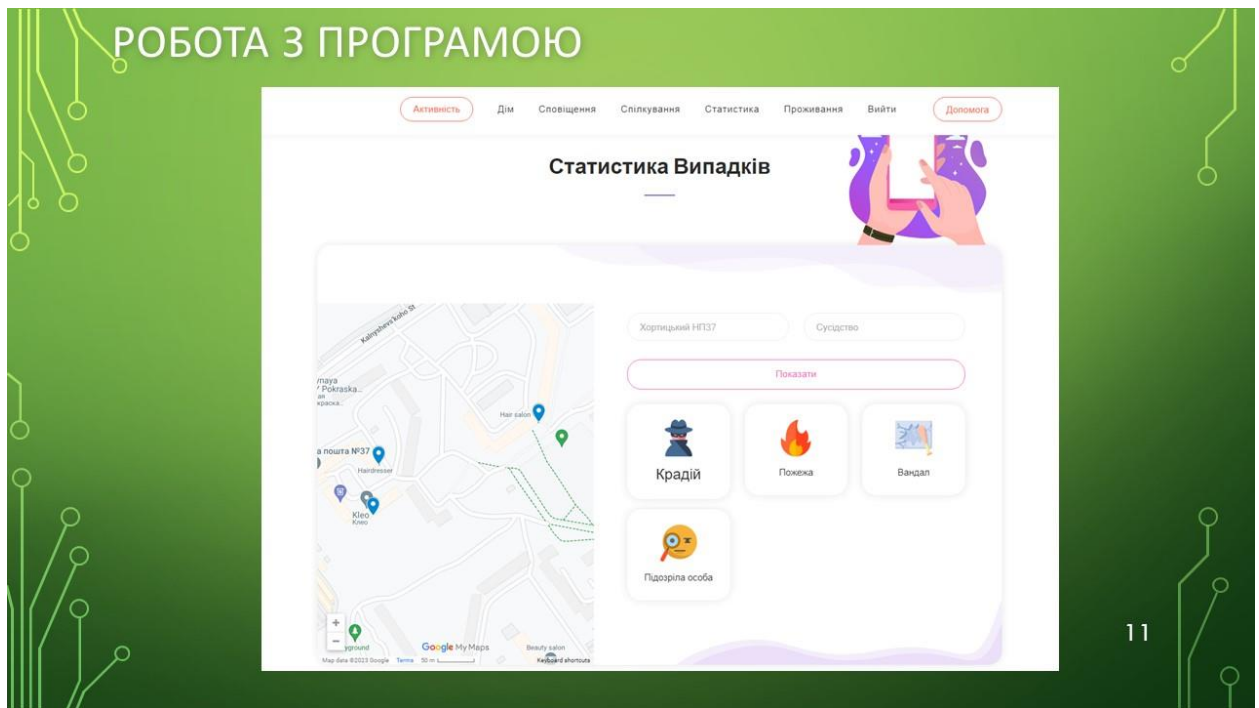
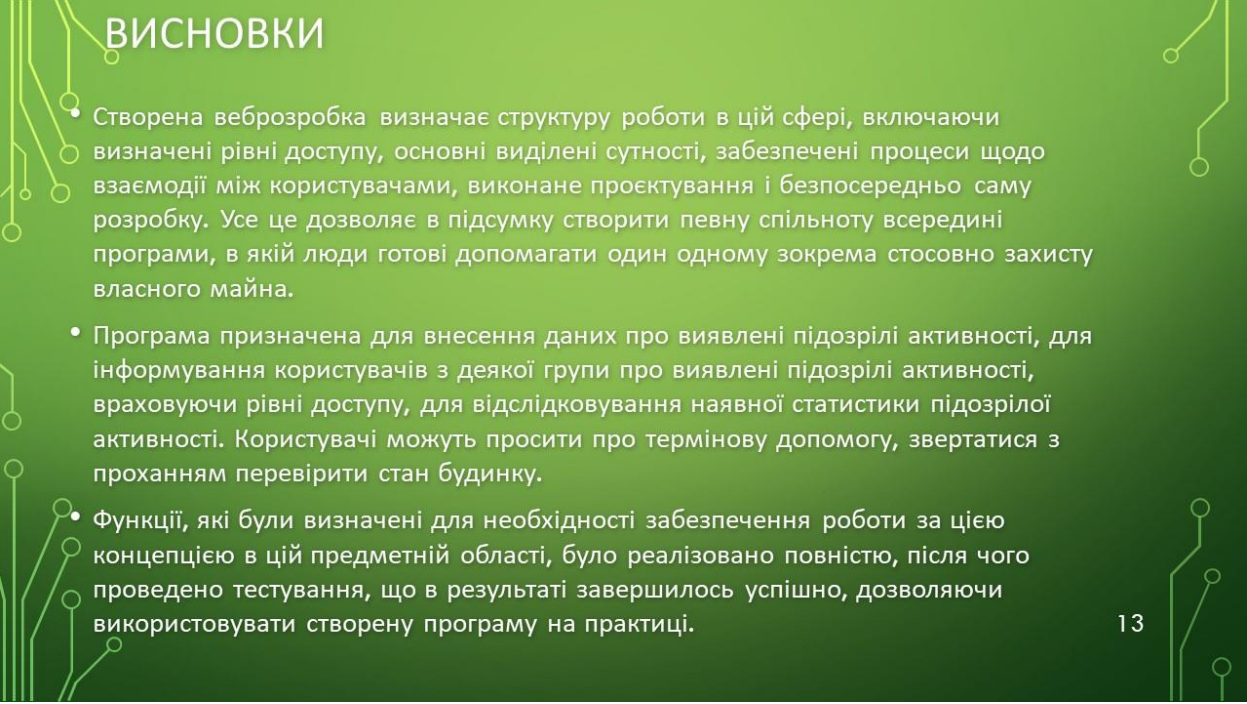


Рисунок В.11 – Робота з програмою

Сценарій		Підсумок	ТЕСТУВАННЯ ПРОГРАМИ		1	2
1	2		1	2	Додавання користувача до сусідства	+
Перегляд повної інформації підозрілої активності	+		Відправлення сповіщення про необхідність термінової допомоги	+	Вилучення користувача з сусідства	+
Перегляд власних записів про виявлені підозрілі активності	+		Сповіщення групи користувачів про критичну активність	+	Створення району	+
Перегляд сусідств за районом	+		Персональне сповіщення користувача	+	Редагування даних району	+
Внесення власного місця проживання	+		Перегляд відправлених сповіщень користувачем	+	Додавання сусідства до району	+
Створення сусідства	+		Перегляд історії отриманих сповіщень користувачем	+	Вилучення сусідства з району	+
Редагування даних сусідства	+		Визначення близької людини	+	Підтвердження рівня доступу користувача у сусідстві	+
Редагування даних власного місця проживання	+		Перегляд власних близьких людей	+	Підтвердження рівня доступу сусідства у районі	+
Редагування персональних даних користувача	+		Активація відповідальності близької людини	+	Перегляд сусідств користувача	+
Авторизація користувачів	+		Деактивація відповідальності близької людини	+	Перегляд історії дій користувача	+
Реєстрація користувачів	+		Перегляд статистики виявлення підозрілих активностей за категоріями	+	Пошук користувачів	+
Деактивація сусідства	+		Встановлення серйозності підозрілої активності	+	Інформування про виявлену підозрілу активність	+
Перегляд всіх власних місць проживання	+		Перегляд власних версій за виявленими підозрілими активностями	+	Редагування даних підозрілої активності	+
			Перегляд даних сусідства	+	Перегляд поточного стану активностей	+
					Перегляд даних житлового району	+
					Визначення версії стосовно підозрілої активності	+
					Голосування за версію стосовно підозрілої активності	+
					Перегляд повідомлень у спілкуванні за сусідством користувача	+
					Перегляд переліку спілкувань	+
					Відправлення повідомлення у спілкування	+
					Перегляд списку користувачів, з якими були прямі повідомлення	+
					Відправлення повідомлення користувачу з сусідства або району	+
					Визначення інформації про заплановані дії	+

Рисунок В.12 – Тестування програми



ВИСНОВКИ

- Створена веброзробка визначає структуру роботи в цій сфері, включаючи визначені рівні доступу, основні виділені сутності, забезпечені процеси щодо взаємодії між користувачами, виконане проектування і безпосередньо саму розробку. Усе це дозволяє в підсумку створити певну спільноту всередині програми, в якій люди готові допомагати один одному зокрема стосовно захисту власного майна.
- Програма призначена для внесення даних про виявлені підозрілі активності, для інформування користувачів з деякої групи про виявлені підозрілі активності, враховуючи рівні доступу, для відслідковування наявної статистики підозрілої активності. Користувачі можуть просити про термінову допомогу, звертатися з проханням перевірити стан будинку.
- Функції, які були визначені для необхідності забезпечення роботи за цією концепцією в цій предметній області, було реалізовано повністю, після чого проведено тестування, що в результаті завершилось успішно, дозволяючи використовувати створену програму на практиці.

13

Рисунок В.13 – Висновки за роботою