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

ВИПУСКНА РОБОТА

на тему:

«Веб-сервіс для громадського об'єднання із можливістю швидкого голосування та використанням картографічного API»

Завідувач
випускаючої кафедри **Довбиш А.С.**
Керівник роботи **Проценко О.Б.**
Студента групи ІН – 72 **Перехрестюк П.О.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедри Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ

до випускної роботи

Студентки четвертого курсу, групи ІН-72 спеціальності "Інформатика" денної форми навчання Перехрестюк Поліни Олегівни.

Тема: " Веб-сервіс для громадського об'єднання із можливістю швидкого голосування та використанням картографічного API "

Затверджена наказом по СумДУ

№_____ от _____2021 р.

Зміст пояснювальної записки: 1) аналітичний огляд методів зібрання даних з опитування населення; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, структур і критеріїв, що використовуються веб-сервісом для збору і обробки аналітичних даних; 5) розробка веб-сервісу для громадського об'єднання; 6) аналіз результатів.

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Проценко О.Б.

Завдання приняв до виконання _____ Перехрестюк П.О.

РЕФЕРАТ

Записка: 71 стор., 20 рис., 5 табл., 1 додаток, 19 джерел.

Об’єкт дослідження — алгоритм та методи збору аналітичних даних щодо впливу населення на планування забудови міст.

Мета роботи — розробка веб-сервісу для громадського об’єднання із можливістю швидкого голосування та використанням картографічного API для збору та обробки статистичних даних.

Методи дослідження — метод аналітично-статистичний.

Результати — розроблено метод та веб-сервіс для збору та обробки статистичних даних з метою дослідження впливу населення на планування забудови міст. Розроблений ресурс реалізовано у формі веб-додатку, створеного за допомогою мови програмування Python із використанням бібліотеки Django.

ВЕБ-СЕРВІС ДЛЯ ГРОМАДСЬКОГО ОБ’ЄДНАННЯ, ОБРОБКА СТАТИСТИЧНИХ ДАНИХ, МЕТОД АНАЛІТИЧНО-СТАТИСТИЧНИЙ, ШВИДКЕ ГОЛОСУВАННЯ, КАРТОГРАФІЧНЕ API.

ЗМІСТ

ВСТУП 5

1 РОЗГЛЯД ТЕМИ ТА ІСНУЮЧИХ РІШЕНЬ 6

1.1 Огляд проблемної області 6

1.2 Методи досліджень. Огляд подібних рішень 8

1.3 Аналіз та результати. Система класифікації 10

1.4 Постановка задачі 16

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ 18

2.1 Вибір засобів програмування 18

2.2 Система управління базою даних 26

2.3 Проектування інформаційної системи 28

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ 30

3.1 Зберігання зображень 33

3.2 Авторизація 35

3.3 Реалізація карти 36

3.4 Математичний опис моделі розрахунку радіусу 41

3.5 Використання програмного додатку 41

ВИСНОВКИ 47

Список літератури 48

ДОДАТОК 50

ВСТУП

Сьогодні існує проблема швидкої комунікації громадського населення із місцевими органами влади. Досить часто доводиться чекати на відповідь більше місяця, а до цього необхідно зібрати підписи – це також займає багато часу. Також наразі збір інформації від користувачів або звичайних жителів міста є досить проблематичною роботою, адже багато хто не має часу для написання змістовних відгуків. Часто міська влада або громадські об’єднання не знають або не помічають на перший погляд незначних, але нагальних проблем міста та потреб людей щодо його благоустрою.

Тому ця робота присвячена розробці веб-сервісу, за допомогою якого користувачі мають змогу без особливих зусиль голосувати за ті місця або об’єкти в місті, які потребують уваги місцевої влади, отже це значно прискорить вирішення питань.

Зібрані дані будуть оброблятися та зберігатися задля подальшої реконструкції міста місцевими органами влади. Крім того, отримані дані будуть відображатися у вигляді інтерактивної карти в інтеграції з картографічним API. Це означає, що актуальна інфографіка із використанням картографічного API може привернути увагу гостей міста та його жителів до цікавих споруд, про які не кожен може здогадуватися або ж попередити про деякі незручності в місті.

У цій роботі будуть дані відповіді на два важливих запитання: 1. Який вплив має діджиталізація на роль громадян у забудові, благоустрою міста та процес прийняття рішень щодо вищевказаних питань? 2. Які переваги та недоліки у залученні містян до процесу благоустрою міста за допомогою інформаційно-комунікаційних технологій?

1 РОЗГЛЯД ТЕМИ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд проблемної області

Сьогодні існує проблема стрімкого розвитку містобудування в Україні. Основними причинами цього є швидкий ріст міст внаслідок переселення людей із селищ в міста з ціллю пошуку більш високооплачуваної роботи і взагалі кращого життя, зменшення кількості природних ресурсів та прагнення людей до екологічного, стабільного та більш планомірного розвитку.

Містяни зазвичай досить нечасто залучаються до процесу планування та конструкції міських ландшафтів та вуличних просторів, але даремно. Знання звичайних пересічних громадян, які вони здобувають проживаючи в місті, користуючись об’єктами інфраструктури, можна вважати одними із найцінніших і ними безсумнівно не можна нехтувати.

Важливим етапом в боротьбі із цими недоліками є залучення технологічного прогресу. Сучасні інформаційні технології дають багато можливостей для оновлення та повного переформування процесу ще на етапі планування. Більш того, вони дозволяють оновити і тим самим покращити «спілкування» між забудовниками та жителями міста. Адже саме містяни є основною фігурою та цільовою групою в проведенні цього величезного і у той же час такого делікатного процесу.

Насамперед потрібно вказати те, що наразі у нашому місті та і взагалі в Україні є проблемою формування «розумного» міста, де всі об’єкти або хоча б більшість з них оптимізовані з точки зору урбаністики. Часто думкою більшості населення нехтують, вважаючи її некомпетентною та не вартою уваги. Тобто офіційно та загалом пересічний громадянин вважається центральною фігурою, насправді ж він грає периферійну роль. Проте, як і було згадано раніше, звичайні жителі міст мають грати та по суті грають ключову роль у генерації та обміні корисних даних про їх повсякденні проблеми пов’язані з оточуючим загальнодоступним середовищем.

Деякі дослідники вважають, що проекти «розумного» середовища не несуть у собі глобального значення та не мають привертати соціальної уваги. Вони вказують на необхідність перенесення фокусу з бачення, яке орієнтоване на цифровізацію, на людське бачення. І хоча з кожним роком технологічні інновації збільшуються, частка населення, яка має доступ до швидкісної інтернет мережі, яка допомагає їм розвиватися та робити більш компетентні рішення, росте, статистичні дані, які отримуються від них, нечасто використовуються взагалі для містобудування.

Все ж таки переваги участі населення у плануванні забудови міста були не раз доведені. Дослідники Еванс-Коулі та Холландер [8] пишуть, що участь містян у цьому процесі допомагає їм краще прийняти зміни у політиці планування та деяких проектів. «Без активної участі громадянина – прогресивні розумні міста не можуть існувати»[9, с.2]. Участь у плануванні має глобальне та вирішальне значення для розвитку комфортних міст, так як саме це дозволяє взяти до уваги та навіть включити у проект реальні потреби та практики пересічних мешканців. Більш того, серйозніше дотримання цілеспрямованих та планомірних дій допомагає досягти реалізації більш вагомих проектів [11].

Планування забудови міст – це непростий процес, у якому є багато складових, у тому числі, як уже було згадано раніше, звичайні жителі цих міст. Тому дуже важливо знайти способи як можна заохотити та підтримати взаємодію між містянами, органами влади та технологічними засобами [12]. Натомість, численні проблеми заважають традиційній участі, а саме: необхідність знаходитись у конкретному місці та в конкретний час, обмежується кількість учасників, час на проведення та тип самим збільшуються витрати. Найголовніше – це відсутність мотивації у самих учасників [13].

З поширенням використання смартфонів, розповсюдженням дешевого мобільного інтернету і тим самим використанням соціальних мереж люди

мають можливість легко висловлювати свої думки та дізнаватись про інші думки. Участь у плануванні тепер стала навіть більш актуальною, так як невдоволення широкої маси людей вже стає більш розповсюдженим і як результат – швидка реакція зі сторони влади. Можна сказати, що сучасні технології – це новий та ефективний спосіб подолання недоліків у процесах планування забудови міст.

Роботи, які віднесені до цієї теми, зазвичай просто визначали та описували лише інструменти, але не надавали конкретні можливості та способи для продукування даних, які можна використати надалі. Більш того, саме ці статті, як правило, більше зосереджуються на самих інструментах, ніж на результатах дослідження та перевагах залучення населення. Тобто дуже важливо зрозуміти наслідки впливу саме цих інструментів на участь у плануванні, аби можна було у подальшому скористатися їх перевагами для вдосконалення містобудування та мати на увазі недоліки, щоб запобігти небажаних помилок.

Ціллю цього дослідження є розкриття потенціалу цифровізації інструментів збору інформації, які спрямовані на розширення можливостей планування забудови міст, а також зрозуміти переваги і недоліки цього процесу.

1.2 Методи досліджень. Огляд подібних рішень

Метою цього дослідження є визначення як цифрові інструменти можна використовувати в процесах генерації та обміну даними для подальшого планування забудови міста. Для цього було вирішено порівняти якомога близьке до мого проекту рішення – міський сайт для розгляду петицій. Цей інструмент включає в собі веб-платформу для створення, голосування та розгляду петицій – питань, які турбують населення міста (Рис. 1.1 та Рис 1.2).

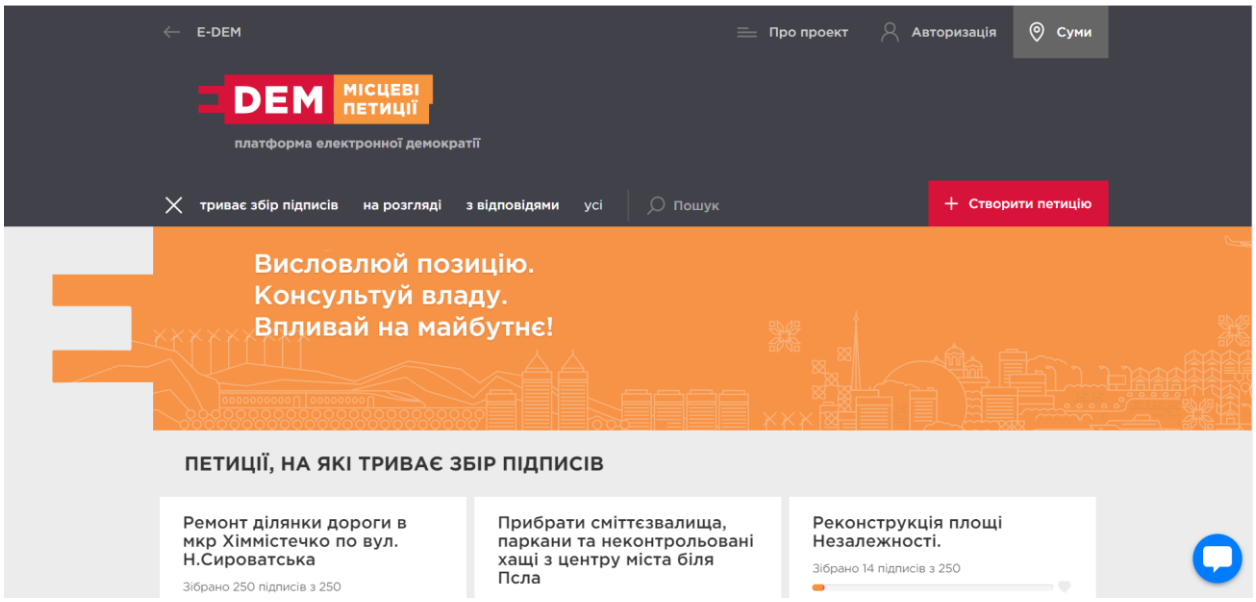

Рисунок 1.1 – Перша сторінка платформи для збору петицій (e-dem.ua)

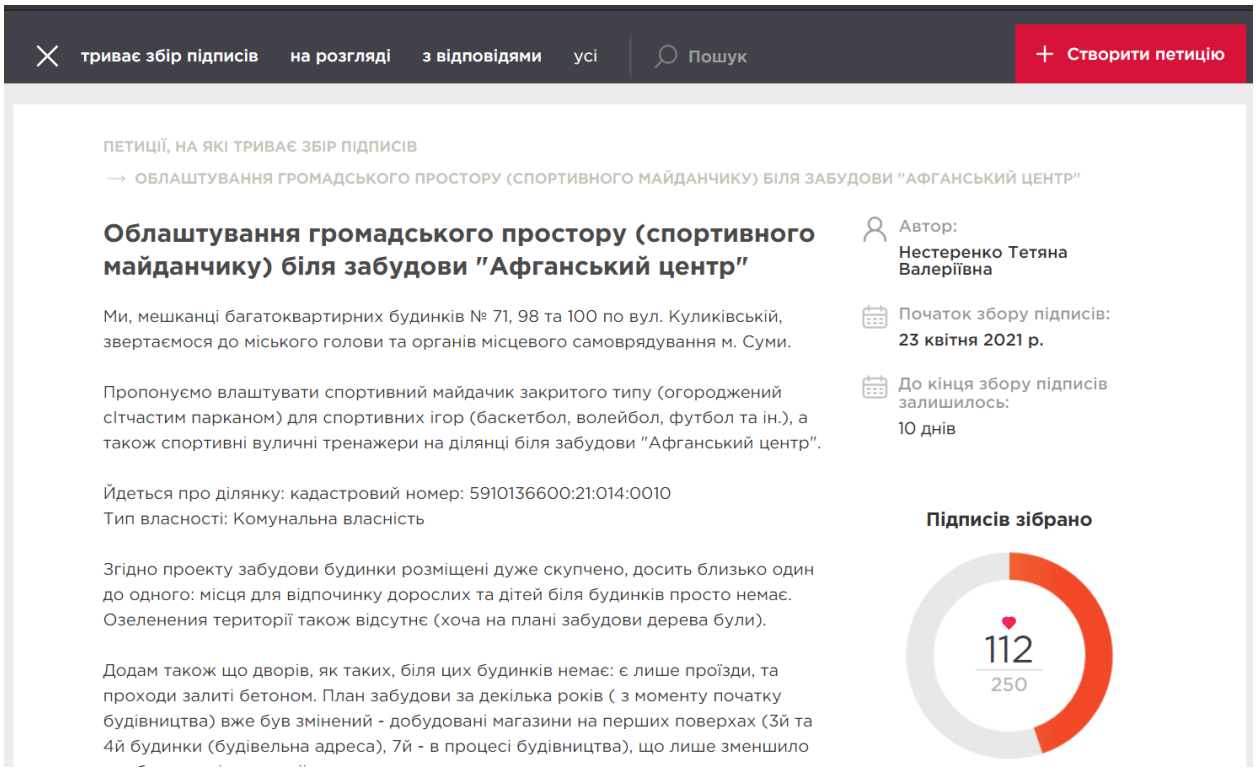

Рисунок 1.2 – Вигляд петиції (e-dem.ua)

Цей проект є чудовим прикладом того, що сучасна людина віддає перевагу використанню інформаційних засобів задля досягнення певної мети, а саме – аби її думка чи запит були почуті та взяті до уваги місцевими органами влади. Глобальна ідея схожа на мою: залучити якомога більше людей за допомогою соціальних мереж та нагальних питань, щоб досягти покращення

міста. Але хотілось би виділити переваги та недоліки саме місцевої системи розгляду петицій. Можу назвати такі переваги цього проекту:

- можливість додавати детальний опис наведеної проблеми;
- можливість додавати більше, ніж одне фото;
- можливість користувача побачити петиції, за які він проголосував, у своєму власному кабінеті.

На мою думку, недоліками цієї системи є:

- необхідно багато часу, аби зібрати підписи;
- якщо підписи зібрані, то це все одно не означає, що питання буде вирішене;
- необхідно знову багато часу на очікування відповіді від місцевих органів влади;
- потрібно приділити час, аби прочитати та освідомити текст та основну ідею проблеми в петиції.

Я намагалася викласти всі, на мою думку, переваги та недоліки даної системи. Основним негативним елементом збору петицій є витрачення великої кількості часу. Натомість, однією з цілей мого проекту є досягнення максимальної швидкості у зібранні думки громади, аби людина не витрачала марно час та простим голосуванням можна було б отримати такий же точний результат.

Науковці кажуть, що сучасній людині необхідно 2 секунди часу, аби оцінити важливість та зацікавленість у тому, що він/вона бачить. Було опитано одинадцятьох людей зі свого оточення та майже всі (9/11) вони погодились, що віддали б перевагу швидкому та чіткому прийняттю рішень та були б раді приймати участь у плануванні благоустрою міста.

1.3 Аналіз та результати. Система класифікації

Якщо говорити про можливості кожної людини або певної групи населення в опитуванні та впливу на планування, то вони дещо відрізняються.

Для того, щоб зрозуміти відмінності між цими можливостями було розроблено концептуальну таблицю для класифікації за різними типами участі. Вона сформована на взаємодії між містянином (учасником) та міською владою. Це означає, що чим вищий рівень учасника, тим більше буде впливу та обміну даними та проблемами. На жаль, дана класифікація не відображає всю повноту різноманітності елементів, так як це лише загальна схема для передачі мого бачення. Отже, на двох найнижчих ланках таблиці («від людини до влади» і «від влади до людей») взаємодія відбувається в односторонньому порядку. Рівень «консультування» описує відповіді учасників на питання від міської влади за допомогою онлайн опитування. «Співробітництво» описує використання різних веб-платформ або мобільних додатків для висловлювання свого відношення до різних питань, які стосуються планування міст. Учасники зазвичай не обмежені в типах питань, тобто можуть задавати як закриті, так і відкриті питання тим самим обмінюючись проектами та пропозиціями з органами міського врядування. На рівні «взаємної співпраці» громадяни мають можливість вільно ділитися своїми ідеями та взаємодіяти зі всіма учасниками процесу. На найвищому рівні «наділення владою» громадяни передають органам влади можливість запровадити та втілити в життя всі ті ідеї, проекти та прохання, які були спочатку висловлені, а потім обговорені на попередніх рівнях. «Пасивне створення питань» – це коли люди діляться якоюсь інформацію, коли вмикають локацію на своєму телефоні, також за допомогою мобільних додатків іноді ведеться збір даних і люди просто не знають, що певна інформація використовується для аналітичних даних. Пасивні дані – це корисне джерело інформації про соціальне життя та структуру міста та вони є відіграють неабияку роль у формування динаміки міста у реальному часі. «Активне опрацювання питань» – це навпаки, коли люди свідомо діляться своїми ідеями у спеціальних сервісах [6].

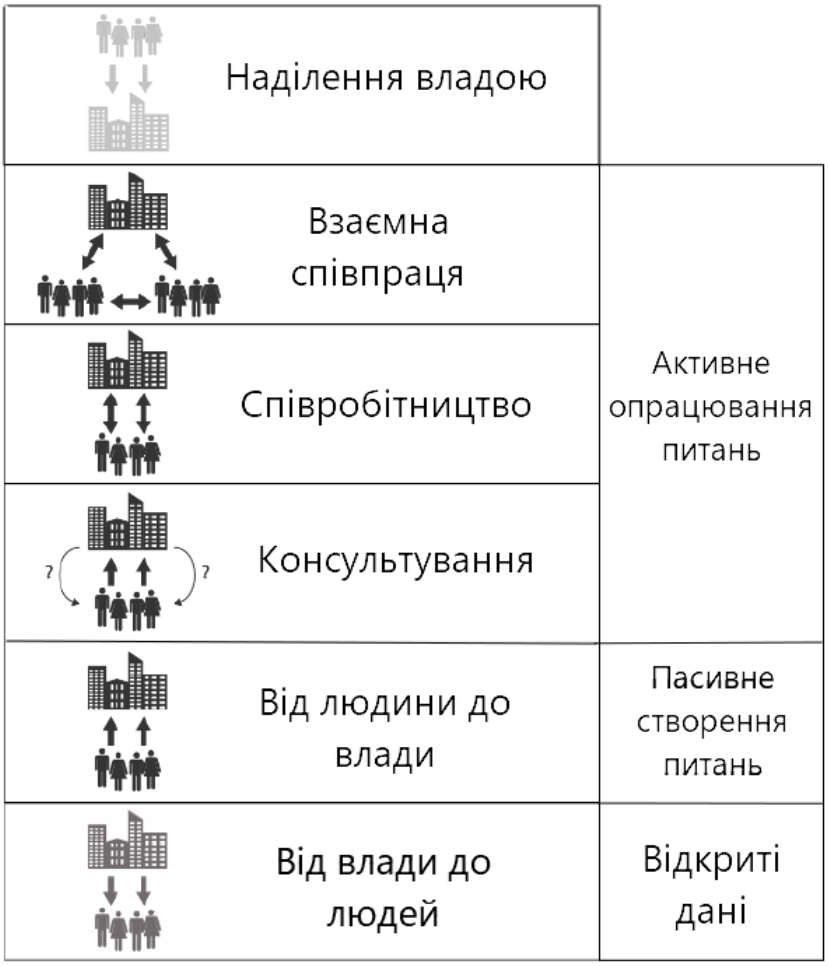

Рисунок 1.3 - Концептуальна таблиця

Основна відмінність між найнижчою та найвищою ланкою є саме те, що інформація йде ніби знизу вгору, включаючи інформацію, яку міська влада не дізнається спеціально. Тобто це невідривно пов’язано з цифровими технологіями.

Переваги цифрової участі

Веб-платформи та сервіси, де люди можуть приймати участь у голосуванні, розширюють способи обміну даними. Дослідження Ейзенштейна та Бульє [14, 15] показало, що цифрові сервіси збільшують кількість охочуючих висловити свою думку і зробити свій внесок у формування певних ідей. Іноді це дозволяє знайти нові соціальні профілі, такі як молоді сім’ї або похилі люди. Таким чином можна залучити більше думок і врахувати більше нюансів. Тим паче, цифрові технології надають доступ до більш різноманітних типів даних. Додатковою перевагою є те, що можна збирати дані, які було

важко зібрати: це може бути сприйняття інформації в реальному часі, думок чи ідей. Ці дані, які зібрані цифровими методами, по суті орієнтовані на громадськість і тим самим можуть створювати нові або уточнювати уявлення про місто. Наприклад, можна не рахувати кількість автомобілів в місті, які проїжджають у певний час у певному місці, а можна подивитися як люди рухаються в якомусь районі в конкретний момент в режимі реального часу. Більш того, є можливість навіть визначити вид транспорту, що використовується, аналізуючи дані з мобільних телефонів, а саме швидкість руху і так далі, таким чином отримуючи точні дані.

Рисунок 1.4 – Знімки екрана з мобільного додатку Unlimited Cities (джерело: додаток Unlimited Cities)

Деякі цифрові інструменти, які використовуються для участі планування забудови міст, можуть імітувати реальні дії, то можна визначити нові варіанти участі для громади. Ці нові варіанти дозволяють учасникам більше зануритися у процес планування. Мається на увазі цифрові інструменти, які використовують доповнену реальність. Гарним прикладом може бути мобільний додаток Unlimited Cities [16] (рис. 1.4) був створений на ідеї доповненої реальності. За допомогою цього додатку учасники можуть

створити новий фото-реалістичний вид міста, налаштовуючи різні параметри такі як густота, озеленіння, колір, гнучкість і так далі. Інструменти цифрової участі безпосередньо змінюють роль громадянина у процесі планування – мешканці перестають бути пасивними і стають активними учасниками [17, 18]. Зараз є прямий доступ до відкритих даних від активістів або місцевої влади в інтернеті, тому люди без проблем можуть пропонувати свої ідеї, подаючи пропозиції та обговорюючи їх (Рис. 1.5). Варто зазначити, що хоча громадяни сьогодні все частіше беруть участь в обміні ідеями, думками та знаннями, вплив на містобудування все ще важко виміряти, на жаль, воно просто не визначено остаточно.

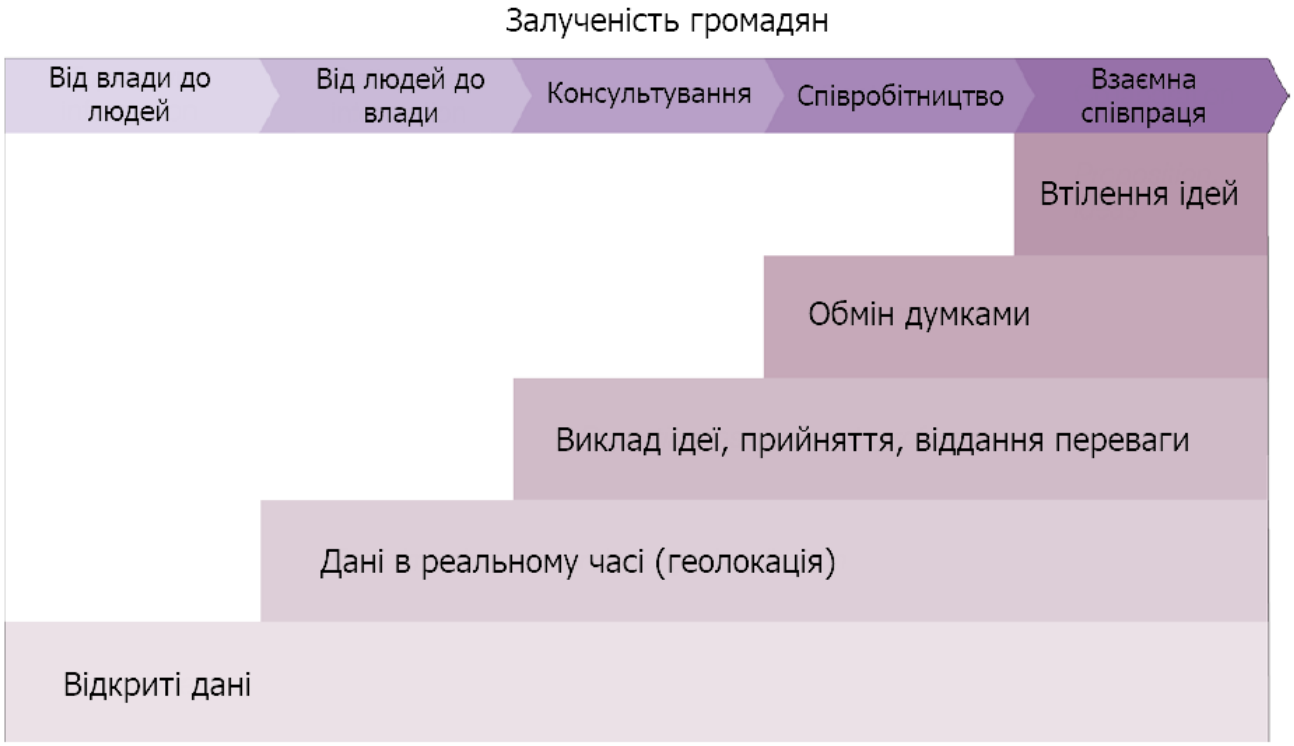

Рисунок 1.5 – Зміна ролі громадянина з отриманням нових міських даних

Недоліки цифрової участі

Величезне різноманіття внесених даних від містян збільшує складність їх обробки. Учасники додають та пропонують занадто багато різнотипних даних за допомогою цифрових інструментів, що виникає питання, яку ж саме інформацію слід брати до уваги та які саме дані є найбільш актуальними. Оскільки зазвичай немає можливості провести дебати чи навіть прості

обговорення, то важко визначити пріоритетні питання, особливо з огляду на те, що думки громадян можуть бути суперечливими. Також надзвичайно важливо враховувати рівномірно внесок громадян і вклад експертів. Системи голосування або зворотного зв'язку можуть виділити найбільш популярні або суперечливі проблеми. Бульє та Лохард [19] пропонують спосіб об'єктивізації суб'єктивних даних шляхом кодифікації та експертної оцінки. Однак ми все одно повинні знайти способи проаналізувати ці дані та перетворити внески у значущі знання.

Існує переріз населення в плануванні містобудування, який ніколи не є репрезентативним. Впровадження цифрової практики ускладнює цю ситуацію, оскільки часто на подібних сервісах відсутня формальна ідентифікація людей за віком, статтю, сімейним станом і так далі. Навіть профілі в соціальних мережах можуть сильно відрізнятися від «профілів» у реальному житті і, отже, відбувається зміна парадигми. І наостанок питання конфіденційності є найважливішим аспектом цифрової участі. Особисті дані слід захищати – це безумовне питання сьогодення. Отже, міська влада повинна бути прозорою щодо того, як використовуються та будуть використовуватися зібрані дані, і хто має доступ до цих даних. Крім того, важливо забезпечити громадянам можливість анонімної участі для забезпечення конфіденційності. Таким чином, методи анонімності даних можуть допомогти захистити конфіденційність учасників [6].

Підсумок аналізу

Беручи до уваги кількість платформ та мобільних додатків, очевидно, що цифрове планування та безпосередня участь громади у плануванні забудови міст є надзвичайно перспективним напрямком для збору даних. Однак недоліки, які були зазначені у цьому аналізі, вказують на необхідність переосмислення всієї системи планування, починаючи від спілкування та обміну інформацією між зацікавленими сторонами до навичок містобудівників. Однак цифрові інструменти не повинні замінювати інші

практики. Натомість їх слід розглядати як новий рівень знань, який можна використовувати для додаткового інформування планування та забудови міст на кожній фазі процесу. Отже, поєднання безлічі методів та інструментів є необхідним для досягнення планування міського простору, орієнтованого на факти. Однак все ще існує значний розрив між внесенням, обробкою та ефективним використанням даних, які необхідно врахувати, щоб сприяти реальним змінам та сприяти інклюзивним, обґрунтованим процесам планування.

Цей аналіз описує можливості виробництва та збору даних, а також те, як дані в даний час використовуються в процесах планування. Аналіз починається з пояснення того, як цифрові інструменти дозволяють громадянам відігравати нову роль у формуванні навколишнього середовища. Потім підкреслюється, як участь у цифрових технологіях може розширити можливості громадян та покращити процеси містобудування. Крім того, цей аналіз показує, що цифрові інструменти створюють нові перспективи планування, розглядаючи міські процеси на основі даних, створених користувачами. Нарешті, під час аналізу всіх даних вдалося визначити певні обмеження, які заважають моделі розумного міста рухатися до більш інформованого, всеохоплюючого планування, а отже, і гнучкого міста. Наразі необхідно проводити додаткові дослідження цього питання, щоб зрозуміти, як ці інструменти можуть бути включені до існуючих процесів планування, і як отримані та зібрані дані можуть ефективно стати частиною процесу. Важливо також визначити початкові етапи та основні моменти для зміни поточної практики, щоб повністю інтегрувати нові форми участі.

1.4 Постановка задачі

Метою роботи є розробка веб-сервісу, який матиме візуальну та серверну частину (Front-end та Back-end) із необхідним функціоналом – користувач відповідає на поставлене запитання, яке відноситься до планування, архітектури та комфортного проживання в місті Суми. Перед тим,

як відповісти на питання користувач обирає своє місцезнаходження на карті і за допомогою розрахунку радіусу йому будуть показані лише конкретні картки, які розташовані у цьому радіусі. Також він може передивитися карту із мітками цих карток із різними кольорами в залежності від рейтингу картки.

Наступні сторінки мають бути присутні у веб-сервісі:

- початкова – вітає користувачів та пропонує пройти авторизацію;
- інструкція – сторінка з поясненням про те, як користуватися сервісом;
- карта – сторінка з картою, де користувач має обрати своє місцезнаходження, якщо він не обрав його вже раніше;
- основна – сторінка з картками, де є фото та питання до користувача;
- додаткова – сторінка, де повідомляється, що картки закінчились, користувач може перейти до основної карти або змінити місцезнаходження;
- фінальна – на цій сторінці розташована карта із мітками та рейтинг основних питань;
- панель адміністратора – для роботи з БД.

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

2.1 Вибір засобів програмування

Мова програмування – це початковий та один із найважливіших етапів планування та розробки веб-додатку. Сьогодні існує велика кількість мов програмування, всі вони відрізняються одна від одної в першу чергу призначенням. Перед початком роботи необхідно чітко зрозуміти ціль та завдання, які стоять перед вами. Враховуючи те, що моя робота складається з візуальної частини та серверної, було вирішено обрати наступні мови програмування:

- Python;
- JavaScript.

Необхідно також пам’ятати, що сучасне програмування не складається лише з одних мов програмування. Важливу роль також відіграють фреймворки, бібліотеки та формальні мови для розмітки сторінок та опису зовнішнього вигляду документа. Загалом вони значно покращують та прискорюють процес розробки та і взагалі, сьогодні навіть важко уявити будь-який проект без використання додаткових модулів. Тому слід зазначити саме ці засоби, які були використані під час роботи:

- CSS;
- HTML;
- Django;
- jQuery.

Коли було обрано мови програмування, додаткові фреймворки та бібліотеки, необхідно було визначитись з IDE, яка могла б коректно інтерпретувати їх та взагалі мала б зрозумілий для користування вигляд. IDE – Integrated Development Environment – Інтегроване середовище розробки. Було вирішено обрати Visual Studio Code.

Visual Studio Code – це легкий і водночас багатофункціональний редактор коду, який запускається на будь-якій платформі, включаючи Windows, macOS та Linux, отже, можна одразу братися до роботи незважаючи на платформу. Також він вже підтримує мову JavaScript і має величезний перелік розширень, які можна встановити, для інших мов (C++, C#, Java, Python, PHP) та найбільш популярних середовищ запуску, наприклад .NET та Unity [1].

У своїй основі Visual Studio Code має дуже вдалий редактор коду і він неодмінно підійде для повсякденного використання. Завдяки підтримці безлічі мов програмування, VS Code допомагає миттєво підвищити продуктивність, а саме він може без проблем виділяти синтаксис, автоматично проставляти дужки, там також вбудована можливість налаштування автоматичного відступу, одночасного виділення багатьох вікон, фрагментів і так далі. Комбінації клавіш зрозумілі навіть початківцям, до того ж налаштовувати їх досить легко та зрозуміло, а відображення комбінацій клавіш дозволяють без зайвих проблем орієнтуватися в коді.

Варто зазначити, що ця IDE підтримує заповнення коду на базі IntelliSense, що включає розширену інтерпретацію семантичного коду, навігацію, а також рефакторинг. Коли написання коду стає складним, збільшується кількість файлів та разом із тим кількість помилок то цей код необхідно налагоджувати. Зазвичай налагодження не є приємною задачею, тому що необхідно передивитися багато сторінок коду. Саме тому Visual Studio Code включає інтерактивний налагоджувач, отже є можливість швидко продивлятися вихідний код, перевіряти змінні, переглядати списки викликів і обробляти команди через консоль розробників.

Під час виконання проекту був забезпечений доступ до репозиторію Git для того, щоб можна було слідкувати за розробкою та у разі виникнення непередбачуваних проблем пов’язаних зі збоєм можна було з легкістю відновити всі дані та продовжити роботу із проектом. Саме тому було і обрано

VS Code, бо він має підтримку Git, і можна працювати з проектом не виходячи з редактора, завантажувати нові зміни та відслідковувати свій прогрес.

Якщо говорити про архітектурний план, то Visual Studio Code поєднує в собі найкращі веб-технології. Завдяки використанню Electron, він поєднує в собі сучасні та все широко розповсюджені технології, як JavaScript та Node.js. Більш того, Visual Code використовує корисну архітектуру служб інструментів, а це дозволяє додавати її до багатьох технологій, що забезпечують прекрасну роботу Visual Studio, а саме технологію Roslyn, TypeScript та вдалий механізм налагодження Visual Studio.

Важливо зазначити, що Visual Studio Code вміщує в собі загальновідомий принцип масштабування, який дозволяє розробникам використовувати розширення та збагачувати свої навички з редагування, створення та налагодження додатків.

Чому було обрано Python? Популярність цієї мови програмування росте з кожним роком. Воно і не дивно, адже це одна із найпростіших мов програмування для початківців. Якщо почати вивчати її, то досить легко можна перейти до більш складних задач. Також це досить потужний інструмент для створення веб-додатків та автоматизації рутинних процесів. Нижче наведено лише деякі найпоширеніші поля, де Python знайшов своє використання:

- Наука про дані;
- Науково-математичні обчислення;
- Веб-розробка;
- Фінанси та торгівля;
- Автоматизація та адміністрування системи;
- Комп’ютерна графіка;
- Базова розробка ігор;
- Тестування на безпеку та проникнення;

- Загальні та специфічні сценарії застосування;
- Картографування та географія.

Розглянемо більш детально чому саме Python:

1. Python ідеально підходить для веб розробки. Він пропонує величезну кількість корисних бібліотек та фреймворків, які роблять веб розробку дуже простою. Деякі завдання, які займають в PHP години, можуть бути виконані за лічені хвилини на Python. Більш того, він часто використовується для парсингу контенту (Web Scraping). Також багато популярних веб-сайтів було створено за допомогою Python, наприклад Reddit.
2. Python чудово підходить для Data Science (наука про дані). Це одна з найвагоміших причин популярності Python. Причиною цього є можливість користування бібліотеками PyBrain та PyMySQL на ШІ.
3. Машинне навчання. Безумовно це один із найпопулярніших напрямків в IT, який рухає всі технології вперед. Алгоритми стають складнішими з кожним днем, найкрашим прикладом є пошукові алгоритми Google, які з кожним днем можуть більш точно відповісти на всі ваші запити. Навколо нас є чат-боти, які майже охопили всі месенджери і навіть Uber (служба таксі) повністю керується алгоритмами. Більшість IT спільноти віддає перевагу у цьому питанні Python.
4. Простота. Найбільша причина для початківців вивчати Python, бо він, на відміну від інших мов, має простий синтаксис і зрозумілі правила. Python – досить читабельний та простий у налаштуванні. Необхідно просто встановити Python і не має проблем із шляхами до проектів та класів як у Java та C++.
5. Python відомий своєю величезною спільнотою. Кожному, хто тільки починає вивчати мову програмування або тим, хто вже є незалежним розробником необхідно іноді звертатися за допомогою або порадою.

Завдяки величезній спільноті можна знайти рішення будь-якої проблеми за лічені хвилини.

6. Бібліотеки та фреймворки. Для Python існує неймовірна кількість бібліотек, фреймворків та модулів з відкритим кодом доступним для розробників. Python має численні бібліотеки для різних потреб. Django та Flask - два найпопулярніші.
7. Автоматизація. Для Python не є проблемою написати програму за 5 хвилин для автоматизації певної рутинної задачі. Python неабияк гарно підходить для написання сценаріїв, інструментів та автоматизації матеріалів.

Всі ці переваги роблять Python багатозадачним. Але головною перевагою для мене є його веб-фреймворк Django. Він був створений двома програмістами із США і зараз налічує велику кількість сайтів написаних на Django, серед яких є Pinterest, The Washington Post, Dropbox і Spotify [2].

Django написаний на чистому Python, тому він має чисту структуру Python. Він побудований на MVT (Model–View–Template) структурі, яка дозволяє розробникам легко змінювати візуальну частину і бізнес логіку одночасно і окремо одна від одної. Три шари (Model, View, Template) взагалі відповідають за різні процеси і можуть використовуватись незалежно один від одного. Нижче представлена структура MVT (Model–View–Template) [4].

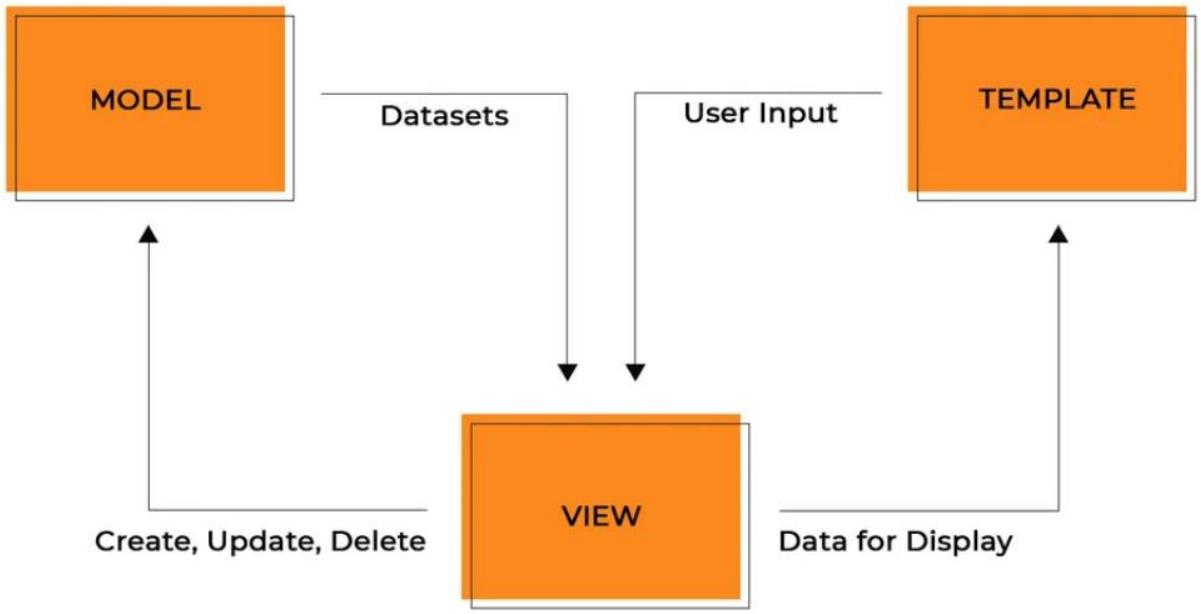

Рисунок 2.1 – MVT структура (джерело: HACKR.IO)

Ця модель представляє єдине джерело інформації про дані, які зберігає код. Вона містить основні поля для даних і Django офіційно підтримує чотири баз даних: PostgreSQL, MySQL, SQLite та Oracle.

Django також має сильну та гнучку систему шаблонів і власну мову розмітки із багатьма інструментами. Взагалі, шаблони – це певні файли, які містять HTML код, який використовується для побудови сайтів. Зазвичай наповнення цих файлів може бути статичним або динамічним. У шаблонах не існує бізнес логіки, тобто він створюється лише для представлення даних. А от моделі містять бізнес-логіку, також власні методи, властивості та атрибути, які допомагають маніпулювати даними. Більш того моделі допомагають розробникам створювати, читати, оновлювати, та іноді й видаляти об’єкти із бази даних. Що ж робить Django таким же чудовим для веб-розробки, як і Python?

1. Величезна екосистема. Django може з легкістю інтегрувати сторонні додатки. В залежності від мети вони можуть бути інтегровані у будь-який проект. Власне, Django складається з багатьох додатків, наприклад додаток для авторизації та відправки електронних листів,

для цих дій можна не писати код самостійно, а просто інтегрувати додаток. До того ж, Django має величезну спільноту, яка може допомогли у будь-якій ситуації.

2. Адміністративна панель за замовчуванням. Адміністративна панель допомагає адміністраторам керувати ресурсом, аби сторонні люди не мали доступу до конфіденційних даних інших людей. Ця панель у Django генерується автоматично за допомогою коду. Також варто зазначити, що вона може легко налаштовуватись – відображати більше або менше полів, колір, формат та положення на сторінці.
3. Вільне встановлення додаткових плагінів. Плагіни та різні компоненти дозволяють розробникам розширювати свої можливості. Існує сотні пакетів, які допоможуть інтегрувати складні системи. Наприклад, додати Google Maps, платіжні системи, або системи авторизації. Якщо у майбутньому вам знадобиться масштабувати ваш додаток, ви можете з легкістю видалити ці плагіни та приєднати нові, більш сучасні.
4. Бібліотеки. Вони відіграють значну роль у веб-програмуванні, неабияк пришвидшують процес розробки. За допомогою встановлених бібліотек розробник може не писати певні фрагменти коду самостійно, а просто використати готові технології у бібліотеках. Деякі бібліотеки включають у собі Django REST фреймворк, який допомагає будувати інтерфейси прикладного програмування, які створюються для управління контентом веб-додатку.
5. ORM (object-relational mapper). ORM допомагає розробникам взаємодіяти з базою даних. ORM – це бібліотека, яка автоматично передає дані в об’єкти. Ця можливість значно пришвидшує розробку адже тепер не потрібно знати як взаємодіяти з базою даних самостійно і забирати звідти дані.

Важливо також зазначити, що для гармонійного веб-додатку необхідно зробити його інтерактивним. Основною задачею для мене було завантаження даних без перезавантаження сторінки та необхідно було забирати дані із бази даних та передавати їх користувачеві. Для цього необхідно використовувати JavaScript та jQuery.

JavaScript – один із найпопулярніших і затребуваних мов програмування. Будь-який рух користувача породжує певну подію – саме ці події обробляє JavaScript. Так як він повністю інтегрується у веб-сторінку, то він може змінювати її будь-яким чином. Ця мова програмування ідеально підходить для веб-сервісів, так як просто неможливо уявити веб-сервіс без JavaScript. Він робить саме те, що необхідно для мого проекту:

- Може задавати зовнішній вигляд додаючи класи та атрибути;
- Відправляти запити на сервер;
- Запитувати дані у користувача.

Переді мною стояла ще одна важлива задача. Так як в основі мого проекту лежить використання картографічного API, на якому відображаються мітки карток, необхідно визначитися із бібліотекою. У мене був вибір між Google Maps і Leaflet. Після невеликого аналізу вибір пав на Leaflet. Далі буде наведена таблиця порівняння цих двох варіантів.

Таблиця 2.1 – Аналіз бібліотек

Теза / Бібліотека	Google Maps	Leaflet
Простота встановлення	+	++
Широкий вибір налаштувань	+	++
Безоплатність	- +	+
Поширеність у використанні	++	+

Отже, Leaflet – безоплатна, легка у використанні бібліотека, яка просто інтегрується в проект шляхом додавання посилання в заголовок CSS документу і посилання JavaScript у кінець файлу. До того ж, Leaflet працює на багатьох платформах включаючи мобільні та настільні. Великою перевагою є те, що до неї можна підключити велику кількість плагінів, до яких є добре написана документація. Код відкритий та може змінюватися будь-ким [5].

У той же час для використання Google Maps API необхідно створити додаткову облікову сторінку і додати банківську картку. Беручи до уваги, що проект не фінансується, а створюється студентом, то додавання банківської картки є мінусом. Варто зазначити, що документація Google Maps API є досить структурованою та великою.

2.2 Система управління базою даних

Для мого проекту необхідна система управління базою даних, де будуть зберігатися зображення, дані користувачів та карток.

За замовчуванням Django в якості бази даних використовує SQLite, але насправді він підтримує ще PostgreSQL, MySQL та Oracle. У чому різниця та чому я обрала саме SQLite?

SQLite – це чудова бібліотека, котра зазвичай вбудовується у додаток. Так як це файлова база даних, виходячи з цього вона дозволяє використовувати різноманітний набір інструментів для обробки та аналізу будь-яких типів даних. У тому випадку, коли використовується SQLite, то зв’язок із додатком відбувається за допомогою прямих та функціональних викликів файлів. При цьому не використовується ніякий інтерфейс, який прискорює роботу операцій.

Переваги:

1. Файлова система: вся база даних зберігається в одному файлі.

2. SQLite використовує в основі SQL, тобто деякі функції, такі як RIGHT OUTER JOIN не доступні, але є багато нових та корисних.

3. Чудово підходить для розробки: інколи розробнику необхідно масштабувати свою базу даних і SQLite може запропонувати досить багатий функціонал для цього, при цьому залишаючись простою системою.

Недоліки:

1. Немає можливості додаткових налаштувань для того аби зробити її більш продуктивною.

MySQL – найпопулярніша система серед розробників. Вона дуже проста в освоєнні, має велику базу знань в інтернеті, тому знайти необхідну інформацію буде легко.

Переваги:

1. Простота.
2. Багато функцій.
3. Швидкість. Так як MySQL не використовує деякі стандарти SQL, то вона може працювати швидше.

Недоліки:

1. Багато обмежень. MySQL не може виконати будь-що.
2. Безпека. Деякі операції реалізовані менш надійно.
3. Рідко оновлюється. Робота над MySQL зараз сповільнилася.

PostgreSQL – найбільш сучасна та прогресивна система. PostgreSQL направлена на повну відповідність всім стандартам SQL. Значною відмінністю є те, що її функціонал є об’єктно-орієнтованим, у тому числі підтримує принцип ACID. PostgreSQL є багатофункціональною системою тому чудово може підтримувати виконання багатьох процесів водночас.

Переваги:

1. Підтримує повну сумісність з SQL.
2. Підтримка документації та статей в мережі зростає з кожним днем.
3. Масштабованість.
4. Об’єктно-орієнтованість.

Недоліки:

1. Продуктивність. У звичайних і простих операції зчитування даних може поступатися іншим подібним системам.
2. Складність є досить високою для використання новачкам.
3. Хостинг знайти досить складно враховуючи вищезазначені недоліки.

Отже, так як SQLite є уже вбудованою системо за замовчуванням, досить легкою для використання та водночас потужною при обробці файлі, що і потрібно даній базі даних, було вирішено зупинити свій вибій саме на ній. Вона не вимагає запущеного сервера. Всі файли бази даних можуть легко переноситися з одного комп'ютера на інший. Однак при необхідності є можливість використовувати в Django більшість поширених СУБД. Для зручної роботи із базою даних у «реальному часі» було вирішено додати адміністративну панель, через яку можна додавати, змінювати та видаляти дані з таблиць. Все це за замовчуванням підтримується в Django.

2.3 Проектування інформаційної системи

У цьому розділі буде наведено Use Case діаграму моєї системи. Use Case діаграма – це діаграма використання системи, тобто її загальний сценарій поведінки. На схемі зображені актори – це ті, хто безпосередньо будуть брати участь у роботі системи та будуть виконувати певні дії, які позначені в овальних фігурах. Позначка «<include>» означає, що дія виконується обов’язково і без участі користувача. Тобто, коли користувач авторизується у системі, то база даних автоматично робить запис в таблицю Users, при цьому користувач не заносить свої дані вручну у базу даних за допомогою запитів. Позначка «<extend>» означає, що дія може бути виконана за якихось умов або не виконана взагалі – це може залежати від користувача [3].

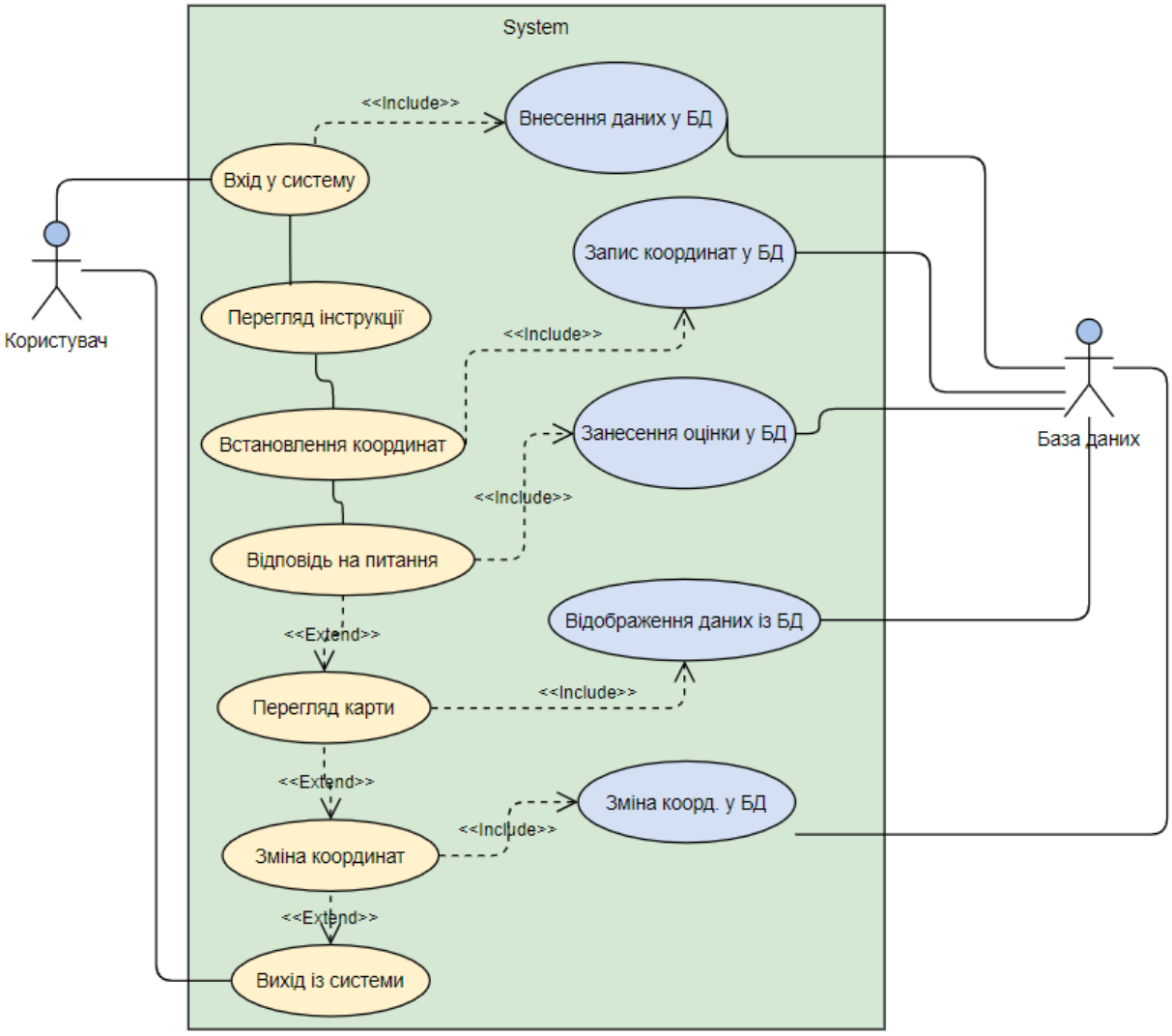

Рисунок 2.2 – Use Case діаграма

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

Розробку програмної частини варто почати зі створення бази даних. У Django не потрібно безпосередньо звертатися до неї – необхідно просто писати свою структуру моделі та інший код, а Django виконує всю роботу за зверненням до бази даних самостійно. Для роботи з базами даних в проекті Django в файлі settings.py визначено параметр DATABASES. Дані для картки users заповнюються автоматично під час авторизації за допомогою Google аккаунту. Далі наведена ER діаграма бази даних мого проекту (див. рис. 2.2).

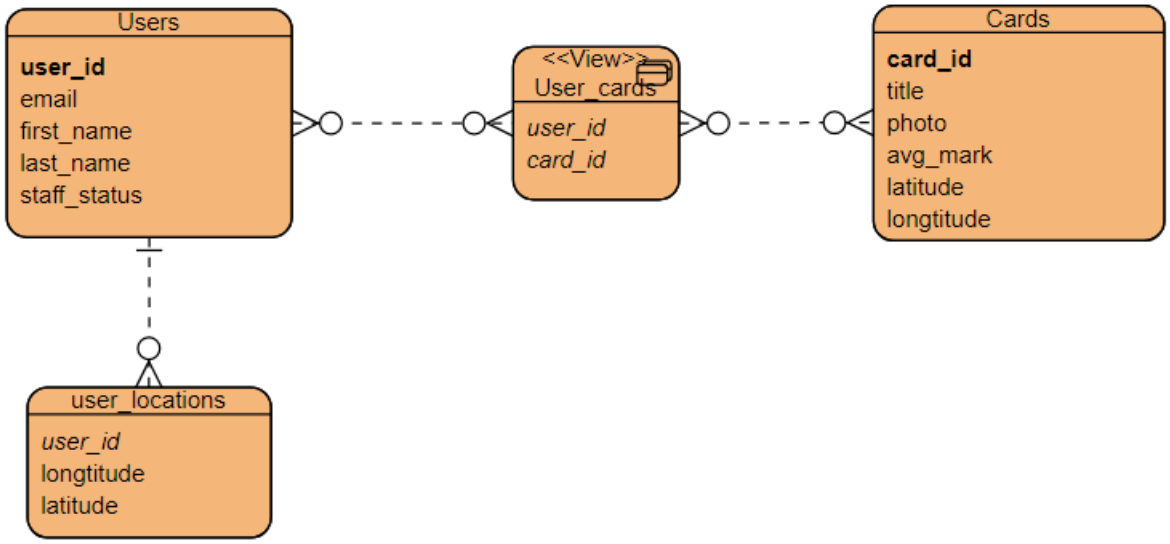

Рисунок 3.1 – ER діаграма

Таблиця 3.1 – Опис полів таблиці Users

Users	
User_id	Ідентифікатор. Створюється автоматично базою
Email	Електрона адреса користувача
First_name	Ім’я
Last_name	Прізвище
Staff_status	Визначення адміністратора. Створюється автоматично базою

Таблиця 3.2 – Опис полів таблиці Cards

Cards	
Card_id	Ідентифікатор. Створюється автоматично базою
Title	Питання, яке буде ставитися користувачеві
Photo	Фото картки
Avg_mark	Середня оцінка картки. Буде визначатися шляхом голосування.
Latitude	Широта. Перша координата на карті для об’єкта.
Longitude	Довгота. Друга координата на карті для об’єкта.

Таблиця User_cards є спільною для USERS і CARDS і створена шляхом типу з’єднання «багато до багатьох». Вона зберігає зв’язок користувача, який відповів на питання аби далі не показувати йому знову ту ж карточку з питанням (див. табл. 3.3).

Таблиця 3.3 – Опис полів таблиці User_cards

User_cards	
User_id	Зовнішній ключ. Ідентифікатор користувача.
Card_id	Зовнішній ключ. Питання, яке буде ставитися користувачеві

Таблиця User_location зберігає id користувача та його місцезнаходження на карті (див. табл. 3.4).

Таблиця 3.4 – Опис полів таблиці User_location

User_location	
User_id	Зовнішній ключ. Ідентифікатор користувача.
Latitude	Широта. Перша координата на карті місцезнаходження користувача.
Longitude	Довгота. Друга координата на карті місцезнаходження користувача.

Для роботи з базами даних в проекті Django в файлі settings.py визначено параметр DATABASES. Ось так він заповнюється автоматично:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

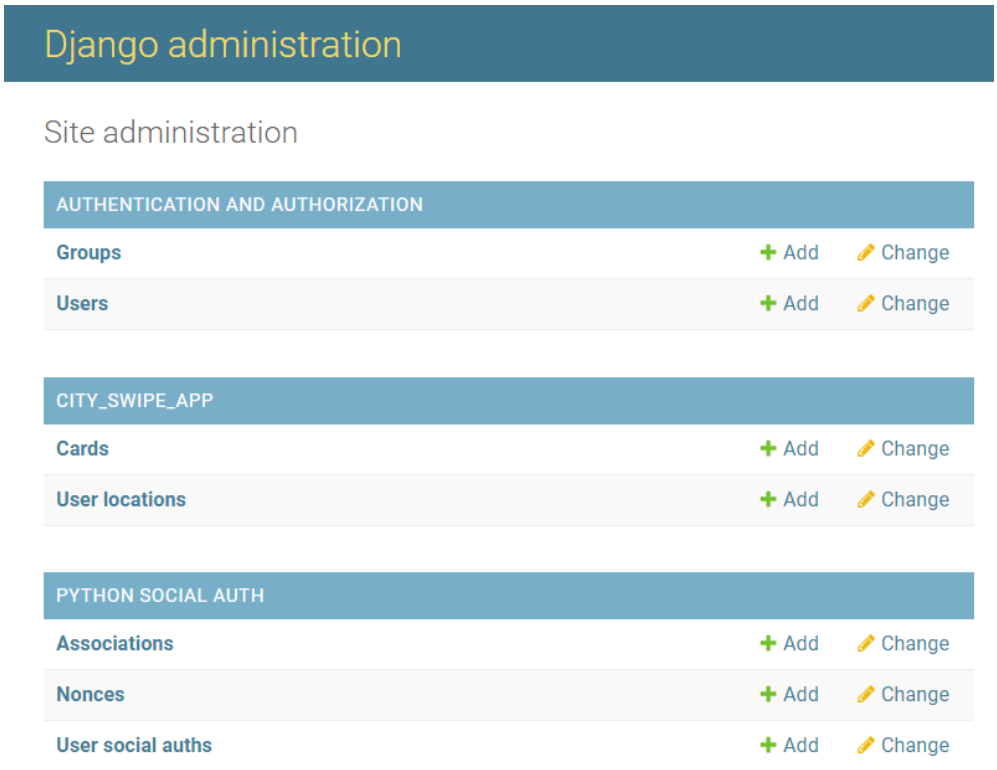

Рисунок 3.2 – Адміністративна сторінка із базою даних

Ось так було створено таблицю Card, де будуть зберігатися всі данні фотокарточок: питання, опис, координати, безпосередньо фото і так далі.

```
class Card(models.Model):
title = models.CharField(max_length=255, default='')
```

```
        about = models.CharField(max_length=255, default='',
blank=True)
        photo = models.ImageField()
        avg_mark = models.IntegerField(default=0)
        latitude = models.FloatField(default=50.908407)
        longitude = models.FloatField(default=34.795837)
        users = models.ManyToManyField(User,
db_table='users_cards', blank=True)
        class Meta:
            db_table = 'Card'
```

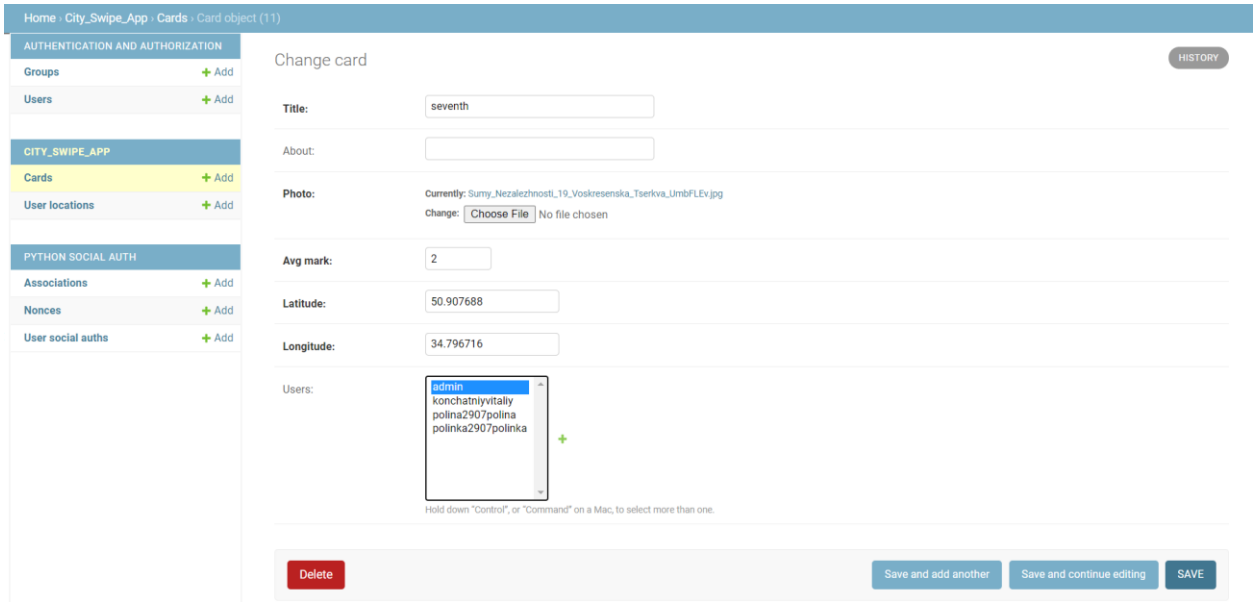

Рисунок 3.3 – Відображення таблиці в адміністративній панелі

3.1 Зберігання зображень

Для зберігання зображень було вирішено використовувати вбудовану систему моделей. За допомогою супер користувача (адміністратора) є можливість керувати даними в базі даних без написання запитів. Django все виконує автоматично.

У файлі admin.py записано такий код, який створює модель Card та UserLocation, модель User створюється автоматично:

```
from django.contrib import admin
from .models import Card
from .models import UserLocation
```

```
admin.site.register(Card)
admin.site.register(UserLocation)
```

У файлі models.py було додано поля для моделей. Для додавання фото було використано вбудовану функцію ImageField().

```
class Card(models.Model):
    title = models.CharField(max_length=255, default='')
    about = models.CharField(max_length=255, default='',
blank=True)
    photo = models.ImageField()
    avg_mark = models.IntegerField(default=0)
    latitude = models.FloatField(default=50.908407)
    longitude = models.FloatField(default=34.795837)
    users = models.ManyToManyField(User,
db_table='users_cards', blank=True)
    class Meta:
        db_table = 'Card'
```

Клас `UserLocation` описує зв’язок один до одного: користувача та координати визначаючи id користувача як первинний ключ.

```
class UserLocation(models.Model):
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        primary_key=True
    )
    latitude = models.FloatField()
    longitude = models.FloatField()
```

3.2 Авторизація

Аутентифікацію користувачів було виконано за допомогою облікових записів Google. Для цього було використано бібліотеку social-app-django, яка реалізує механізм аутентифікації / реєстрації.

Для забезпечення аутентифікації користувачів Google підтримує велику кількість різних протоколів, включаючи OAuth2. OAuth – це відкритий протокол, який забезпечує безпечну авторизацію для веб-додатків, додатків для мобільних пристроїв і настільних комп'ютерів простим способом. Протокол заснований на довіреній третій стороні для встановлення процесу аутентифікації. Він надає клієнтський доступ до ресурсу, делегує процес авторизації на зовнішній сервер авторизації з дозволу власника ресурсу.

В якості першого кроку необхідно було встановити і включити бібліотеку social-auth-app-django в проекті. Після того, як бібліотека встановлена, було додано додаток до списку INSTALLED_APPS в файлі налаштувань проекту, використовуючи ідентифікатор social_django:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'city_swipe_app',
    'social_django',
]
```

Крім цього, було додано бекенд Google OAuth2 в список бекендів аутентифікації. Також, явно було включено бекенд аутентифікації моделі за замовчуванням, щоб продовжити використання сайту адміністратора django з використанням локальних облікових записів:

```
AUTHENTICATION_BACKENDS = (
```

```
        'social_core.backends.google.GoogleOAuth2',
        'django.contrib.auth.backends.ModelBackend',
    )
```

Наступним етапом є налаштування у консолі розробників Google облікового запису. Звідти необхідно було взяти секретні ключи, які зв'язують систему із обліковим записом.

```
    SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = 'INSERT_PROVIDED_KEY_HERE'
    SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET =
'INSERT_PROVIDED_SECRET_HERE'
```

Логіку відображення та безпосередньої авторизації було додано на html сторінку

```
    <a class="button_start" href="{% url 'social:begin' 'google-
oauth2' %}">
                    <div class="google-icon-wrapper">
                        <img class="google-icon"
src="https://upload.wikimedia.org/wikipedia/commons/5/53/Google_
%22G%22_Logo.svg"/>
                    </div>
                Авторизація
                </a>
```

3.3 Реалізація карти

Для реалізації відображення міток на карті біло вирішено використати JavaScript бібліотеку Leaflet.

Бібліотека допомагає реалізувати підтримку різних шарів мап, які побудовані за технологією: WMS, GeoJSON або векторного відображення поверхні. Основні типи об'єктів Leaflet:

- Растрові типи (TileLayer і ImageOverlay).
- Векторні типи (Path, Polygon і специфічні типи, такі як Circle).

- Групові типи (LayerGroup, FeatureGroup і GeoJSON).
- Керуючі елементи (Zoom, Layers і т. д.).

Також існують допоміжні класи для управління проекціями, трансформацій і взаємодії з об'єктною моделлю документа (DOM).

На html-сторінці через тег div визначається область, де має відображатись мапа. Для цій області визначається параметри поведінки мапи: джерело мапи, точка та масштаб з якої починати відображати мапу, маркери тощо. Для того, аби користувач зміг позначити своє місцезнаходження на карті, необхідно було визначити радіус, щоб користувач обирав лише місце у Сумах. Наступний фрагмент коду описує додавання основного шару карти. У `tileLayer` записаний токен, який посилається на API карти.

```
    L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}
/{x}/{y}?access_token=pk.eyJ1IjoibWFwYm94IiwiYSI6ImNpejY4NXVycTA
2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {
    maxZoom: 18,
        id: 'mapbox/streets-v11',
        tileSize: 512,
        zoomOffset: -1
    }).addTo(mymap);
```

Наступний фрагмент коду описує функцію, яка вирішує який колір мітки поставити в залежності від рейтингу картки. Якщо рейтинг більше або дорівнює 5, то мітка буде зеленого кольору, якщо менше 5, але більше або дорівнює 0, то мітка жовта, в іншому випадку – червона.

```
    $.get("/city_swipe_app/getAllCards/")
    .done(function(data) {
        for (let i = 0; i < data.length; i++){
            let marker;
            if (data[i].mark >= 5) {
```

```
                marker = L.marker([data[i].latitude,
data[i].longitude], {icon: greenIcon, opacity: 0.8})
            }
            else if (data[i].mark < 5 && data[i].mark >= 0) {
                marker = L.marker([data[i].latitude,
data[i].longitude], {icon: yellowIcon, opacity: 0.8})
            }
            else if (data[i].mark < 0) {
                marker = L.marker([data[i].latitude,
data[i].longitude], {icon: redIcon, opacity: 0.8})
            }
```

Функція `marker.bindPopup` допомагає вивести короткий опис і рейтинг картки у маленькому віконці, яке з'являється при натисканні на мітку на карті. Так можна з легкістю зрозуміти яка це мітка і чому у неї конкретний колір.

```
            marker.bindPopup(data[i].about + '<br> оцінка: ' +
data[i].mark);
            markers.addLayer(marker);
        }
        mymap.addLayer(markers);
        fillStatistic(data);
    });
```

Далі йде функція, яка заповнює статистичні дані з бази. Функція `cards.sort` розраховує картку з найвищим рейтингом, або з найнижчим рейтингом в залежності від формули.

```
    function fillStatistic(cards){
        cards.sort(function(a, b){
            return b.mark - a.mark;
        })
        let bestCards = [];
```

```
        let k = 3;
        for (let i = 0; i < cards.length; i++){
           if (k != 0) {
            bestCards.push(cards[i]);
            k--;
           }
        }
```

Якщо це `b.mark - a.mark`, то буде відібрано картку з найбільшим рейтингом, якщо `a.mark - b.mark`, то з найнижчим. Значення записується у відповідну змінну. У циклі `for` виводяться три перші карточки за рейтингом. Функція `showStatistic` виводить ці три найбільш позитивно та негативно оцінені карточки за допомогою функції `getElementById`.

```
        cards.sort(function(a, b){
            return a.mark - b.mark;
        })
        let worstCards = [];
        k = 3;
        for (let i = 0; i < cards.length; i++){
           if (k != 0) {
            worstCards.push(cards[i]);
            k--;
           }
        }
        showStatistic(bestCards,
document.getElementById('bestCards'));
        showStatistic(worstCards,
document.getElementById('worstCards'));
    }
```

Функція для виведення статистики за допомогою створення елементів (`createElement`) та присвоювання для них `classList`. Цикл `for` потрібен для того, аби елементи створювалися в залежності від кількості самих карток.

```
function showStatistic(Cards, coloumn) {
    for(let i = 0; i < Cards.length; i++){
        let card = document.createElement('div');
        card.classList.add('card');

        let title = document.createElement('span');
        title.classList.add('title');
        title.innerText = 'Питання: ' + Cards[i].title;

        let about = document.createElement('span');
        about.classList.add('about');
        about.innerText = 'Деталі: ' + Cards[i].about;

        let mark = document.createElement('span');
        mark.classList.add('mark');
        mark.innerText = 'Рейтинг: ' + Cards[i].mark;
```

Функція `appendChild` з'єднує два або більше елементів разом, аби вони були один за одним. Це потрібно для зв'язування елементів.

```
        card.appendChild(title);
        card.appendChild(about);
        card.appendChild(mark);

        coloumn.appendChild(card);
    }
}
```

3.4 Математичний опис моделі розрахунку радіусу

Далі представлена математична модель розрахунку радіусу за допомогою коду:

```
function calcDistance(d1, d2) {
  let R = 6378137;
  let dLatitude = rad(d2.lat - d1.lat);
  let dLongitude = rad(d2.lng - d1.lng);
  let formul = Math.sin(dLatitude / 2) * Math.sin(dLatitude / 2) +
    Math.cos(rad(d1.lat)) * Math.cos(rad(d2.lat)) *
    Math.sin(dLongitude / 2) * Math.sin(dLongitude / 2);
  let c = 2 * Math.atan2(Math.sqrt(formul), Math.sqrt(1 - formul));
  let dist = R * c;
  return dist;
}
```

3.5 Використання програмного додатку

Усе починається з авторизації. У цій роботі авторизація відбувається тільки за допомогою google пошти. Можна цілком впевнено зазначити, що цього достатньо.

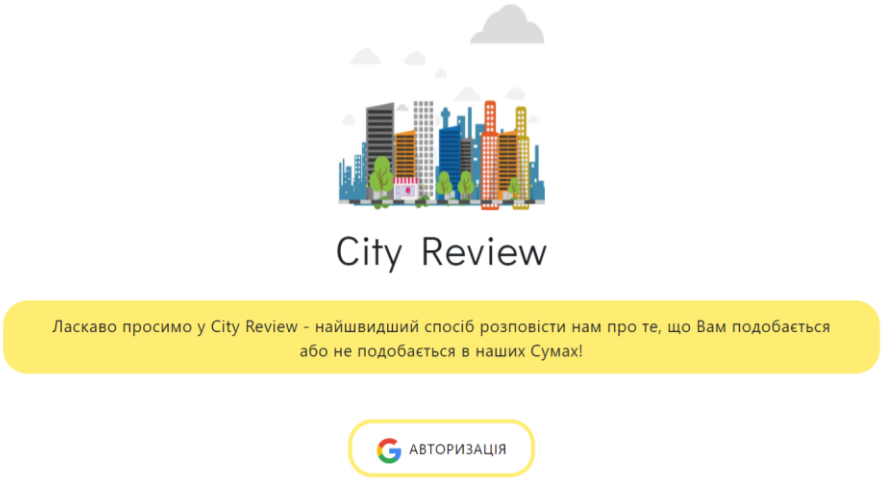

Рисунок 3.4 – Сторінка авторизації

Після авторизації користувач автоматично переходить до сторінки з інструкцією.

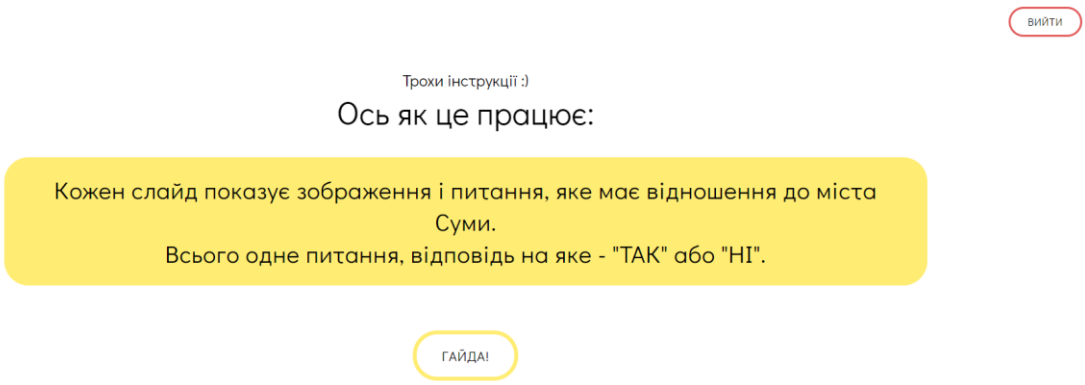

Рисунок 3.5 – Інструкція

Наступним кроком користувачу пропонується обрати своє місце проживання (це може бути місце, де він/вона найчастіше буває). Це необхідно для того, аби система змогла показати конкретні картки в радіусі 1.5км від точки, яку обрав користувач на карті.

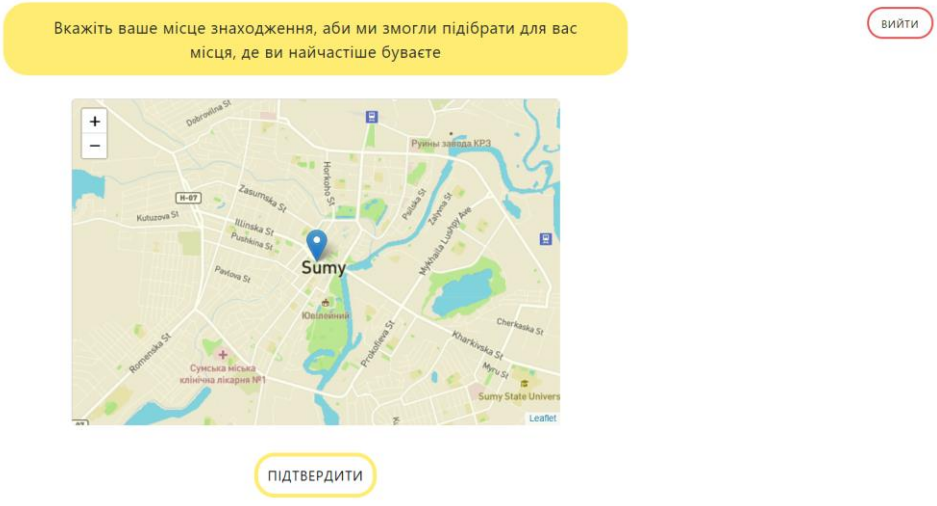

Рисунок 3.6 – Місце проживання

Додатково було вказано загальний радіус міста, аби користувач ненавмисно не обрав точку поза містом. Для наочності радіус позначений жовтим кольором.

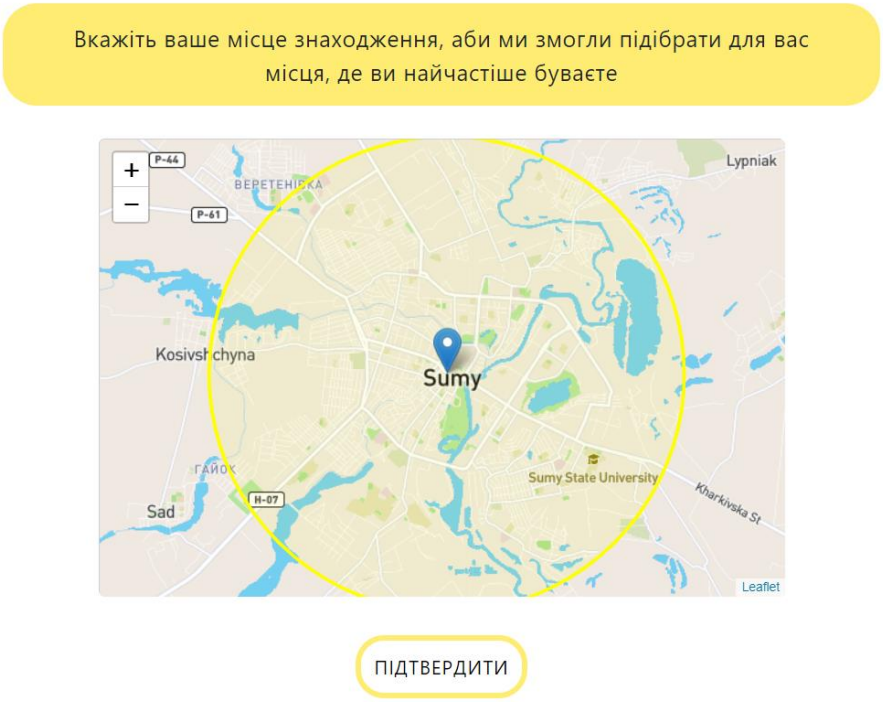

Рисунок 3.7 – Радіус міста

Після того, як було обрано точку на карті користувачу показують картки, які знаходяться в радіусі 1.5км. Користувач має відповісти на поставлене запитання. Далі будуть наведені приклади карток.

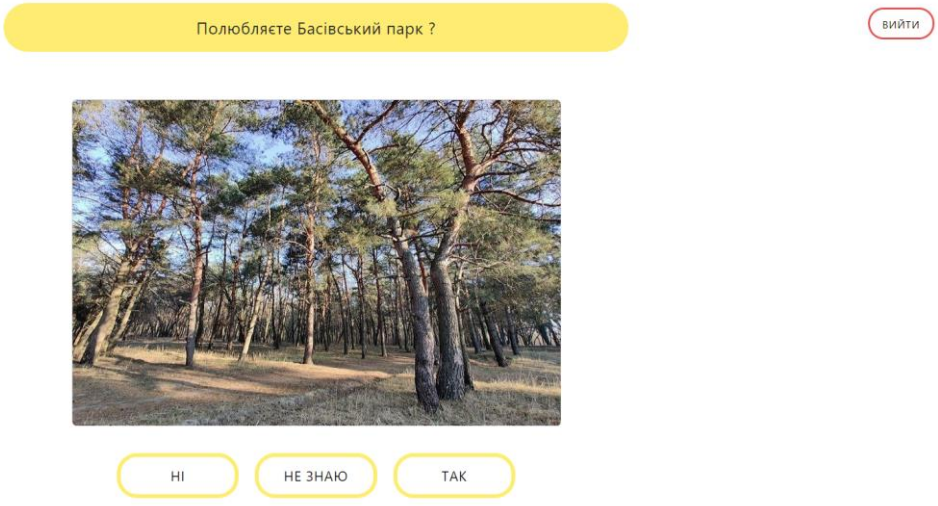

Рисунок 3.8 – Приклад картки

Рисунок 3.9 – Приклад картки

Після того, як всі картки закінчились, користувачу показується інформаційна сторінка. Також є можливість змінити місце знаходження натиснувши на кнопку [Змінити місце розташування] та перейти на перегляд карти із статистикою.

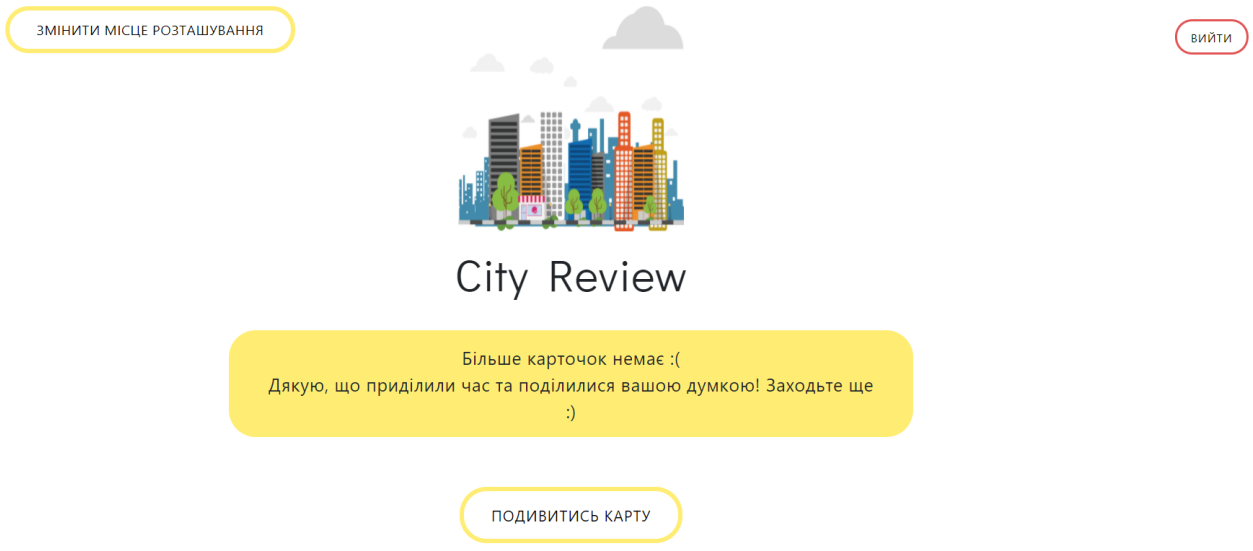

Рисунок 3.10 – Інформаційна сторінка

На загальній карті можна побачити статистику із трьох найбільш позитивно оцінених карток та найбільш негативно оцінених, основну карту із

кластерами міток (зелені кола). При збільшенні карти кластери роз’єднуються і з’являються мітки. Все залежить від кількості міток та масштабу карти.

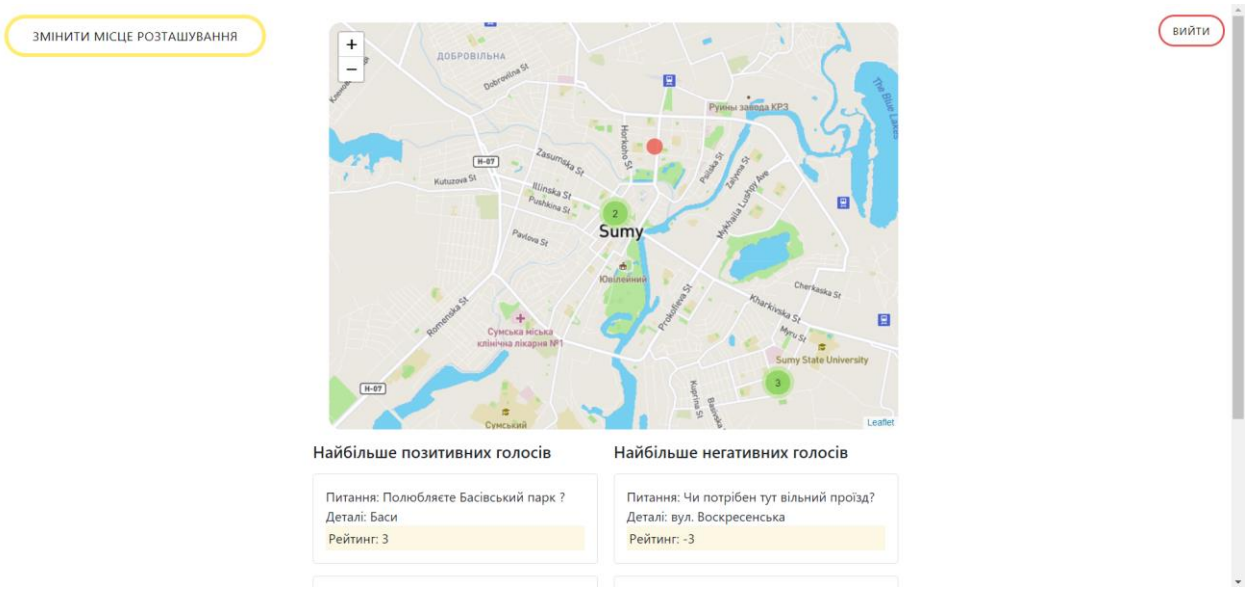

Рисунок 3.11 – Карта із статистикою

При натисканні на мітку можна подивитися її рейтинг та короткий опис. В залежності від рейтингу колір мітки може змінюватися.

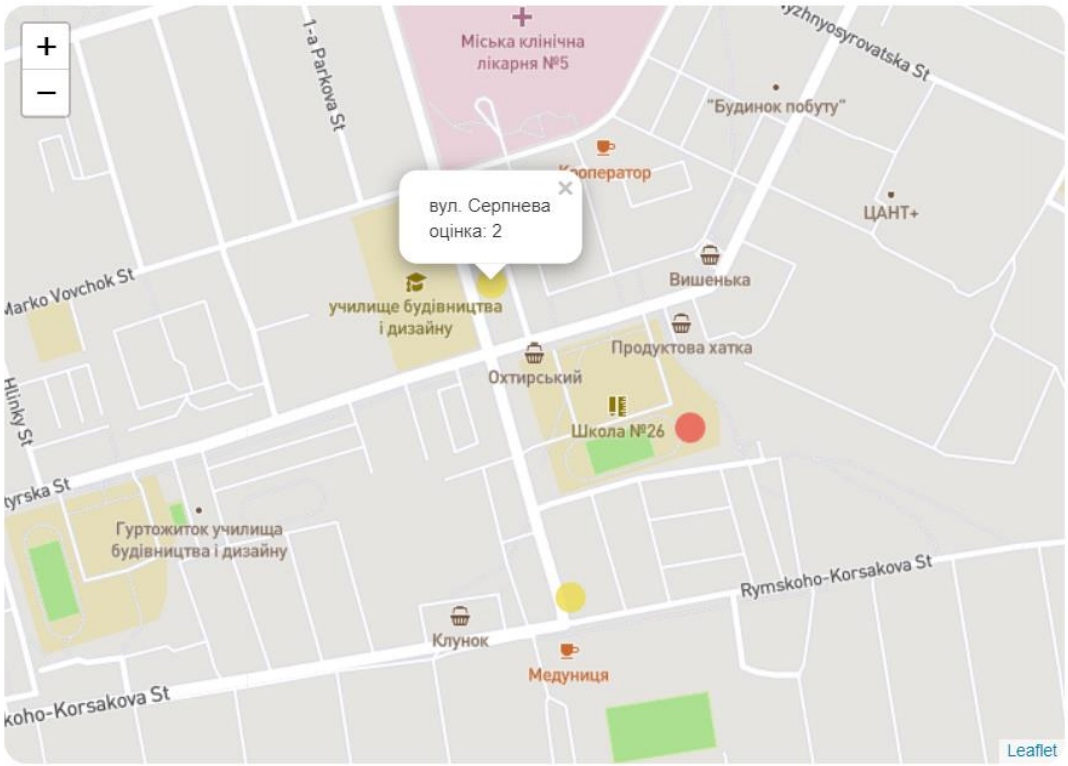

Рисунок 3.12 – Детальний огляд міток

Якщо опустити сторінку нижче, то можна побачити статистику, про яку було згадано раніше.

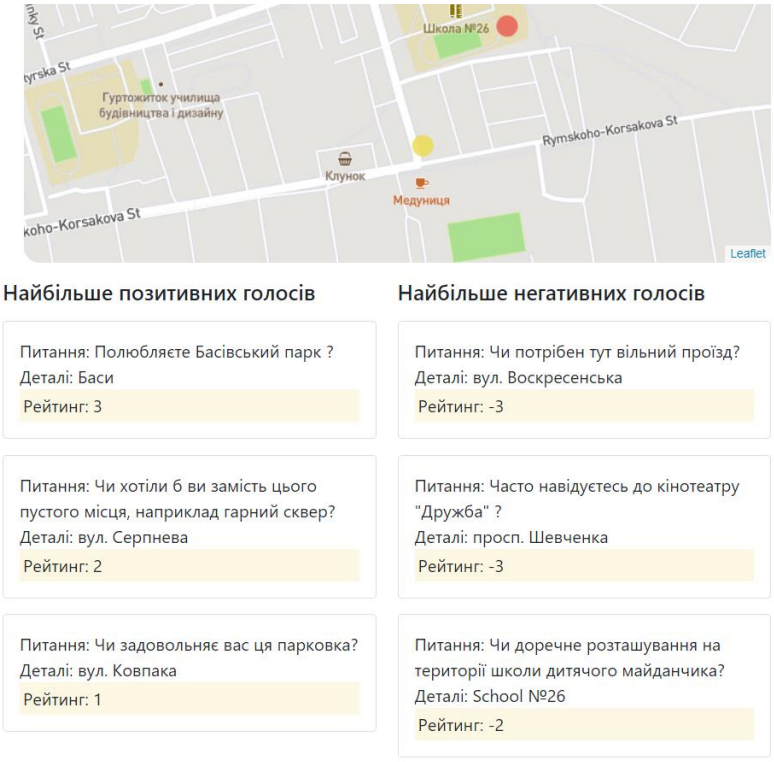

Рисунок 3.13 - Статистика

ВИСНОВКИ

Під час виконання випускної роботи був проведений аналіз участі громади у планування містобудування. Безсумнівно було вирішено, що участь містян несе в основному позитивний характер та для успішного розвитку міста є дуже важливим кроком залучення громадян до голосування та у подальшому брати до уваги їх думки.

Наступним етапом було провести порівняння з аналогом, спроектувати та розробити веб-сервіс, який матиме візуальну та серверну частину (Front-end та Back-end) із необхідним функціоналом – користувач відповідає на поставлене запитання, яке відноситься до планування, архітектури та комфортного проживання в місті Суми.

Після виконаної роботи, можна сказати, що даний веб-сервіс має потенціал до розвитку та буде корисним як для громади, так і для міської влади аби разом розпочати роботу над красивим та комфортним містом.

Список літератури

1. Alessandro Del Sole Visual Studio Code Succinctly, 15-19, 2017
2. Rehman, Christopher Paul, Christopher R. Paul «The Linux Development Platform: Configuring, Using and Maintaining a Complete Programming Environment», 122-124, 2002.
3. В. Доронов Django 2.1. Практика створення веб-сайтів на Python. - СПб.: БХВ-Петербург, 2019.
4. А. Петухов UML. Основи. Короткий посібник по стандартам мови об'єктного моделювання, 18-31, 2011
5. Mike Owens The Definitive Guide to SQLite, Vol. 1, 156-178, 2013
6. U. Malik, M. Goldwasser, B. Johnston, SQL for Data Analytics: Perform fast and efficient data analysis with the power of SQL, 314-320, 2019
7. Опитування громадської думки. Посібник для журналістів. URL: https://dif.org.ua/uploads/pdf/1356088645_2202.pdf (дата звернення: 16.05.21)
8. J. Evans-Cowley and J. Hollander, The New Generation of Public Participation: Internet-based Participation Tools, Planning Practice & Research, Vol. 25, No. 3, 397–408, 2010.
9. Джордж Геллап, Сол Форбс Рей, Пульс демократії. Як працюють опитування громадської думки, 47-92, 2017
10. T. Nam and T. A. Pardo, Smart City as Urban Innovation: Focusing on Management, Policy, and Context, Proceedings of the 5th International Conference on Theory and Practice of Electronic Governance, New York, USA, 185–194, 2011.
11. A. Martos, R. Pacheco-Torres, J. Ordóñez, and E. Jadraque-Gago, Towards successful environmental performance of sustainable cities: Intervening sectors. A review, Renewable and Sustainable Energy Reviews, Vol. 57, 479–495, 2016.

12. A. Kunze, R. Burkhard, S. Gebhardt, and B. Tuncer, Visualization and Decision Support Tools in Urban Planning, in Digital Urban Modeling and Simulation, Springer, Vol. 242, Berlin, 279–298, 2012.
13. E. Gordon, S. Schirra, and J. Hollander, Immersive Planning: A Conceptual Model for Designing Public Participation with New Technologies, Environment and Planning B: Planning and Design, Vol. 38, No. 3, 505–519, 2015.
14. D. Boullier, Digital sociology. Armand Colin, Paris, 2016.
15. M. Saujot and T. de Feraudy, Urban crowdsourcing: digital technologies for sustainable cities?, Iddri Blog, 2016.
16. Unlimited cities. URL: http://unlimitedcities.org/ (дата звернення: 31.05.21)
17. P. V. Thakuriah, N. Tilahun, and M. Zellner, Big Data and Urban Informatics: Innovations and Challenges to Urban Planning and Knowledge Discovery, Proceedings NSF Workshop on Big Data and Urban Informatics, 4–32, 2015.
18. M. Saujot and T. de Feraudy, Urban crowdsourcing: digital technologies for sustainable cities?, Iddri Blog, 2016.
19. D. Boullier and A. Lohard, Opinion mining et Sentiment analysis : Méthodes et outils., OpenEdition Press, Marseille, 2012.

ДОДАТОК

settings.py

```
from pathlib import Path
import os
BASE_DIR = Path(__file__).resolve().parent.parent
SECRET_KEY = '_+fe!wv_!279+%%=bf-7y8)n8#^-
oz!ce)v+pnb7kr!b2l7pq^'


DEBUG = True


ALLOWED_HOSTS = []


# Application definition


INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'city_swipe_app',
    'social_django',
]


MIDDLEWARE = [
```

```
        'django.middleware.security.SecurityMiddleware',
        'django.contrib.sessions.middleware.SessionMiddleware',
        'django.middleware.common.CommonMiddleware',
        'django.middleware.csrf.CsrfViewMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
        'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.clickjacking.XFrameOptionsMiddleware',
    ]

    ROOT_URLCONF = 'city_swipe.urls'

    TEMPLATES = [
        {
            'BACKEND':
'django.template.backends.django.DjangoTemplates',
            'DIRS': [],
            'APP_DIRS': True,
            'OPTIONS': {
                'context_processors': [
                    'django.template.context_processors.debug',
'django.template.context_processors.request',
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
```

```
                ],
            },
        },
    ]

    WSGI_APPLICATION = 'city_swipe.wsgi.application'


    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': BASE_DIR / 'db.sqlite3',
        }
    }
    AUTH_PASSWORD_VALIDATORS = [
        {
            'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarity
Validator',
        },
        {
            'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator'
,
        },
        {
            'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator
',
```

```
        },
        {
            'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidato
r',
        },
    ]
    LANGUAGE_CODE = 'en-us'
    TIME_ZONE = 'Europe/Istanbul'
    USE_I18N = TrueUSE_L10N = True
    USE_TZ = True
    STATIC_URL = '/static/'
    STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static'),)
    MEDIA_URL = '/media/'
    MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

    AUTHENTICATION_BACKENDS = (
        'social_core.backends.google.GoogleOAuth2',
        'django.contrib.auth.backends.ModelBackend',
    )

    SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = '51860423878-
uahtkvhfhk65i78i245bn209tdgd975t.apps.googleusercontent.com'
    SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET =
'x4i22jVnNZAOLTSwYMdKeg2S'
    SOCIAL_AUTH_URL_NAMESPACE = 'social'
    SOCIAL_AUTH_GOOGLE_OAUTH2_AUTH_EXTRA_ARGUMENTS = {'prompt':
'select_account'}
```

```
LOGIN_URL = '/login/google-oauth2/'
LOGIN_REDIRECT_URL = '/city_swipe_app/instruction/'
LOGOUT_REDIRECT_URL = '/city_swipe_app/'
```

views.py

```
from django.http import HttpResponse, HttpResponseBadRequest
from django.shortcuts import render, redirect
from django.views.decorators.csrf import csrf_exempt
import json
from .models import Card
from .models import UserLocation
from django.contrib.auth.models import User
from geopy.distance import geodesic

def index(request):
    if request.user.is_authenticated:
        return redirect('/city_swipe_app/instruction')
    else:
        return render(request, "index.html")

def instruction(request):
    return render(request, "instruction.html")

def mainPage(request):
    if request.user.is_authenticated:
        return render(request, "mainPage.html")
    else:
```

```
            return render(request, "index.html")


    def endPage(request):
        if request.user.is_authenticated:
            return render(request, "end.html")
        else:
            return render(request, "index.html")


    def mapPage(request):
        if request.user.is_authenticated:
            return render(request, "mapPage.html")
        else:
            return render(request, "index.html")


    def getCard(request):
        if request.user.is_authenticated:
            user_lat = request.GET['latitude']
            user_lng = request.GET['longtitude']
            user_id = request.user.id
            non_view_cards =
Card.objects.exclude(users__id=user_id)
            user = User.objects.get(id=user_id)


            revalant_cards = []
            for card in non_view_cards:
                if calcDistance(user_lat, user_lng,
card.latitude, card.longitude) <= 1.5:
```

```
                    revalant_cards.append(card)


            if len(revalant_cards) >= 1:
                card = dict(id = revalant_cards[0].id, title =
revalant_cards[0].title, about = revalant_cards[0].about,
latitude = revalant_cards[0].latitude, longitude =
revalant_cards[0].longitude, photo =
revalant_cards[0].photo.url)
                revalant_cards[0].users.add(user)
                return HttpResponse(json.dumps(card),
content_type='application/json')
            else:
                return HttpResponse('404',
content_type='application/json')
        else:
            return redirect('/city_swipe_app/')


    def getUser(request):
        if request.user.is_authenticated:
            user_id = request.user.id
            user_location =
UserLocation.objects.filter(user_id=user_id)
            if user_location:
                result = dict(username=request.user.username,
longitude = user_location[0].longitude, latitude =
user_location[0].latitude)
                return HttpResponse(json.dumps(result),
content_type='application/json')
            else:
```

```
                return HttpResponse('404',
content_type='application/json')
        else:
            return redirect('/city_swipe_app/')


    @csrf_exempt
    def setUserLocation(request):
        if request.user.is_authenticated:
            user = request.user
            if request.method == 'GET':
                return HttpResponseBadRequest('This is POST
method')
            elif request.method == 'POST':
                lat = request.POST['latitude']
                lng = request.POST['longtitude']
                location = UserLocation(user=user, latitude=lat,
longitude=lng)
                location.save()
                return HttpResponse('ok',
content_type='application/json')
        else:
            return redirect('/city_swipe_app/')


    def resetUserLocation(request):
        if request.user.is_authenticated:
            user = request.user
            UserLocation.objects.get(user=user).delete()
```

```
            return HttpResponse('ok',
content_type='application/json')
        else:
            return redirect('/city_swipe_app/')


    @csrf_exempt
    def submitAnswer(request):
        if request.user.is_authenticated:
            if request.method == 'GET':
                return HttpResponseBadRequest('This is POST
method')
            elif request.method == 'POST':
                card_id = request.POST['card']
                answer = request.POST['answer']


                card = Card.objects.get(id=card_id)


                if answer == 'no_btn':
                    card.avg_mark = card.avg_mark - 1
                if answer == 'yes_btn':
                    card.avg_mark = card.avg_mark + 1


                card.save()
                return HttpResponse('ok',
content_type='application/json')
        else:
            return redirect('/city_swipe_app/')
```

```
    def getAllCards(request):
        if request.user.is_authenticated:
            allCards = Card.objects.all()
            listOfCards = []
            for card in allCards:
                cardObject = dict(id = card.id, title =
card.title, about = card.about, latitude = card.latitude,
longitude = card.longitude, photo = card.photo.url, mark =
card.avg_mark)
                listOfCards.append(cardObject)
            return HttpResponse(json.dumps(listOfCards),
content_type='application/json')
        else:
            return redirect('/city_swipe_app/')
    def calcDistance(lat1, lng1, lat2, lng2):
        point1 = (lat1, lng1)
        point2 = (lat2, lng2)
        return geodesic(point1, point2).km
```

models.py

```
    from django.db import models
    from django.contrib.auth.models import User


    class Card(models.Model):
        title = models.CharField(max_length=255, default='')
        about = models.CharField(max_length=255, default='',
blank=True)
        photo = models.ImageField()
```

```
        avg_mark = models.IntegerField(default=0)
        latitude = models.FloatField(default=50.908407)
        longitude = models.FloatField(default=34.795837)
        users = models.ManyToManyField(User,
db_table='users_cards', blank=True)
        class Meta:
            db_table = 'Card'


    class UserLocation(models.Model):
        user = models.OneToOneField(
            User,
            on_delete=models.CASCADE,
            primary_key=True
        )
        latitude = models.FloatField(
        longitude = models.FloatField()
```

urls.py

```
from django.conf.urls import url
from django.urls import path
from . import views
from .views import getCard


app_name = 'city_swipe_app'


urlpatterns = [
    url(r'^$', views.index, name='index'),
```

```
        path('instruction/', views.instruction,
name='instruction'),
        path('mainPage/', views.mainPage, name='mainPage'),
        path('getCard/', views.getCard, name='getCard'),
        path('getUser/', views.getUser, name='getUser'),
        path('setUserLocation/', views.setUserLocation,
name='setUserLocation'),
        path('submitAnswer/', views.submitAnswer,
name='submitAnswer'),
        path('endPage/', views.endPage, name='endPage'),
        path('mapPage/', views.mapPage, name='mapPage'),
        path('getAllCards/', views.getAllCards,
name='getAllCards'),
        path('resetUserLocation/', views.resetUserLocation,
name='resetUserLocation'),
    ]
```

main.js

```
'use strict';

let userLatitude = 50.907688;
let userLongtitude = 34.796716;

let centerLat = 50.907688;
let centerLong = 34.796716;

function rad(x) {
  return x * Math.PI / 180;
}
```

```
    function calcDistance(d1, d2) {
      let R = 6378137;
      let dLatitude = rad(d2.lat - d1.lat);
      let dLongitude = rad(d2.lng - d1.lng);
      let formul = Math.sin(dLatitude / 2) * Math.sin(dLatitude
/ 2) +
        Math.cos(rad(d1.lat)) * Math.cos(rad(d2.lat)) *
        Math.sin(dLongitude / 2) * Math.sin(dLongitude / 2);
      let c = 2 * Math.atan2(Math.sqrt(formul), Math.sqrt(1 -
formul));
      let dist = R * c;
      return dist;
    }

    $(document).ready(function() {
        getUserLocation();
        let buttons = document.getElementsByClassName('answer-
btn');
        for (let i = 0; i < buttons.length; i++) {
          buttons[i].addEventListener('click', function(){
            submitAnswer(this);
          });
        }
    });

    function getUserLocation() {
```

```
        $.get("/city_swipe_app/getUser/", {})
          .done(function(data){
            if (data === 404) {
              showLocationForm()
            } else {
              let user = data;
              userLatitude = user.latitude;
              userLongtitude = user.longitude;
              getCard();
            }
          });
      }

      function submitAnswer(answer) {
        let cardId =
document.getElementById('cardImage').dataset.cardid;
        $.post("/city_swipe_app/submitAnswer/", {answer:
answer.id, card: cardId});
        getCard();
      }

      function submitLocation() {
        $.post("/city_swipe_app/setUserLocation/", {latitude:
userLatitude, longtitude: userLongtitude});
        document.getElementById('submitLocation').remove();
        getCard();
      }
```

```
    function getCard() {
      let card;
      $.get("/city_swipe_app/getCard/", {latitude: userLatitude,
longtitude: userLongtitude})
        .done(function(data) {
          if (data === 404) {
            window.location.replace("/city_swipe_app/endPage");
          }else {
            showCard(data);
          }
      });
      return card;
    }

    function showLocationForm() {
        document.getElementsByClassName('btn-
container')[0].hidden = true;
        let title = document.getElementById('questionTitle');
        let mapElem = document.createElement('div');
        let button = document.createElement('button');
        title.innerHTML = 'Вкажіть ваше місце знаходження, аби
ми змогли підібрати для вас місця, де ви найчастіше буваєте';
        mapElem.id = 'map';
        button.id = 'submitLocation';
        button.innerText = 'Підтвердити'
```

```
        button.addEventListener('click', function() {
          submitLocation();
        });

        document.getElementById('card').appendChild(mapElem);

document.getElementById('mainContainer').appendChild(button);

        let mymap = L.map('map').setView([50.907688, 34.796716],
13);


    L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}
/{x}/{y}?access_token=pk.eyJ1IjoibWFwYm94IiwiYSI6ImNpejY4NXVycTA
2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {
            maxZoom: 18,
            id: 'mapbox/streets-v11',
            tileSize: 512,
            zoomOffset: -1
        }).addTo(mymap);

        let circle = L.circle([50.907688, 34.796716], {
          color: 'yellow',
          fillColor: 'yellow',
          fillOpacity: 0.08,
          radius: 5000
        }).addTo(mymap);
```

```
        let marker = L.marker([50.907688,
34.796716]).addTo(mymap);
        mymap.on('click', function(e){
          let center = {
            lat: centerLat,
            lng: centerLong,
          }
          let point = {
            lat: e.latlng.lat,
            lng: e.latlng.lng,
          }

          userLatitude = e.latlng.lat;
          userLongtitude = e.latlng.lng;

          if(calcDistance(center, point) <= 5000) {
            marker.remove()
            marker = L.marker(e.latlng).addTo(mymap);
          }
        });
    }

    function showCard(card) {
        document.getElementsByClassName('btn-
container')[0].hidden = false;
        document.getElementById('card').innerHTML = '';
```

```
        document.getElementById('questionTitle').innerText =
card.title;
        let photo = document.createElement('img');
        photo.src = card.photo;
        photo.id = 'cardImage'
        photo.dataset.cardid = card.id;

        let aboutCard = document.createElement('aboutCard');
        aboutCard.innerText = card.about;
        aboutCard.className = 'aboutCard';

        document.getElementById('card').appendChild(aboutCard);
        document.getElementById('card').appendChild(photo);
    }
```

map.js

```
    $(document).ready(function(){

document.getElementById('resetLoc').addEventListener('click',
function(){
            $.get("/city_swipe_app/resetUserLocation/");
            window.location.replace("/city_swipe_app/mainPage");
        })
    })

    let mymap = L.map('map').setView([50.907688, 34.796716],
13);
```

```
    let greenIcon = L.icon({
        iconUrl: '/static/city_swipe_app/green.png',
        iconSize: [20, 20],
    });

    let yellowIcon = L.icon({
        iconUrl: '/static/city_swipe_app/yellow.png',
        iconSize: [20, 20],
    });

    let redIcon = L.icon({
        iconUrl: '/static/city_swipe_app/red.png',
        iconSize: [20, 20],
    });

    L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}
/{x}/{y}?access_token=pk.eyJ1IjoibWFwYm94IiwiYSI6ImNpejY4NXVycTA
2emYycXBndHRqcmZ3N3gifQ.rJcFIG214AriISLbB6B5aw', {
    maxZoom: 18,
        id: 'mapbox/streets-v11',
        tileSize: 512,
        zoomOffset: -1
    }).addTo(mymap);

    let markers = L.markerClusterGroup();

    $.get("/city_swipe_app/getAllCards/")
```

```
    .done(function(data) {
        for (let i = 0; i < data.length; i++){
            let marker;
            if (data[i].mark >= 5) {
                marker = L.marker([data[i].latitude,
data[i].longitude], {icon: greenIcon, opacity: 0.8})
            }
            else if (data[i].mark < 5 && data[i].mark >= 0) {
                marker = L.marker([data[i].latitude,
data[i].longitude], {icon: yellowIcon, opacity: 0.8})
            }
            else if (data[i].mark < 0) {
                marker = L.marker([data[i].latitude,
data[i].longitude], {icon: redIcon, opacity: 0.8})
            }
            marker.bindPopup(data[i].about + '<br> оцінка: ' +
data[i].mark);
            markers.addLayer(marker);
        }
        mymap.addLayer(markers);
        fillStatistic(data);
    });

    function fillStatistic(cards){
        cards.sort(function(a, b){
            return b.mark - a.mark;
        })
```

```
        let bestCards = [];
        let k = 3;
        for (let i = 0; i < cards.length; i++){
           if (k != 0) {
            bestCards.push(cards[i]);
            k--;
           }
        }

        cards.sort(function(a, b){
            return a.mark - b.mark;
        })
        let worstCards = [];
        k = 3;
        for (let i = 0; i < cards.length; i++){
           if (k != 0) {
            worstCards.push(cards[i]);
            k--;
           }
        }
        showStatistic(bestCards,
document.getElementById('bestCards'));
        showStatistic(worstCards,
document.getElementById('worstCards'));
     }

     function showStatistic(Cards, coloumn) {
```

```
    for(let i = 0; i < Cards.length; i++){
        let card = document.createElement('div');
        card.classList.add('card');

        let title = document.createElement('span');
        title.classList.add('title');
        title.innerText = 'Питання: ' + Cards[i].title;

        let about = document.createElement('span');
        about.classList.add('about');
        about.innerText = 'Деталі: ' + Cards[i].about;

        let mark = document.createElement('span');
        mark.classList.add('mark');
        mark.innerText = 'Рейтинг: ' + Cards[i].mark;

        card.appendChild(title);
        card.appendChild(about);
        card.appendChild(mark);

        coloumn.appendChild(card);
    }
}
```