МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

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

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інтерактивна інформаційна система «Мапа бомбосховищ України»»
Здобувача групи ІН-91 Нефьодова Назара Юрійовича

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

_____ Назар НЕФЬОДОВ
(підпис)

Керівник,
асистент,
кандидат фізико-математичних наук Олександр ВЛАСЕНКО _____
(підпис)

Суми – 2023

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

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

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-91 Нефьодова Назара Юрійовича

1. Тема роботи: «Інформаційна технологія прогнозування курсу валют»
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи _____
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Огляд проблеми та потенційних можливостей інтерактивної інформаційної системи. 2) Огляд технологій, що використовуються для створення інтерактивних мап. 3) Проектування архітектури веб-додатку та дизайну. 4) Розробка інформаційної системи з інтерактивною мапою бомбосховищ України. 5) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 2023 р.

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз предметної області*		
2	*Постановка завдання*		
3	*Розробка архітектури системи, структури бази даних*		
4	*Практична реалізація інформаційної системи*		
5	*Аналіз результатів та підведення висновків*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 70 стр., 17 рис., 1 додаток, 16 літературних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, адже ідея інтерактивної мапи може бути розповсюджена у будь-яку область життя, а подібна інформаційна система може врятувати життя, вчасно надавши актуальну інформацію.

Об’єкт дослідження – дослідження необхідних аспектів для побудови інтерактивної інформаційної системи «Мапа бомбосховищ України».

Мета роботи – розробка веб-додатку мовою програмування Java із застосуванням Spring Framework для поширення та регуляції даних про бомбосховища на території України.

Результати – проведено аналіз літератури, інструментів та методів розробки веб-додатків для створення інформаційних систем, що дозволяють обмінюватись даними між клієнтом та сервером із застосуванням мови програмування Java. Після вибору інструментів та розробки архітектури інформаційної системи, було розроблено інтерактивну інформаційну систему, що дозволяє усім користувачам переглядати інформацію про усі бомбосховища, занесені у систему, а зареєстрованим користувачам дозволяє також створювати нові, видаляти старі та змінювати бомбосховища. При створенні інформаційної системи було застосовано Spring Framework, а також використано Google JavaScript Maps API.

ВЕБ-СЕРВІС, ІНФОРМАЦІЙНА СИСТЕМА, ІНТЕРАКТИВНА МАПА, WEB-SERVICE, INFORMATION SYSTEM, SPRING FRAMEWORK

ЗМІСТ

ВСТУП

Стрімкий ріст інформаційних технологій зробив найшвидшим і найлегшим засобом розповсюдження інформації Інтернет. Інтернет надає можливість передавати інформацію далеко і швидко, а також достатньо надійно навіть для надважливих даних.

До того ж, сучасний світ вже давно намагається доносити інформацію якомога доступніше і прозоріше. Одним із сучасних методів є інтерактивні мапи. При наочному зображенні таких мап у мережі Інтернет ми можемо отримати високу актуальність інформації, легкість сприйняття, а також загальну доступність до даних.

Області використання інтерактивних мап достатньо широкі, а тем, які може розкрити інтерактивна карта – ще більше. У медійній практиці, це можуть бути мапа голосування по регіонам, депопуляція, зміни клімату, міграція, тощо [1].

Інформацією, яка потребує контролю за точністю, але і найбільшої швидкості є різноманітні військові дані – розташування ворога, прогнозовані точки нападу. Під час війни часто страждає цивільне населення. Тому цивільні потребують також хоча б мінімуму інформації – чи знаходяться вони зараз у небезпеці, чи ні.

Чи не найбільшою загрозою виступають обстріли артилерії та авіаудари. Найкращий спосіб перечекати ці тяжкі години у бомбосховищах. Проте, актуальність інформації про бомбосховища залишає бажати кращого у більшості міст України. Для поліпшення актуальності даних про бомбосховища та її доступності можна створити єдиний регульований простір, у якому надавати можливості підтримки консистентності даних. Такою системою може виступити інтерактивна мапа бомбосховищ, як головний засіб візуалізації інформації.

Актуальність такої роботи залишатиметься завжди, як у мирний час, так і в умовах війни. Навіть за відсутності військових дій на території держави,

така система може знадобитись і для інших загроз, наприклад стихійних лих. Адже варіантів використання такої системи безліч:

- проведення навчальних бесід із школярами, показуючи їм найближчі бомбосховища від школи чи дому;
- актуалізація даних про бомбосховища по всій країні;
- пошук найближчого бомбосховища під час артобстрілу;
- виявлення завалених чи інакше заблокованих бомбосховищ службами із надзвичайних ситуацій.

Актуальність розроблюваного додатку під час війни має особливе значення, адже він може стати в нагоді всьому цивільному населенню країни.

Об’єктом дослідження роботи є розгляд аспектів побудови інтерактивної системи мапи бомбосховищ.

Предметом дослідження виступає процес проектування та побудови інтерактивної інформаційної системи-мапи.

Метою роботи є створення інформаційної системи «мапа бомбосховищ України», доступної для загального доступу. Бомбосховища до того ж, мають статус, наявні умови, а також інші додаткові параметри.

Інтерактивність інформаційної системи впливає на сприймання інформації, за умови актуальності та затребуваності інформації.

Створено систему, яка надає можливості додавання, перегляду, редагування та видалення бомбосховищ на мапі із інтерактивною взаємодією. Перегляд даної системи-мапи доступний всім, а функціонал додавання, редагування та видалення лише дозволеним конкретним користувачам, які представляють певну установу чи організацію.

Структура роботи складається із аналізу предметної області, визначення задачі та функціональних вимог, огляду та вибору інструментів для розробки моделювання архітектури системи та планування дизайну, проектування бази даних, практичної реалізації інформаційної системи, висновків, списку літератури та додатків.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Вибір об’єкту для інтерактивної мапи

Інтерактивна мапа може мати багато застосувань. Головна мета такої системи – забезпечити актуальність даних. Прикладів, коли людям потрібна найсвіжіша інформація – безліч. Місця для паркування, вільні місця у ресторанах, наявність бензину на заправці, завантаженість мийок чи СТО, тощо.

Кожен із прикладів потребує різних даних для передачі, але головна проблема, що доволі часто інформація може бути неповною. Наприклад інформація про наявність бензину повинна містити тип бензину, скільки літрів залишилось, а також де саме знаходиться заправка і це як мінімум необхідної інформації, адже бажано ще передавати інформацію про ціну, чергу на заправці також. Але ж є газові заправки, або заправки із різними типами бензину. Це спричиняє окрему незлагодженість даних, адже їх важко уніфікувати. Такі нюанси є майже у всіх прикладах, перелічених вище.

Окремо стоїть питання комерційних мереж. Мається на увазі, що іноді певна мережа може бути незацікавлена, щоб інформація про них розміщувалась на інтерактивнім мапі. Для згаданого прикладу із заправками це означає неповну інформацію на мапі, що заважатиме користувачеві оцінити повну картину щодо ситуації із найближчими заправками.

Тим не менш, усі вище згадані приклади об’єднує важливість актуальної інформації. Якщо інформація не буде досить свіжою і точною, інформаційна система не матиме сенсу. Задля цього і використовується інтерактивна мапа – вона дозволяє швидко оцінити обстановку і прийняти рішення. Та все ж таки не враховується важливість безпосередньо інформації, окрім важливості актуальності цієї інформації. Так, наприклад, інформаційна система, яка показує який із ресторанів має вільні місця, полегшить життя користувача, але не буде надто впливовою.

Через згадані вище фактори, задля поєднання значущості інформації та важливості її актуальності було знайдено найбільш влучний варіант для інформаційної системи – мапа бомбосховищ України. Безперечно, варіант нагнітає смуток, як і сам факт необхідності такої системи. Проте користь такої інтерактивної мапи в рази вища за користь мапи заправок із свіжою інформацією про різні типи пального.

1.2 Аналіз принципів побудови інтерактивних інформаційних систем-мап

Інтерактивні мапи – це один із достатньо нових способів представлення інформації. Зазвичай, їх визначають як відображення інформації, прив’язане до певного географічного контексту, де є можливість взаємодії людини із мапою або динамічне відображення інформації.

Сьогодні є багато різних бізнес-рішень та онлайн-сервісів для створення інтерактивних мап як на умовах бізнес-для-бізнесу, так і для простого використання людиною. Такими можуть бути ArcGIS, Google Fusion Tables, Mapbox і багато інших [2]. Сервіси можуть бути розроблені як для новачків, так і для просунутих картографів, мати свої переваги і недоліки. До того ж, вхідні дані також можуть мати різний формат. Так, наприклад, сервіс CartoDB потребує географічних координат для побудови інтерактивної мапи [3].

Тим не менш, ніщо не заважає взяти звичайну географічну карту й організувати на ній відображення інформації за допомогою мови програмування.

Використання онлайн-сервісів або чужих систем може бути вигідніше, якщо потрібен єдиний підхід до організації даних, а також дані мають статистичний характер. У поставленій задачі, безпосередньою інформацією будуть геомітки, які нестимуть інформацію про певне сховище. До того ж, ці дані мають збиратися і оновлюватися регулярно, що буде не так легко

автоматизувати за допомогою подібних сервісів, а отже й актуальність даних залишатиметься під питанням.

Для створення мап із залученням мов програмування також існують сервіси, але їх мета зазвичай різна. Деякі надають можливості для максимальної кастомізації безпосередньо карти, налаштування дизайну мапи, проте мають обмежений функціонал. Інші дають можливість проведення різноманітних операцій із даними та автоматизувати їх оновлення, проте жертвують налаштуванням вигляду мапи. Важлива простота мапи, а також максимальна актуальність даних. У таких випадках можна використовувати Application programming interfaces (API) – спеціалізовані бібліотеки програмування для вирішення конкретних задач. Адже вже є інструменти, які надають можливість створювати карту й оперувати з нею.

Найкращим рішенням для поставленої задачі буде створення веб-сервісу, який надає можливості перегляду всіх доступних бомбосховищ для будь-кого, а для довірених користувачів – функціонал додавання, редагування та видалення бомбосховищ. Відображення інформації має відбуватись на інтерактивній мапі із мітками бомбосховищ. До того ж, для побудови самої мапи використовується одне із загальнодоступних API.

1.3 Огляд аналогічних інтерактивних систем

Перш за все, що варто визначити – поняття інтерактивності мапи. Зазвичай, інтерактивністю називається активна взаємодія у якомусь процесі. Для мапи інтерактивність може полягати у декількох видах. Постійне оновлення даних може видаватись за інтерактивність, бо зазвичай мапи статичні, а якщо вони активно оновлюються, з’являється динамічність у системі, відбувається певна активність, яка й сприймається за інтерактивність. Окрім цього, інтерактивність може полягати у наданні користувачу додаткових можливостей взаємодії: зменшення чи збільшення масштабу, різний вигляд мапи, активні для натиснення елементи мапи, легенда мапи, яку

можна перемістити чи сховати, тощо. Це вже більш яскравий спосіб інтерактивності, який і буде втілено у мапі. Конкретно, маркери сховищ будуть клікабельні, тобто при натисненні на них відкриватиметься окрема сторінка, а також при операціях, що змінюють сховища, маркери будуть додатковою опцією для взаємодії із користувачем. Окремо варто розглядати не просто статичні картинки мапи із додатковим функціоналом активної взаємодії, а карти, де за кожною точкою стоїть реальне географічне положення. Тобто системи, де є відповідність географічних координат із мапою, яка відображається користувачеві. Такі системи, як правило, найкраще показують можливості інтерактивної мапи і є цінними для користувачів. Оглянемо інтерактивні мапи та інформаційні системи, схожі за ідеєю для реалізації.

Найбільш поширеними у світі мапами є Google Maps. Google не позиціонує свої мапи як інтерактивні, але весь функціонал про це свідчить. Карти надають багато різноманітних можливостей. Перш за все, звичайне масштабування мапи, а також вигляд із супутника. Також є додатковий функціонал перегляду вулиць. Також на мапу нанесено багато важливих закладів, пам’яток і додаткової інформації про транспортну систему. Окрім цього, велика частина функціоналу Google Maps – це пошук адрес або конкретних локацій, а також вибудовування маршруту до обраного місця різним транспортом. Не варто забувати, що Google велика корпорація, яка активно розвивається і тому важко реалізувати щось подібне, але Google Maps це той варіант інформаційної системи, на який варто рівнятись.

Іншим сучасним, але більш простим варіантом інформаційної системи із інтерактивною мапою можна назвати Deep State – мапу війни в Україні з Росією. Важливо, що мапа містить тільки територію двох воюючих країн, а не всього світу. Основний функціонал карти – динамічне оновлення даних, щодо ліній фронту, а також позначення позицій та стратегічних пунктів ворога, реалізований за допомогою інтерактивних міток. Функціонал розширюється за рахунок інтеграції із різноманітними системами. Наприклад, розрахунок

зони ураження снаряду із заданою масою на заданій локації, а також рекомендації у такій ситуації.

Окрім вже готових рішень, існують конструктори, які дозволяють створити власні інтерактивні мапи. Такі конструктори, мають певну популярність, але їх функціонал доволі обмежений. До того ж, їх можна поділити за цільовими користувачами – це або розробники, яким пропонується специфічне API, або це звичайний інтернет-користувач, якому пропонується створити мапу на замовлення або у конструкторі власноруч.

Розглянемо один із подібних конструкторів від MapHub. У ньому ми маємо мапу світу, вигляд якої обирається за декількома доступними варіантами. Далі, користувач власноруч додає об’єкти на мапу. Це може бути лінія, може бути певна зона (виділена полігоном), може бути маркер. Із приємних переваг – маркери мають широкий спектр можливих змін – додавання фото, зміна кольору, вставлення посилання в маркер, зміна вигляду маркеру. Також, усі ці об’єкти можна редагувати і копіювати до іншої інтерактивної мапи, щоб не робити одну й ту ж роботу двічі. Користувач також може додати до мапи власний текст чи фото.

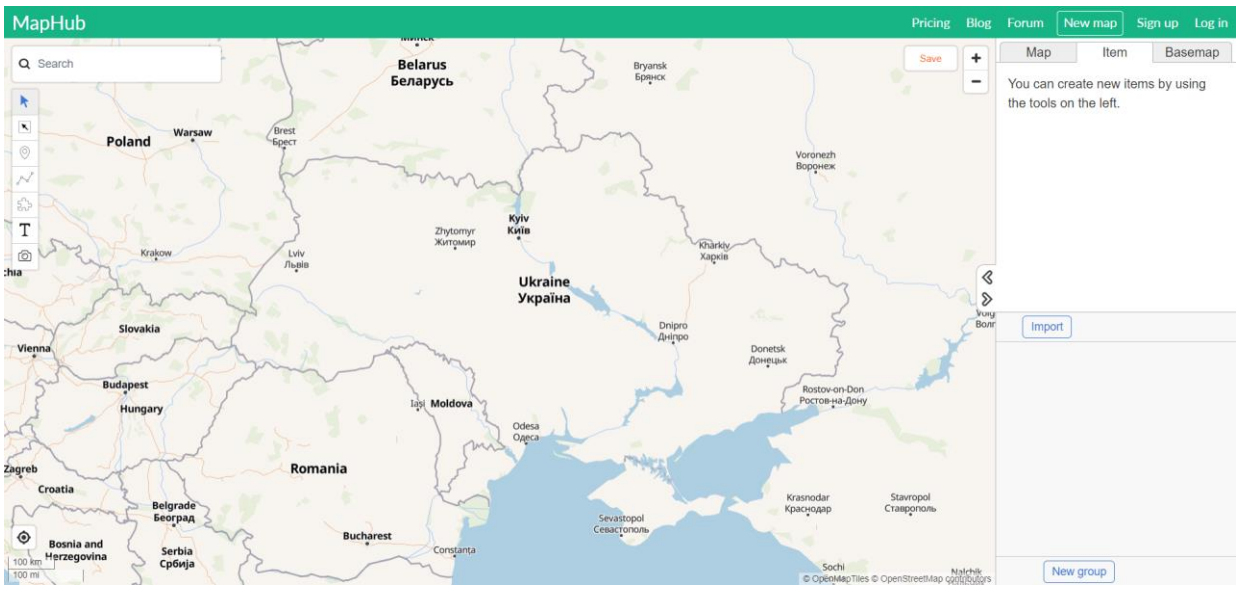

Рисунок 1.1 – Вікно онлайн-конструктору MapHub

Тим не менш, хоча увесь цей функціонал надається безкоштовно, його доволі важко використовувати на практиці. Адже немає засобу влаштувати подібні мапи до свого додатку, сайту чи ще куди небудь. У результаті ми можемо використовувати цю мапу або на сайті безпосередньо, або як статичне зображення. Такі мапи досить негнучкі і не надають достатньої інтерактивності, на відміну від інформаційних систем із мапами зроблених власноруч або за допомогою специфічного API для розробників.

1.4 Інструменти для розробки веб-додатків

Наразі кількість доступних технологій для програмування зростає із небаченою раніше швидкістю. Існує широкий вибір мов програмування, безліч бібліотек до кожної із мов та багато додаткових сервісів для полегшення роботи.

Для створення веб-додатку нам знадобляться не так вже й багато інструментів:

- мова програмування, яка забезпечуватиме бізнес-логіку веб-сервера;
- сервер бази даних, що міститиме всю інформацію щодо сховищ;
- основний фреймворк, який визначатиме архітектуру додатку та відповідатиме за оперування усіма іншими фреймворками та роботою серверу;
- додаткові фреймворки чи бібліотеки, для зв’язку бізнес-логіки та користувацького інтерфейсу, оформлення користувацького інтерфейсу;
- API для створення мапи.

1.5 Постановка задачі

У ході виконання літературного огляду нами було визначено необхідний інструментарій для роботи, розглянуто можливі варіанти реалізації та визначено переваги та недоліки вже існуючих програмних рішень. Визначили

основні потреби інформаційної системи. Отже, необхідно виконати наступні задачі:

1. Виконати огляд технологій для програмування. Обрати найкращі для веб-додатку із інтерактивною мапою з перевагою до швидкодії та простоти використання, не забуваючи про безпеку для операцій додавання, видалення та редагування.
2. Спроектувати структуру бази даних – визначити основні сутності, їх відношення між собою.
3. Виконати проектування структури додатку, обрати патерни проектування за потребою.
4. Визначити алгоритм роботи додатку, спланувати внутрішні процеси.
5. Провести тестування інформаційної системи. Проаналізувати отримані результати, зробити висновки.

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Перш за все необхідно спроектувати базу даних. Надалі при виборі подальших елементів ми вже будемо знати, із якими сутностями нам доведеться працювати.

2.1 Визначення бази даних і проектування архітектури

Оскільки ми будемо використовувати Object-Relational Mapping (ORM) технологію, тому необхідно обрати базу даних, яка підтримуватиме таку можливість. ORM – це технологія відповідності між об’єктами в об’єктно-орієнтованій мові програмування та інформації у базі даних [4].

Перевага надається реляційним БД, через простоту відображення даних, а також строгість правил зберігання даних. Одними із найбільш популярних рішень будуть: Oracle, PostgreSQL, MySQL, MS SQL Server. Розглянемо яка із них буде найвигіднішою для додатку.

Почнемо із бази даних Oracle, розглянемо її переваги та недоліки.

Oracle має високу ефективність обробки транзакцій і підтримує різні рівні ізоляції. Оскільки нам доведеться досить часто оновлювати кількість поточних людей у сховищах, ми будемо регулярно оновлювати дані. Також Oracle має внутрішню скриптову мову PL/SQL, яка розширює можливості автоматизації процесів ведення системи бомбосховищ. Oracle також має широку підтримку у подальших технологіях, через що буде легко інтегрувати її із іншими обраними інструментами.

Одним із важливих мінусів Oracle вважається складність її підтримки. Проте для невеликого за масштабами додатку, така проблема не актуальна. Також можливим аргументом є відсутність типів даних і додаткових функцій для географічної інформації, як наприклад у Postgre. Контраргументом є те, що для мапи бомбосховищ немає потреби створювати просторові зображення із координат, їх обраховувати, тощо. Фактично нам треба зберігати два числа

– широту і довготу і ці точки будуть незалежні одні від одної, через що і відпадає необхідність у додаткових функціях. Безпосередньо координати ми отримаємо із обраного API, а валідацію даних легко налаштувати на рівні додатку. Окрім широти та довготи нам знадобиться можливість зберігати числові та текстові дані, для яких Oracle має різноманітні типи даних.

PostgreSQL, як і Oracle, є потужною базою даних із високим рівнем безпеки. Особливістю є велика кількість функціональних можливостей, включаючи підтримку геоданих та створення функціоналу через власні розширення. Важливою перевагою є відкритий код БД, а також безкоштовність використання. Важливим недоліком PostgreSQL є складність налаштування, а також слабка продуктова підтримка, порівняно із Oracle. До того ж, усього переліку функцій для роботи із геоданими нам непотрібно, адже ми тільки зберігаємо географічні координати і ніяк не оперуємо ними. Збереження координат у вигляді пари дробових чисел легко вирішується на рівні мови програмування.

MySQL є безкоштовною і має відкритий код як і PostgreSQL, має спрощений синтаксис SQL, а також легко налаштовується. Головною проблемою є досить вузький функціонал, на фоні вищезгаданих Oracle та PostgreSQL. До того ж, є доволі суттєві обмеження в обсязі даних, які можна зберігати. MySQL досить проста у використанні, але не підходить у випадку, якщо дані будуть зростати при розширенні додатку.

MS SQL Server підтримує велику кількість платформ, а також вбудовані інструменти для роботи із даними в операційній системі Windows. Окремою перевагою є функціональні можливості, однією із яких є підтримка даних JSON формату. Основні недоліки це мала розповсюдженість та необхідність ліцензії. До того ж, у веб-додатку JSON використовуватиметься тільки для передачі даних і не потребуватиме зберігання. А для операцій із даними буде використовуватись ORM, тому необхідності в додаткових інструментах MS SQL Server немає потреби.

Серед усіх баз даних, оглянутих вище, найбільш під потреби додатку підходять Oracle та PostgreSQL. Тим не менш, із ціллю подальшого розвитку додатку, більш вигідним варіантом є Oracle. Переваги, яка надає PostgreSQL можуть бути отримані на рівні додатку, а не бази даних. Отже, у якості бази даних обираємо Oracle, який цілком задовольняє потреби інформаційної системи.

2.2 Мова програмування

Далі перейдемо до вибору мови програмування, якою буде написано бізнес-логіку додатку. Кожна мова програмування створена для певних цілей, а також має свої переваги та недоліки. Для розробки підійде мова програмування, яка має потенціал для створення веб-додатків й інтегрується із базою даних Oracle, до того ж бажано об’єктно-орієнтована і підтримує концепції об’єктно орієнтованого програмування (ООП), оскільки необхідно працювати із визначеними об’єктами – бомбосховищами, користувачами веб-додатку, тощо.

Варіантами мов програмування, що підтримують концепції ООП, будуть Java, C++, C#, Python, Ruby, PHP. Коротко розглянемо, яку мову програмування варто вибрати для реалізацій поставлених цілей.

Java досить відома і стара мова програмування. Основне у цій мові те, що вона об’єктно орієнтована, тому розробка системи із інтерактивною картою повинна пройти без складнощів та підводних каменів. Важливо відмітити, що Java підтримується і до сьогодні, хоча і не всі версії. Найчастіше Java використовується для створення Enterprise додатків, але ніщо не обмежує її для створення невеликих за розміром веб-ресурсів. До того ж Java має багато вже існуючих бібліотек та можливостей для інтеграції із додатковими технологіями, які можуть знадобитись під час розробки. Окремо варто відмітити потенціал Java для розробки мобільних додатків під Android. Мовою Java написано безліч різноманітних мобільних додатків, тому якщо буде

необхідність розробити окрім веб-ресурсу ще й мобільний додаток із інтерактивною мапою, то можна буде використовувати вже створену модель.

C++ – є об’єктно орієнтованим розширенням мови C. Перевагами C++ є можливість створення додатків із великою швидкодією та гарною оптимізацією використовуваних ресурсів. Проте, це не ті переваги, які необхідні для веб-додатків. До того ж, ООП на C++ часто ставиться під сумнів, адже діє за власними правилами.

C# – це вже однозначно об’єктно-орієнтована мова програмування, на відміну від C++. Основною проблемою C# це розробка під платформу .NET і відсутність повноцінної кроссплатформеності. Але це ж є і перевагою C# - розробка може бути універсальною і можна розробляти веб-сайти, ігри, мобільні додатки, тощо. До того ж Microsoft, яка створила C#, постійно оновлює і доповнює мову програмування.

Python – це високорівнева об’єктно-орієнтована мова програмування, яка широко використовується у багатьох галузях програмування. Python часто стає вибором для створення простих додатків, адже мову досить легко читати і підтримувати код. Важливими перевагами Python є відкритий вихідний код, платформонезалежність, динамічна типізація та велика кількість вже готових бібліотек для будь-яких цілей. Тим не менш, є і недоліки: прожерливість до пам’яті через динамічну типізацію, відсутність повної підтримки багатопоточності, а також проблеми під час вибудовування архітектури.

Наступний варіант мови програмування – Ruby, яка вважається досить нішевою мовою, через свої особливості, які у різних ситуаціях можуть виступати як плюсами, так і мінусами. Ruby дуже продумана мова програмування, через що вона іноді буває надмірною. Суттєвими недоліками є повільність виконання коду Ruby, а також відсутність потенціалу для створення великих корпоративних додатків. До того ж, код на Ruby іноді досить важко читати, що ускладнює підтримку.

PHP – це один з найпоширеніших інструментів для веб-розробки, що використовується для створення динамічних веб-сайтів та веб-додатків.

Однією з основних переваг PHP є його відкритий код, що робить його доступним для вільного розширення функціоналу[5]. Однак, є і деякі недоліки використання PHP. PHP може бути дещо повільним в порівнянні з іншими мовами програмування. Крім того, недостатність безпеки є однією з основних проблем, з якими можуть зіткнутися веб-розробники при використанні PHP[5]. Незважаючи на ці недоліки, PHP залишається дуже популярним інструментом для веб-розробки, зокрема завдяки великій кількості готових рішень і фреймворків, які роблять розробку додатків на PHP продуктивною і ефективною.

Отже, варіантів багато, але пріоритет стоїть на потенціал для розробки веб-додатку, наявність бібліотек для роботи із мапами, підтримка ORM для роботи із БД, а також наявність певної безпеки або можливість її налаштування. За цими параметрами найбільш влучним вибором буде Java, яка має велику підтримку спільноти, розвивається й сьогодні, а також довела свою надійність роками.

Таким чином, мова програмування Java може бути використана для створення веб-додатку без будь-яких проблем.

2.3 Основний фреймворк чи архітектурне рішення

Наразі вже давно відомо, що краще використовувати вже створені принципи програмування при розробці будь-чого, адже при виявленні проблем у продукті їх можна легко вирішити використовуючи досвід інших. До того ж, використання патернів програмування чи архітектурно впливових фреймворків спрощує сам процес розробки, бо вже буде певний план реалізації додатку.

Для мови програмування Java окремо виділяється Spring фреймворк, який фактично є фреймворком над фреймворками. Адже окрім архітектурного рішення Spring пропонує легкий спосіб інтеграції із іншими бібліотеками та фреймворками.

Spring Framework – це один з найбільш популярних інструментів для розробки веб-додатків на мові Java. Він надає широкий набір функціональності для розробки як простих, так і складних веб-додатків, таких як взаємодія з базами даних, обробка HTTP-запитів, безпека, тестування, інтеграція з іншими інструментами та багато іншого[6].

Spring надає зручний інтерфейс для роботи з іншими популярними фреймворками, такими як Hibernate для роботи з базами даних або Struts для керування веб-інтерфейсом. Це дозволяє розробникам ефективно використовувати функціональність, яку надає Spring Framework, з іншими інструментами, що забезпечує більшу гнучкість і ефективність у розробці веб-додатків.

Завдяки своїй популярності та гнучкості, Spring Framework використовується великою кількістю компаній і проектів для розробки веб-додатків на мові Java. Він є потужним інструментом, який допомагає зменшити час розробки і полегшити процес розробки веб-додатків.

Через свою архітектурну значущість, Spring має особливості роботи у середовищі Java. Перш за все, Spring надає контейнер для додатку із можливістю ін’єкції залежностей. Окремо варто пояснити, що ж таке ін’єкція залежностей. Це підхід до розробки програмного забезпечення, при якому об’єкти отримують залежності зовні, замість самостійного створення[6]. Такий підхід дозволяє розділити логіку створення об’єктів та логіку використання цих об’єктів, задля полегшення розширення коду та тестування.

Повернемось до spring контейнера, який створює контекст програми, де відбувається керування класами та залежностями. У рамках цього контексту будуть існувати біни (bean), які міститимуть екземпляр певного класу. Із цієї структури випливає зручне використання будь-яких додаткових залежностей, багато з яких існують у Spring начебто окремо, але використовувані тільки при інверсії контролю у Spring контексті.

2.4 Додаткові фреймворки та бібліотеки

Різноманітні бібліотеки потрібні для різноманітних задач, які вже вирішені іншими програмістами, а тому економлять час на розробку додатку. Як було згадано, Spring має багато модулів, із яких ми використовуватимемо Spring Data JPA для роботи із базою даних, Spring Security для безпеки даних і веб-додатка, Spring Web MVC для організації структури веб-частини додатку, а також Spring Validation для перевірки даних, які приходять від користувачів і йдуть до бази даних.

Але є і бібліотеки, які необов'язково є частиною Spring. Із подібних технологій ми будемо користуватись Thymeleaf для забезпечення обміну інформації між frontend та backend частинами, а також Bootstrap для візуального оформлення UX/UI дизайну.

2.5 API для мап

Основною можливістю додатку має бути взаємодія із мапою. Але на створення мапи, її вигляду, організацію роботи із нею піде багато часу. Тому, через існування загальнодоступних API ми можемо значно спростити процес створення мапи у рамках додатку і зосередитись безпосередньо на веб-сайті.

Одним із найпопулярніших інструментів є Maps JavaScript API від Google. Maps API використовує знайомі багатьом карти від Google[7]. До того ж, завдяки його популярності ми зможемо знайти багато додаткової інформації по створенню та використанню мапи.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ДОДАТКУ

Для практичної реалізації було розроблено функціональну модель, спроектовано базу даних, а згодом і дизайн додатку. Після реалізації розробленої архітектури додатку, інтерактивна мапа бомбосховищ була готова до роботи.

3.1 Приклади використання

Перш за все, розглянемо Use Case діаграму (рис 3.1). Вона розповість нам, як саме може бути використано додаток.

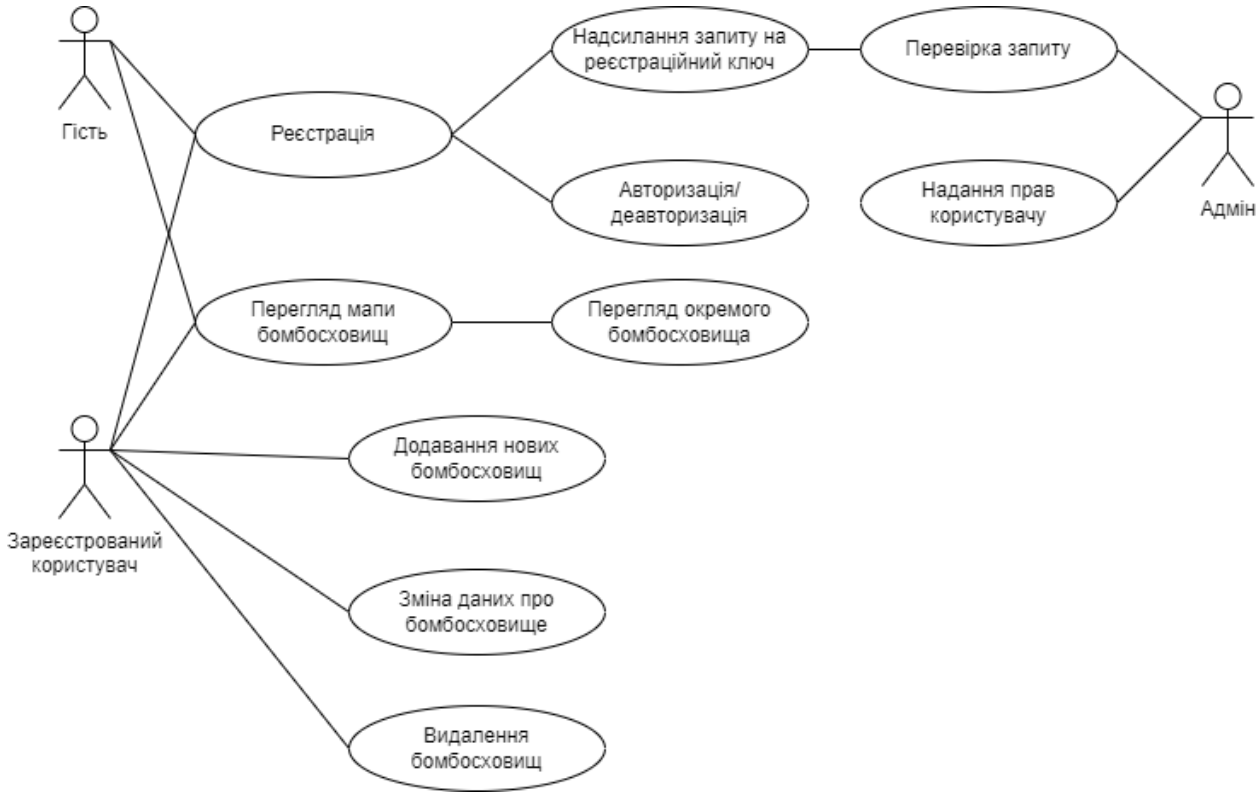

Рисунок 3.1 – Use Case діаграма інформаційної системи

Загалом, функціонування системи відбувається без особливого нагляду. Лише ключ для реєстрації необхідно передавати вручну, що зроблено із метою безпеки. Користувач надсилає свої дані через додаток для отримання ключа

доступу. Дані відправляються додатком на пошту адміністратора. Далі вже за рішенням адміністратора ключ може бути надіслано на пошту користувача, для завершення процесу реєстрації. Таку систему зроблено із ціллю обмеженого доступу до функціоналу, що змінює стан бази даних, який має надаватися людиною особисто.

Усі інші варіанти використання системи вже працюватимуть автоматично. Перегляд бомбосховищ не потребує жодної підтримки, а операції додавання, змінення та видалення бомбосховищ має подвійну валідацію даних. Дані спочатку перевіряються на frontend частині, де показується помилка, яке саме поле некоректне або яка помилка сталась. Далі після відправки на сервер інформація знову перевіряється, і у випадку невдачі користувач автоматично переходить на сторінку помилки. Сторінка містить опис помилки у випадку, якщо це звичайна помилка із описом. У іншому випадку сторінка помилки повідомить коротку відомість, але без точної інформації задля збереження конфіденційності окремих параметрів додатку.

3.2 Архітектурне рішення

Основна архітектура додатку підпорядковується Spring MVC, а саме розділенню обов'язків між моделлю, виглядом та контролером [8]. Концепт Web MVC можна побачити на рисунку 3.2.

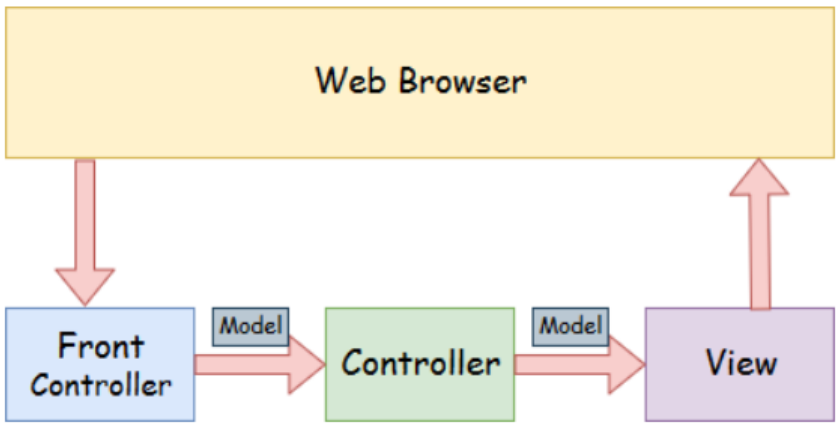

Рисунок 3.2 – Концепт Web Model-View-Controller [8]

Модель у схемі відповідає за всі дані, які містяться у додатку. Тобто «модель» тільки передається між відповідними частинами web-додатку і у кінці, за потреби, може бути збережена у БД. View слугує представником інформації у певному форматі [8]. За даного випадку, ми передаємо HTML сторінки, оформлені згідно певних CSS файлів та із підключеними JavaScript файлами, для забезпечення інтерактивності додатку. Front Controller отримує дані від користувача і додатково оброблює їх, передаючи до контролеру HTTP запит (або лише дані із запиту, необхідні контролеру). Controller, у свою чергу, повністю відповідає за бізнес-логіку додатку, опрацьовуючи дані [8]. До того ж, ніщо не вимагає, щоб був лише один контролер, обов'язки можна розділити.

Загальний принцип роботи MVC паттерну визначено, можна розібратись у деталях реалізації.

Із окремих шарів додатку можна виділити сервісний шар, де відбувається обробка на Java-рівні, а також репозиторії, які працюють безпосередньо із базою даних [9].

Сервісні шари оперують загальними процесами. Так, зазвичай саме там відбуваються усі важливі розрахунки, окремі налаштування, маніпуляція із Java-об'єктами. У додатку визначено декілька сервісів. Окремий шар для надсилання повідомлення із запитом на реєстрацію від користувача. Окремий сервіс для контролю реєстрації, який забирає дані із контролера, організовує загальну модель сховища, та записує його у базу даних. Ще один сервіс для створення користувачів та їх авторизації. А також сервіс для сховищ, який допомагатиме контролеру отримувати дані про сховища, а також зберігати їх, змінювати та видаляти.

Контролери відповідають за взаємодію із користувачем за певними правилами, але не повинні виконувати бізнес-логіку. Зазвичай, контролери ділять за тією групою функціоналу, за яку вони відповідають. У додатку наявні контролер для користувачів, контролер для сховищ, REST контролер для сховищ, а також контролер порадник, який відповідає за обробку помилок.

Але є ще окремий шар для роботи із базою даних. Ми використовуємо специфікацію JPA, яка визначає правила взаємодії Java додатку і бази даних. Конкретно використовується реалізація JPA Hibernate, який автоматично визначено у Spring.

Обговоримо також модель, яка використовуватиметься для передачі даних в інформаційній моделі. Найголовніші її елементи – сховище, користувач та ключ реєстрації. Усі інші класи моделі – частини головних елементів або ж їм підпорядковуються. Частинами моделі сховища є умови, які описані окремим списком можливих значень, статус, який так само має список можливих значень, а також координати, адже такої сутності немає у Java «із коробки», а додаткові бібліотеки нам непотрібні, якщо ми можемо легко створити їх самі. Частинами користувацької моделі виступають юзер ролі. Частинами моделі, які залежні від ключа доступу, є запит на ключ доступу та реєстраційний запит.

Варто зауважити, що робота із картами відбувається саме у Frontend-частині додатку у зв’язку із особливостями роботи Google Maps JavaScript API, обраного для відображення мап. Але оскільки API не споживає багато ресурсів, це не спричиняє проблем ні клієнту, ні серверу.

Таким чином, було визначено архітектуру інформаційної системи, спираючись на концепти побудови додатків, що використовують Spring Framework. Лістинг коду, який відповідає за архітектуру веб-сервісу та іншу бізнес-логіку можна побачити у додатку А.

3.3 Проектування бази даних

На вдачу, інформаційна система-мапа не надто складна систему у проектуванні. Основні сутності це сховища та користувачі. У реляційних БД дані знаходяться у таблиці, які мають декілька полів для даних. Між таблицями може бути створено зв’язок, який допомагає підтримувати консистентність інформації у базі даних. Для бомбосховища потрібно

зберігати координати, площу, місткість, статус сховища, додаткову інформацію, а також наявні умови у сховищі.

Координати можна зберігати у якості двох чисел, максимальні значення яких будуть обмежуватись та перевірятись вже на рівні додатку. Тому для координат достатньо створити два поля Latitude та Longitude – широта і довгота. Для додаткової інформації, місткості, статусу та площі створимо також поля відповідного типу. Oracle має всі необхідні нам типи даних: цілі числа, текстові дані, дробові числа, тощо. Як первинний ключ, за яким можна визначати сховище можна поставити складений первинний ключ із довготи та широти, адже їх комбінація для кожного сховища тільки одна й два сховища на одній точці існувати не може.

Окремим питанням є умови, які наявні у сховищі. Перш за все, у рамках додатку існує певний перелік усіх можливих умов, але на сховищах необов’язково будуть всі із списку, може бути декілька, а може бути й жодного. Через те, що умов може бути декілька, стає неможливо зберігати їх у рамках однієї таблиці, або ж таблицю доведеться денормалізувати, що погано позначиться на консистентності даних та легкості підтримки. Задля вирішення цієї проблеми необхідно створити окрему таблицю із умовами для кожного сховища із зовнішнім ключем таблиці сховищ. Таким способом у таблиці із умовами буде декілька записів для різних умов одного сховища.

Варто згадати також те, що Oracle надає можливість виставлення точності для значень із плаваючою точкою. При виставленні певної точності Oracle автоматично округлюватиме числа після коми[10]. Для вирішення поставленої задачі, сховища повинні мати чітку позицію на мапі, тому збереження кожної цифри після коми важливе. До того ж, необхідно залишити стандартні значення при створенні таблиці й виділити побільше пам’яті для значень координат.

До цього моменту вже було реалізовано окрему логіку реєстрації користувачів. У поставленій задачі існує три категорії користувачів: неавторизовані користувачі, неавторизовані користувачі, які подали заявку на

реєстрацію, а також повістю зареєстровані користувачі. Для першої категорії у базі даних місця немає, вона повністю регулюється за допомогою Spring Security. Інформацію від другої групи вже потрібно зафіксувати. Як мінімум потрібно зберегти введену пошту, згенерований ключ, який підходитиме тільки до цієї пошти, а також пароль, введений користувачем. Потреби у контактній інформації із заявки немає, вона буде напряму відправлена поштою і не потребує зберігання. І для авторизованих користувачів має бути окрема таблиця із полями, необхідними для Spring Security. Вона має містити логін, пароль, значення чи не заблоковано акаунт, значення чи активовано акаунт і роль акаунту. Усі ці параметри надають можливість керувати акаунтами без їх видалення. Роль акаунту надає широкі можливості для масштабування додатку, адже можна створити нові ролі окрім зареєстрованих користувачів, наприклад для системних адміністраторів або для користувачів із обмеженими правами. У результаті для організації безпеки маємо три таблиці. Таблиця незареєстрованих користувачів, але які подали заявку, таблиця зареєстрованих користувачів, а також пари пошта-ключ, які потрібні в момент надсилання листа поштою для довіреної особи.

Розглянемо Entity Relations (ER) діаграму бази даних (рис 3.3).

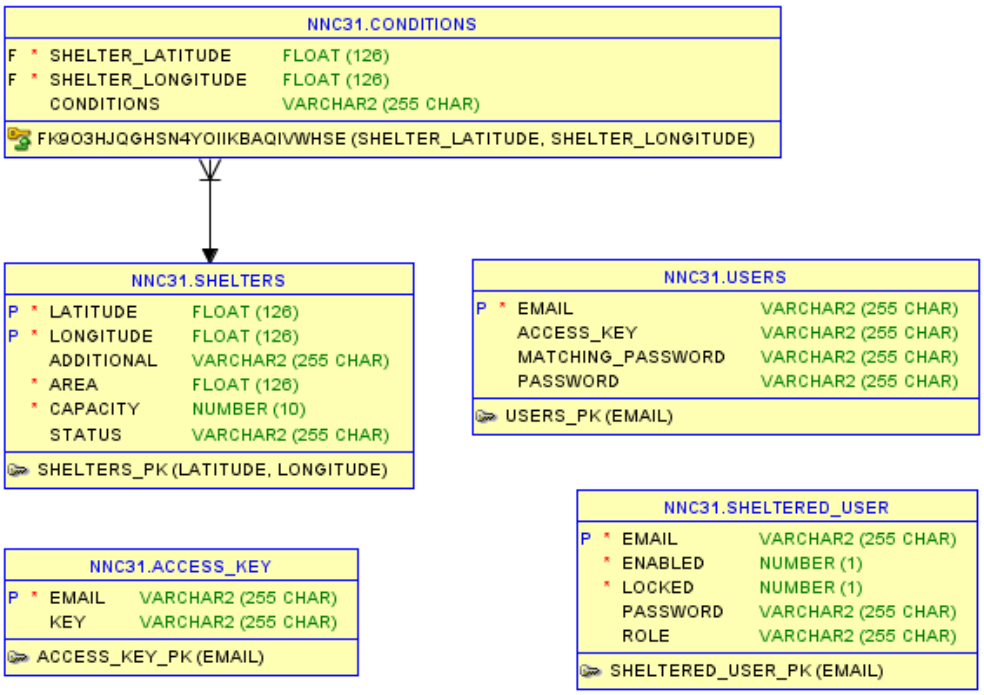

Рисунок 3.3 – ER діаграма бази даних

ER діаграма необхідна для пояснення логічної структури бази даних (БД) і відображає взаємозв’язки сутностей [11].

Основна таблиця – Shelters, у ній зберігається складений ключ із широти та довготи, а також вся інформація про сховище. Тільки умови, наявні у сховищі винесені в окрему таблицю, оскільки це множинний атрибут.

Інші таблиці створені для керування безпекою. Таблиця із наявними ключами доступу у прив’язці до пошти, користувачі для real-time контролю безпеки, а також користувачі, що статично збережені в БД [12].

3.4 Дизайн додатку

Перейдемо до дизайну додатку. Оформлення виконане загалом із використанням стандартних можливостей Bootstrap фреймворку, але іноді із застосуванням окремих бібліотек, сумісних із Bootstrap, які надають додаткові, але важливі можливості для дизайну.

Загальний дизайн дотримується чорно-білих тонів, зроблений у єдиному стилі. Для деяких сторінок використовується Java Script, щоб сторінки мали інтерактивність, Frontend валідацію, а також для генерації мапи.

Для генерації мапи використовується Google Maps API, яке запускається у Java Script із переданими параметрами. До параметрів належить масштаб мапи, початкова центральна точка. Маркери на мапі з’являються вже після попередньої генерації мапи, для них окремо повинні бути визначені координати та супутні параметри. Супутніми параметрами можуть бути іконка маркера, випадне меню маркера, окрема дія при натисканні на маркер, розмір, тощо. Окремо у скрипті, де ініціалізується мапа, можуть бути задані додаткові дії, наприклад програвання звуку при натисканні або завантаження потрібного сховища при натисканні. Для кожної сторінки скрипт створення мапи свій, адже для сторінки створення нового сховища нам не треба буде змінювати вже існуючі маркери, тобто сховища. А ось для сторінки редагування сховища

потрібно додати можливість динамічного завантаження даних про сховище без оновлення сторінки.

Розглянемо рисунок 3.4 «Головна сторінка» додатку.

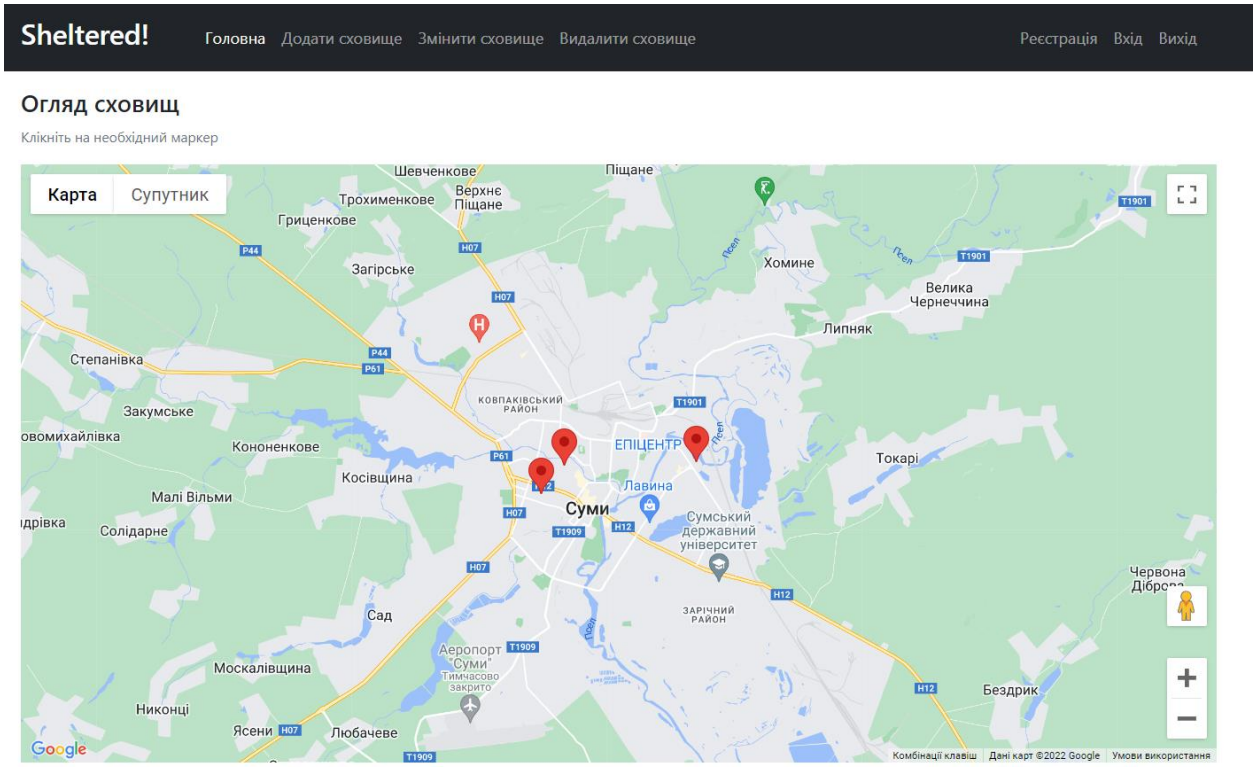

Рисунок 3.4 – Головна сторінка Sheltered

Як видно, доступ до функціоналу знаходиться в навігаційному меню. Через нього можна потрапити до всіх необхідних сторінок: реєстрація, операції зі сховищами, вхід та вихід із акаунту.

Розглянемо детальніше мапу, яка займає основну частину контенту. Це досить відома стандартна Google maps карта, наразі для тесту вона відцентрована по місту Суми. Використовуючи колесо миші або відповідні кнопки у правому нижньому кутку можна зменшити або збільшити масштаб. У верхньому лівому кутку можна перемкнути вигляд мапи на вигляд із супутника, при якому функціонал маркерів зберігається (рис. 3.5). Навпроти перемикача вигляду є також кнопка для входження у повноекранний режим. Також на рисунку 3.6 можна побачити як працює перегляд вулиць. При

перегляді вулиць мапа змінюється виглядом тієї вулиці, куди перетягнуто іконку чоловічка. Якщо маркер знаходиться на цій вулиці – його також можна активувати і він виконає запрограмовану дію.

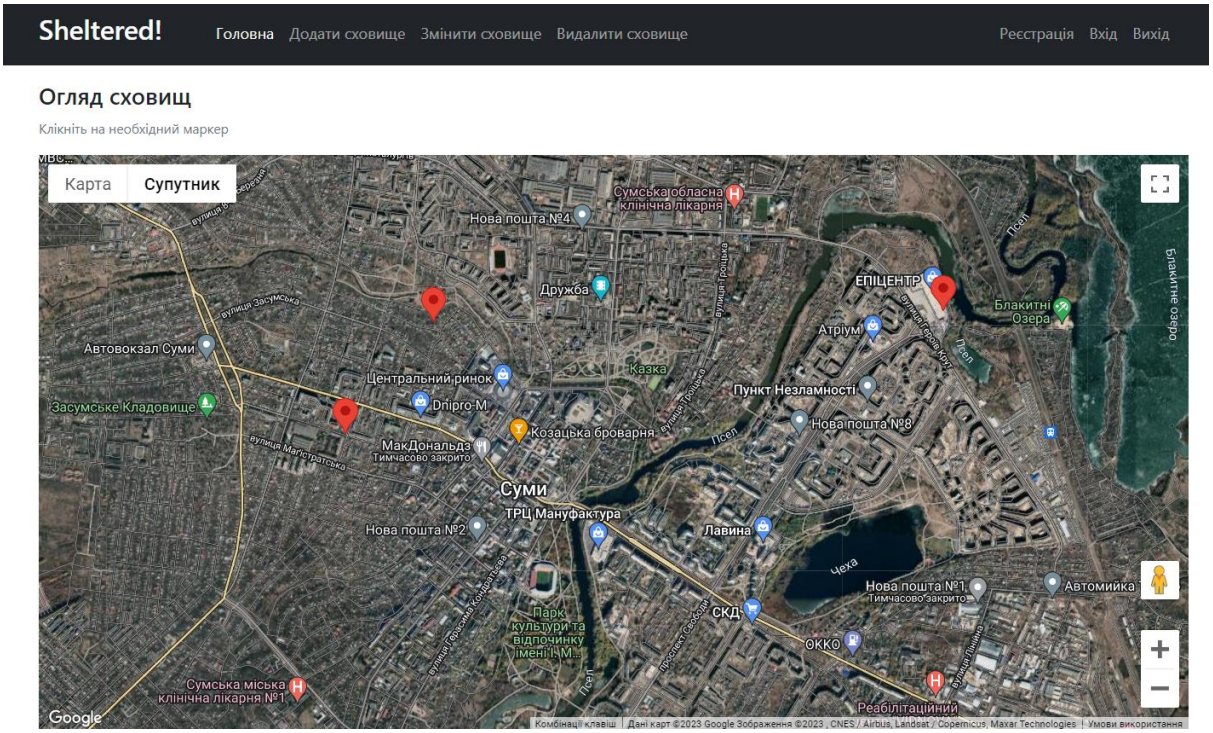

Рисунок 3.5 – Мапа Sheltered при перегляді із супутника

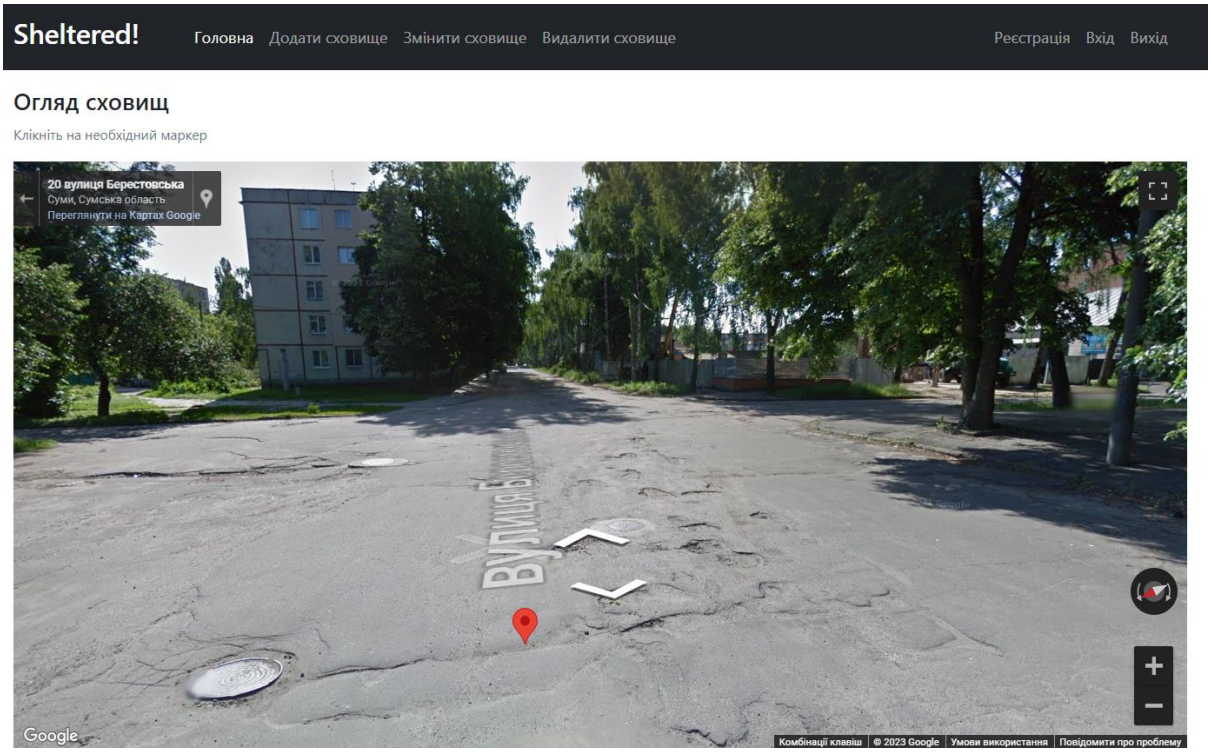

Рисунок 3.6 – Перегляд вулиць у Sheltered

Розглянемо принцип роботи мапи на головній сторінці. Наявна можливість побачити мапу із мітками-сховищами на ній. За підказкою над мапою, можна натиснути на маркер, щоб переглянути інформацію про необхідне сховище. Мапа працює завдяки Google Maps JavaScript API, тому має усі функції доступні у рамках прикладного програмного забезпечення [13].

Розглянемо вигляд сторінки окремого сховища (рис. 3.7)

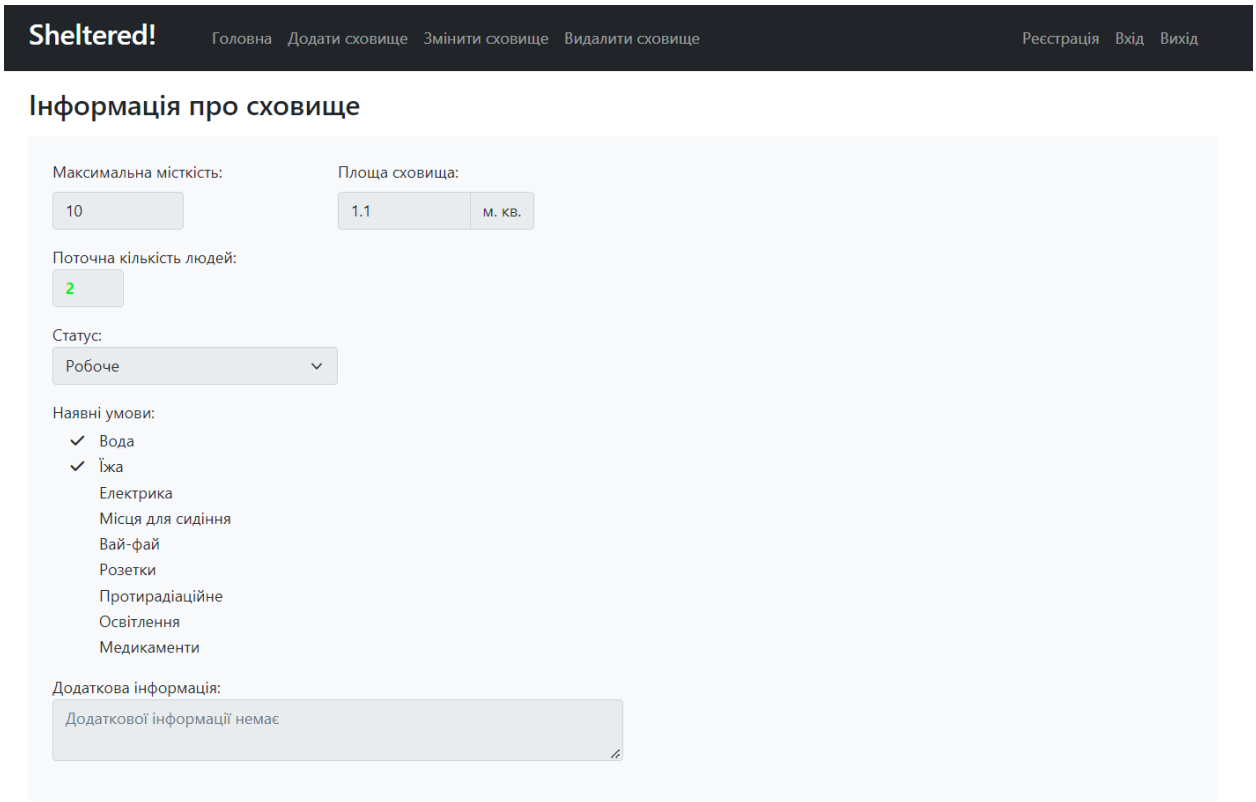

Рисунок 3.7 – Сторінка сховища Sheltered

Як бачимо, тут відображені усі параметри сховища, яких не видно на мапі. Наявні умови відмічені галочкою зі списку усіх можливих умов. Показано статус, додаткова інформація, якщо така наявна), а також місткість та площа. Також відображається поточна кількість людей, пофарбована у відповідності із відношенням кількості людей до максимальної місткості. Чим більший відсоток сховища буде заповнений, тим далі змінюватиметься колір.

Наразі оновлення інформації на таких сторінках не відбувається автоматично після оновлення сховища, тільки після оновлення сторінки

безпосередньо. Для подальшого покращення, можна було б використовувати технологію AJAX, за допомогою якої оновлювати конкретні поля без взаємодії користувача та системи. Таким чином вдалося б підвищити інтерактивність додатку та зменшити навантаження на сервер, адже не довелося б перезавантажувати всі інші параметри сховища після оновлення [14]. Єдиною проблемою, що за великої кількості користувачів довелося б оновлювати дуже багато сторінок після оновлення сховища.

Розглянемо авторизацію та деавторизацію на сайті перед функціоналом редагування сховищ. На рисунку 3.8 можна побачити поля, які необхідні для реєстрації користувача, через якого можна буде редагувати сховища. Нам потрібні пошта, пароль, а також ключ доступу. Якщо ключ доступу ще не отриманий користувачем, то необхідно натиснути на синій клікабельний текст над кнопкою реєстрації, яка веде до сторінки подачі запиту на ключ доступу (рис. 3.9)

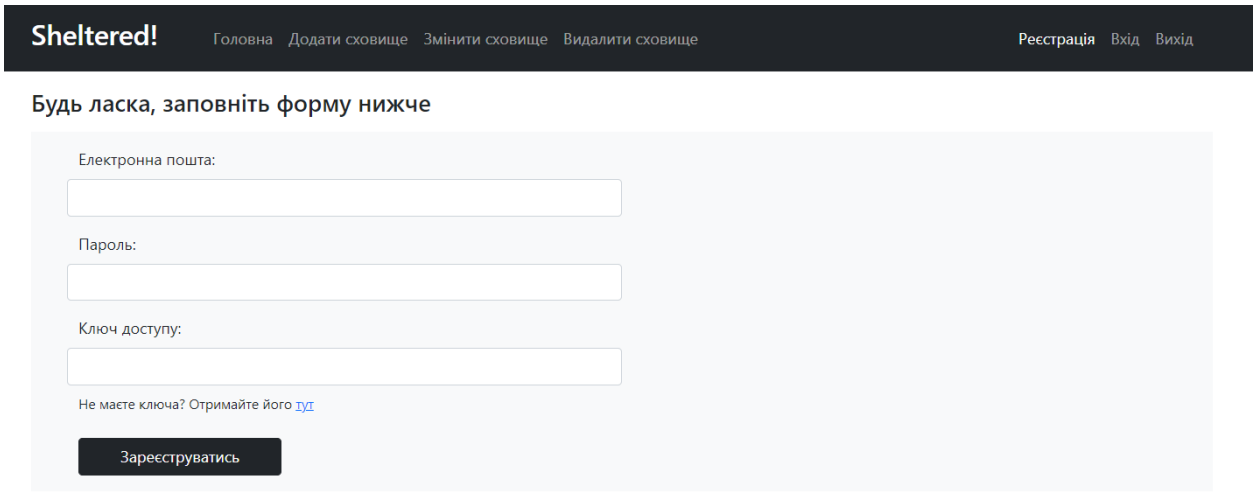

Рисунок 3.8 – Сторінка реєстрації

Ключ доступу потрібен, щоб операція реєстрації контролювалася людиною, а не проводилась автоматично. Після надсилання запиту на ключ доступу, повідомлення надходить на пошту уповноваженої особи, яка вже вручну перевіряє інформацію, хто запитує про реєстрацію на сайті.

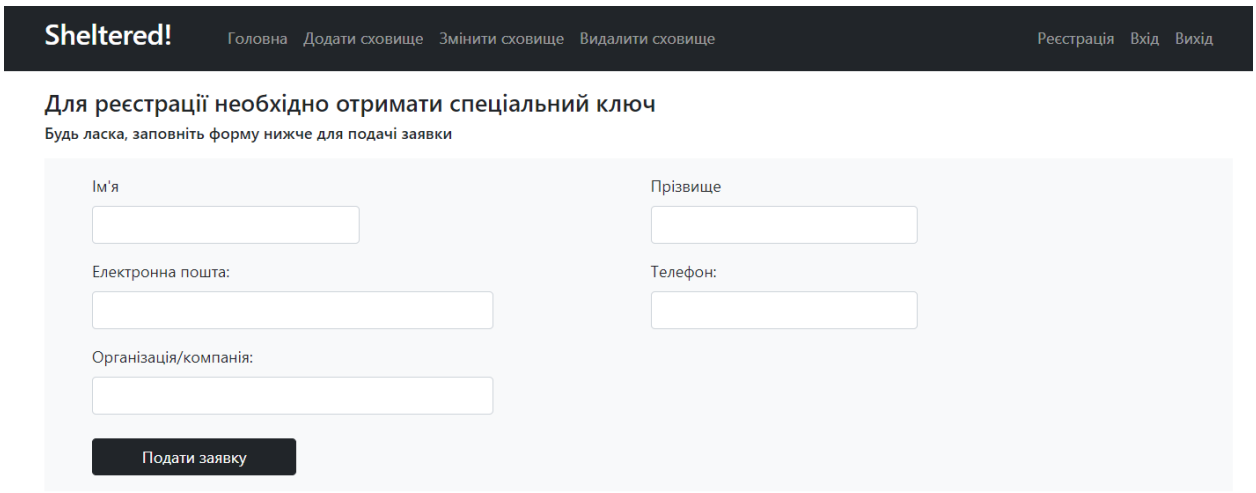

Рисунок 3.9 – Сторінка заявки на ключ доступу до Sheltered

Сторінка подачі запиту на ключ доступу потребує контактних даних особи, яка намагається отримати доступ. При натисненні кнопки «Подати заявку» уся введена інформація перевіряється на Frontend частині, а далі вже надсилається від імені додатку адміністратору, чия пошта зашита у конфігураційні параметри додатку, тому може бути змінена будь-коли. Під час надсилання пошти також генерується ключ доступу, який адміністратор вже може надіслати особі, яка запитувала доступ. На рисунку 3.10 можна побачити приклад такого листа.

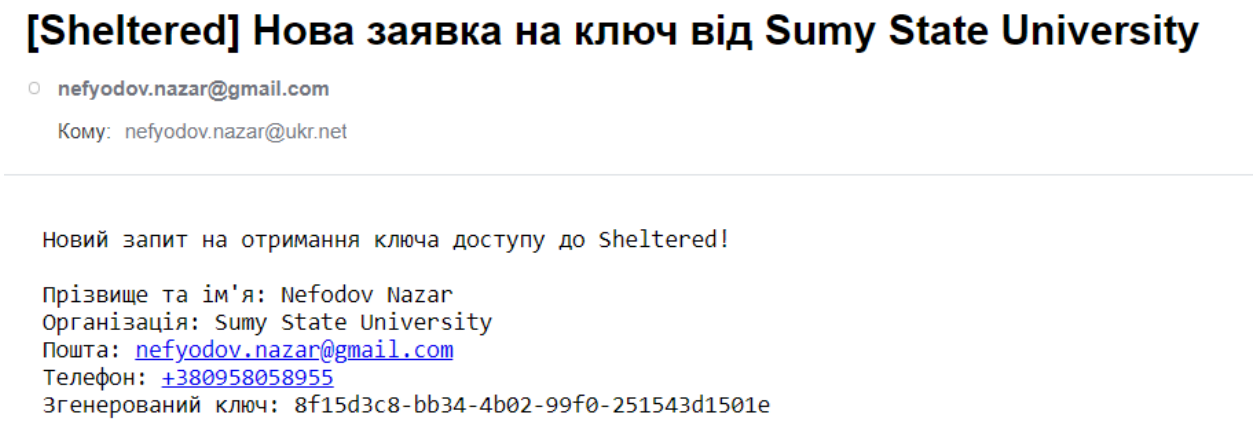

Рисунок 3.10 – Лист на запит ключа доступу від Sheltered

Сторінка авторизації (рис. 3.11) не має особливих полів, тільки пошту та пароль і кнопку входу. Звісно, при неправильно введених даних сайт повідомляє про помилку банером під навігаційним меню.

При натисканні на кнопку виходу, користувача пересилає на сторінку авторизації, із повідомленням про успішний вихід (рис. 3.9). До того ж, спроба перейти на сторінки додавання, зміни чи видалення сховища також перенаправляються до сторінки авторизації, якщо користувач ще не увійшов до акаунту.

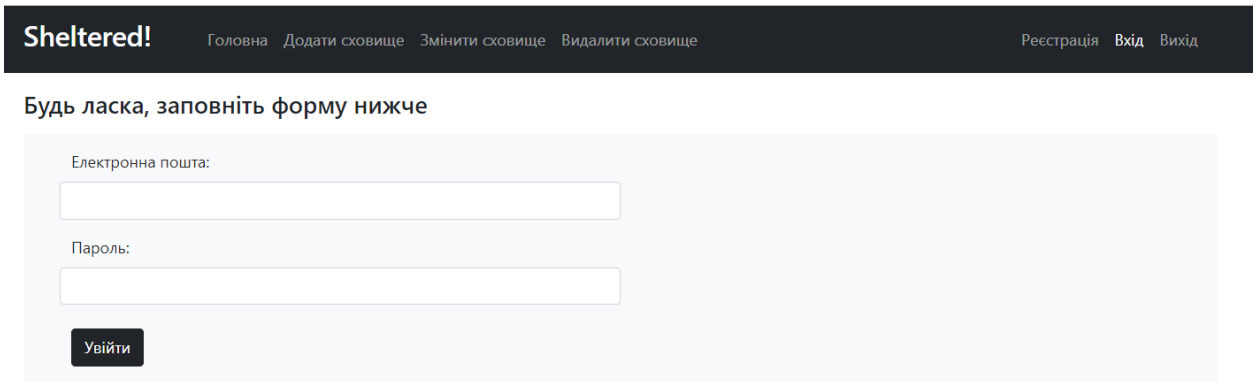

Рисунок 3.11 – Сторінка авторизації Sheltered

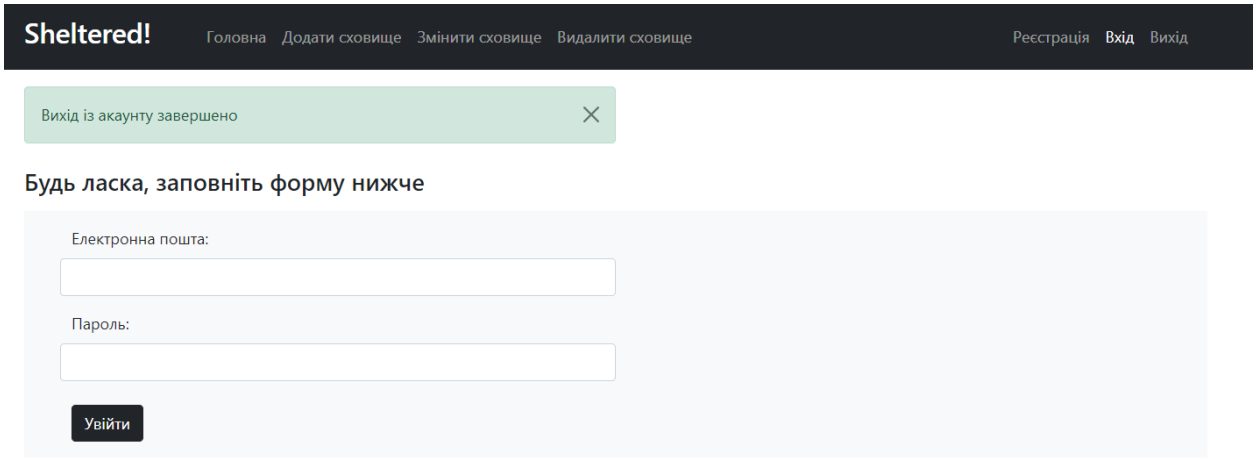

Рисунок 3.12 – Вихід із акаунту в Sheltered

Перейдемо до важливої частини функціоналу додатку – операцій додавання, видалення та зміни сховищ. Сторінки схожі за дизайном, на кожній

із них є інтерактивна мапа, на якій можна визначати предмет операції: маркер із можливістю перетягування для нового сховища, клікабельні маркери для вибору сховища для видалення чи для зміни. Усі інші поля вже залежать від операції. На рисунку 3.13 видно усі поля необхідні для заповнення інформації про сховище. При введені неймовірних значень, наприклад площі менше нуля, або інших помилок вводу з’являється повідомлення про помилку у відповідному полі після натискання кнопки додавання сховища (рис. 3.14).

Для встановлення координат нового сховища, потрібно перетягнути маркер із плюсом на потрібну точку. Після кінця перетягування лунає тихий клік для додаткового комфорту.

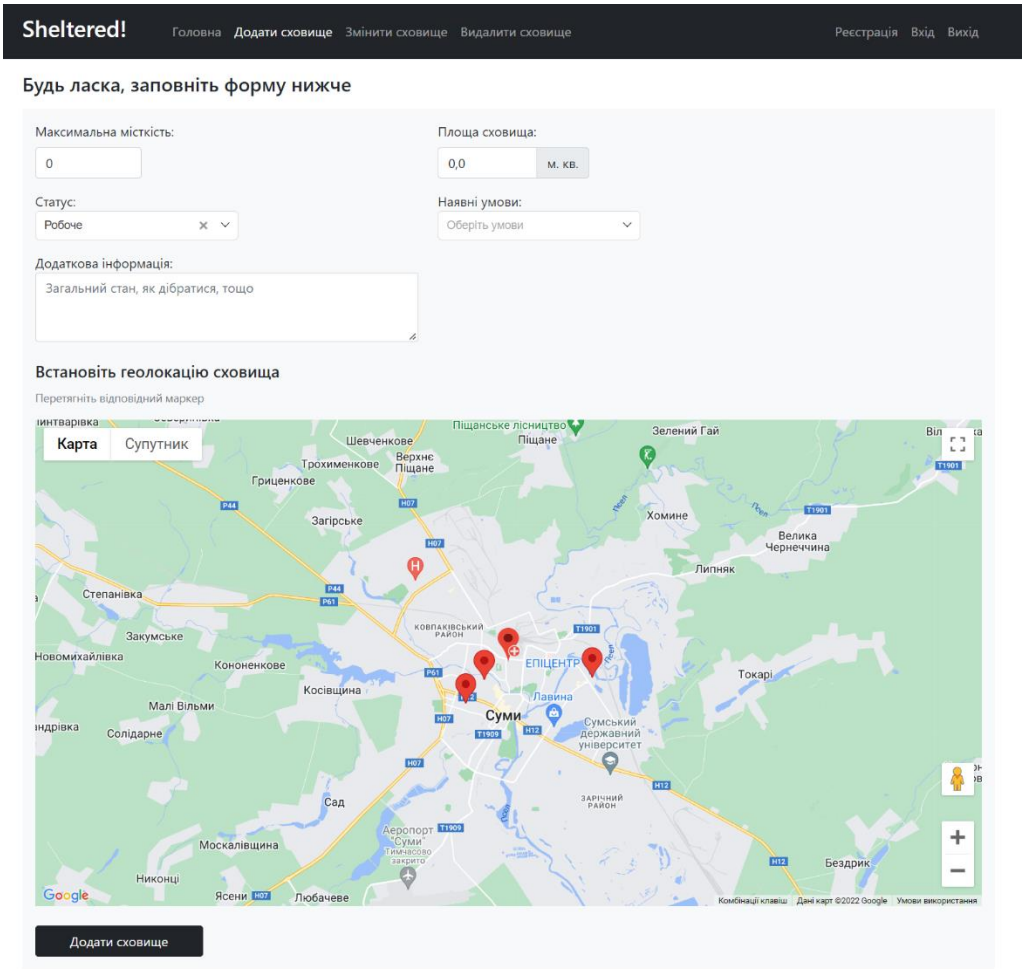

Рисунок 3.13 – Сторінка додавання нового сховища Sheltered

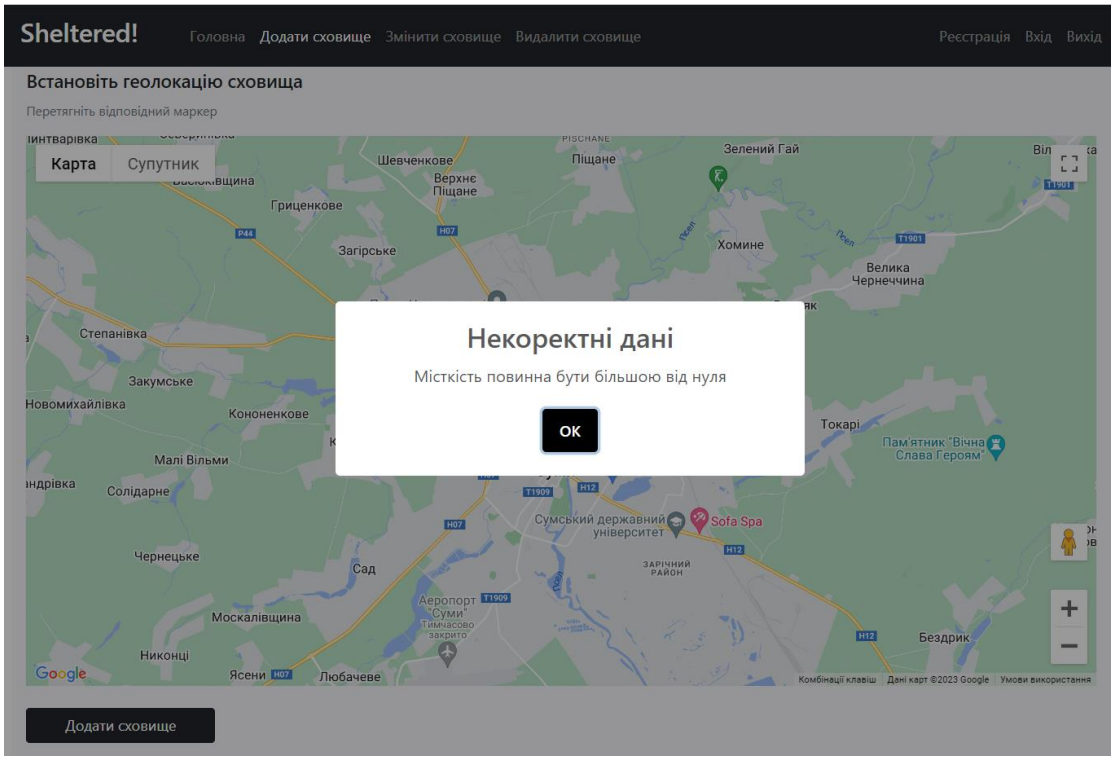

Рисунок 3.14 – Помилка даних під час додавання сховища

На рисунку 3.15 можна побачити сторінку редагування сховища. Вона містить клікабельні маркери, які змінюють свій символ при виборі певного сховища. Після вибору сховища також відкривається нижнє меню із усіма поточними параметрами сховища, доступними для редагування. Якщо обрати інше сховище – дані зміняться на відповідні. Таким чином залишиться тільки ввести нове значення у необхідне поле, а потім натиснути кнопку, щоб зберегти нові параметри. Валідація тут працює так само, як і при додаванні. Перевірку на те, чи було обрано сховище, не було додано, адже поки не буде натиснуто хоча б на один маркер, нижнє меню із параметрами для редагування не відкриється. До того ж, при перемиканні обраного сховища лунає звук клацання.

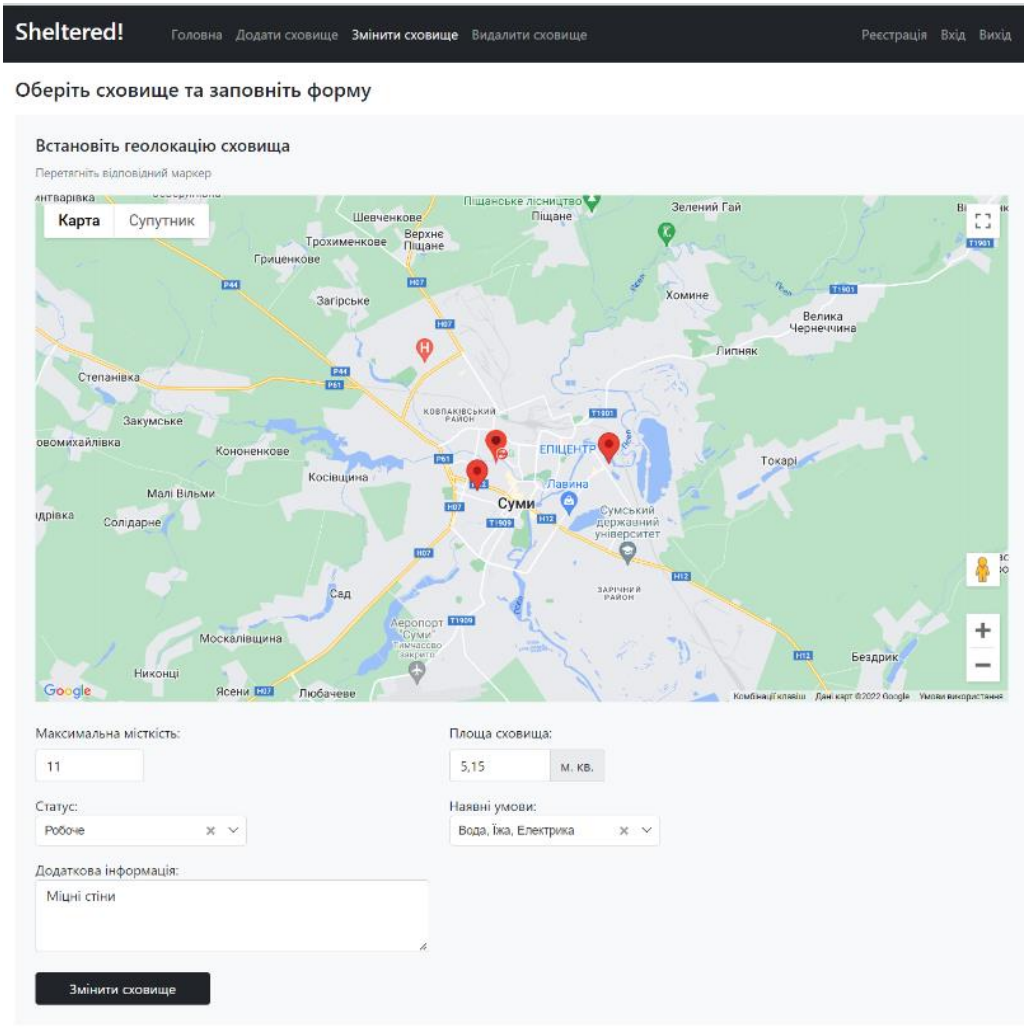

Рисунок 3.15 – Сторінка зміни сховища Sheltered

Залишилася тільки сторінка видалення (рис. 3.16). Вона також має клікабельні маркери із своєю піктограмою, і єдину кнопку – видалити сховище. Маркери при перемиканні також озвучують клацання. Валідацій при видаленні для даних звісно ж немає. Але є окрема перевірка на те, що сховище було обрано. Якщо перейти на сторінку і одразу спробувати активувати видалення сховища, то на сторінці з’явиться червоний банер під навігаційним меню про неможливість видалення сховища.

Усі можливості інтерактивної мапи працюють на всіх сторінках, маркери клікабельні під час будь-яких режимів перегляду.

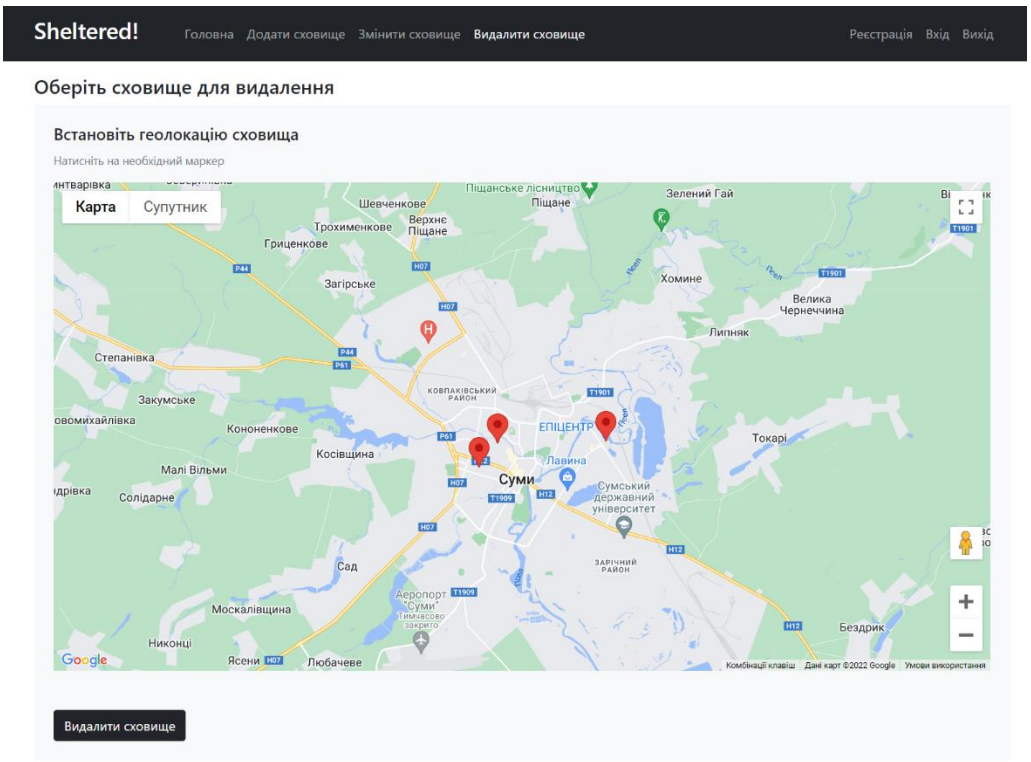

Рисунок 3.16 – Сторінка видалення сховища Sheltered

Дизайн сторінок зроблено таким чином, щоб було якомога менше можливостей ввести некоректні дані. Але навіть при введені неправильних координат, задовгих значень, від’ємних чисел, тощо, відбудеться валідація на клієнтській стороні – виведеться повідомлення із інформацією про неправильне поле. Тим не менш, якщо комусь вдасться обійти client-side validation, присутня перевірка даних на стороні сервера, яка вже ніяким чином не дасть завершити операцію із додавання некоректного сховища. Адже якщо доступні ресурси для валідації і на стороні серверу, і на стороні серверу, не буде зайвим перевірити дані на обох сторонах[15].

ВИСНОВКИ

У ході виконання переддипломної практики було виконано:

1) Літературний огляд сучасних джерел для веб-додатків та вбудовування інтерактивних інформаційних систем-мап у додаток. Виявлено можливості використання автоматизованих конструкторів, а також спеціалізованих інструментів для розробки

2) Визначено основні цілі веб-додатку, потенційні можливості застосування. Розглянуто різницю застосування веб-додатку у різних сферах життя із різним степенем важливості та терміновості інформації, яка надається інтерактивною інформаційною системою.

3) Сформовано вимоги для веб-додатку. Обрано методи розв’язку поставленої задачі. Розглянуто необхідні інструменти, необхідні для реалізації інформаційної системи.

4) Розглянуто та обрано інструменти для розробки веб-додатку із інтерактивною системою-мапою. У якості бази даних було обрано Oracle. Як мову програмування та основний framework було обрано Java із використанням Spring Framework.

5) Спроектовано архітектуру додатку, архітектуру бази даних, розроблено дизайн. Загальна архітектура побудована на основі моделі Spring MVC. Архітектура бізнес-логіки найбільш сумісна із Spring Framework – багатошарова із розділенням на сервіси та репозиторії.

6) Реалізовано поставлену задачу із створення інформаційної системи на базі спроектованої архітектури. Створена інформаційна система надає можливості перегляду бомбосховищ на інтерактивній мапі усім користувачам та операції зміни, додавання та видалення бомбосховищ авторизованим користувачам веб-додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лінченко А. Д. Візуалізація даних за допомогою мап: базові аспекти. *Поліграфічні, мультимедійні та Web-технології*: тези доп. Міжнародної науково-технічної конф., м. Львів, 17-18 жовтня 2018 р. / Українська академія друкарства. Львів, 2018. С. 291—296.

2. Інтерактивні мапи в журналістиці: для новачків та просунутих користувачів [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://ms.detector.media/how-to/post/16287/2016-03-21-interaktyvni-mapy-v-zhurnalistytsi-dlya-novachkiv-ta-prosunutykh-korystuvachiv/.

3. Лейберюк О. М. Інтерактивні веб-карти: сутність і основні етапи створення (на прикладі веб-ресурсу Carto). *Український географічний журнал*. 2016. № 4. С. 54–58. DOI: 10.15407/ugz2016.04.054 (дата звернення 04.03.2023)

4. Bauer C., King G., Gregory G. Java persistence with Hibernate. New York: Manning Publications Co., 2016. 580 p.

5. Lockhart J. Modern PHP. 1st ed. Sebastopol: O'reilly Media, 2015. 270 p.

6. Walls C. Spring in Action. 5th ed. New York: Manning Publications, 2019. 498 p.

7. Duckett J. JavaScript and JQuery – Interactive Front-End Web Development. 1st ed. Indianapolis: John Wiley & Sons, 2014. 640 p.

8. Spring MVC Tutorial [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.javatpoint.com/spring-mvc-tutorial.

9. Best Practices in Spring Boot Project Structure [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://medium.com/learnwithnk/best-practices-in-spring-boot-project-structure-layers-of-microservice-versioning-in-api-cadf62bd3459.

10. Oracle FLOAT [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.oracletutorial.com/oracle-basics/oracle-float/.

11. Діаграма ER: Модель діаграми взаємовідносин суб'єктів [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://uk.csstricks.net/8224622-er-diagram-entity-relationship-diagram-model-dbms-example.

12. Spring Security: Authentication and Authorization In-Depth [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.marcobehler.com/guides/spring-security.

13. Introduction to JavaScript and the Google Maps API [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://michaelminn.net/tutorials/google-maps-api/index.html.

14. Що таке AJAX? Синхронізація асинхронних запитів [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://itchief.ru/javascript/ajax-introduction.

15. Input Validation on Client-Side or Server-Side? – 2022. – Режим доступу до ресурсу: https://www.packetlabs.net/posts/input-validation/.

16. Top 10 Mapping APIs: Google Maps, Microsoft Bing Maps and MapQuest [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://www.programmableweb.com/news/top-10-mapping-apis-google-maps-microsoft-bing-maps-and-mapquest/analysis/2015/02/23.

ДОДАТОК А

Лістинг програмного коду

```
        @Component
public class ContextConfig {

   ShelterRepository shelterRepository;

   @Autowired
   public ContextConfig(ShelterRepository shelterRepository) {
      this.shelterRepository = shelterRepository;
   }
}

        @Configuration
public class MvcConfig implements WebMvcConfigurer {

}

        @Configuration
public class PasswordEncoder {

   @Bean
   public BCryptPasswordEncoder bCryptPasswordEncoder() {
      return new BCryptPasswordEncoder();
   }
}

        @Configuration
@EnableWebSecurity
```

```
public class SecurityConfig {

    private UserDetailsService userService;
    private final BCryptPasswordEncoder passwordEncoder;

    private final Logger LOG = LogManager.getLogger(SecurityConfig.class);

    @Autowired
    public SecurityConfig(UserDetailsService userService,
BCryptPasswordEncoder passwordEncoder) {
        this.userService = userService;
        this.passwordEncoder = passwordEncoder;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .mvcMatchers("/shelter/home").permitAll()
                .mvcMatchers("/shelter").permitAll()
                .mvcMatchers("/shelter/**").authenticated()
                .mvcMatchers("/user/**").permitAll()
                .mvcMatchers("/api/**").permitAll();

        http.authenticationProvider(authProvider());
        http.formLogin()
                .defaultSuccessUrl("/shelter/home", true)
                .loginPage("/user/login")
                .loginProcessingUrl("/user/perform-login")
                .usernameParameter("username")
                .passwordParameter("password")
```

```
                .permitAll();
        http.logout().logoutUrl("/user/logout").permitAll();
        http.httpBasic();
        return http.csrf().disable().build();
    }

    @Bean
    public DaoAuthenticationProvider authProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setPasswordEncoder(passwordEncoder);
        provider.setUserDetailsService(userService);
        return provider;
    }

}


        @ControllerAdvice
public class ControllerAdviser {

    private final Logger LOG = LogManager.getLogger(ControllerAdviser.class);

    @ExceptionHandler(BindException.class)
    public ModelAndView handleBindException(BindException ex) {
        LOG.warn("BindException caught with message: " + ex.getMessage());
        ModelAndView mav = new ModelAndView();
        List<String> errorMsgs = new ArrayList<>();
        ex.getBindingResult().getAllErrors().forEach((errorMsg) ->
errorMsgs.add(errorMsg.getDefaultMessage()));
        mav.setViewName("error");
```

```
        mav.addObject("errors", errorMsgs);
        return mav;
    }

    @ExceptionHandler(IllegalStateException.class)
    public ModelAndView handleIllegalStateException(IllegalStateException ex) {
        LOG.warn("IllegalStateException caught with message: " +
ex.getMessage());
        ModelAndView mav = new ModelAndView();
        List<String> errorMsg = Collections.singletonList(ex.getMessage());
        mav.setViewName("error");
        mav.addObject("errors", errorMsg);
        return mav;
    }

    @ExceptionHandler(Exception.class)
    public ModelAndView handleException(Exception ex) {
        LOG.warn("Exception caught with message: " + ex.getMessage());
        LOG.warn("Exception class: " + ex.getClass().getCanonicalName());
        ModelAndView mav = new ModelAndView();
        mav.setViewName("error");
        return mav;
    }
}

        @Controller
@RequestMapping("/shelter")
public class ShelteredController {

    private final ShelterService shelterService;
```

```
    @Autowired
    public ShelteredController(ShelterService shelterService) {
        this.shelterService = shelterService;
    }

    @GetMapping("/home")
    public String index(Model model) {
        Double lat = null, lng = null;
        model.addAttribute("lat", lat);
        model.addAttribute("lng", lng);

        model.addAttribute("shelters", shelterService.findAll());
        return "index";
    }

    @GetMapping("")
    public String showShelterSubmit(Model model,
                                    @RequestParam(value = "lat", required = false) Double lat,
                                    @RequestParam(value = "lng", required = false) Double
lng) {
        Shelter shelter = shelterService.findByCoords(lat, lng);

        if (shelter == null) {
            model.addAttribute("error", "Неможливо відобразити сховище");
            return "/home";
        } else {
            model.addAttribute("shelter", shelter);
            model.addAttribute("statusLabel", shelter.getStatus().label);
            return "shelter";
```

```
    }
}

@GetMapping("/add-form")
public String addShelter(Model model) {
    model.addAttribute("shelter", new Shelter());
    model.addAttribute("shelters", shelterService.findAll());
    model.addAttribute("statuses", ShelterStatus.values());
    model.addAttribute("conditions", ShelterConditions.values());
    return "add";
}

@PostMapping("/add")
public RedirectView addShelterSubmit(@Valid @ModelAttribute Shelter
shelter, RedirectAttributes redirectAttributes) {
    shelterService.addShelter(shelter);
    redirectAttributes.addFlashAttribute("success", "Сховище успішно
додано");
    return new RedirectView("/shelter/add-form", true);
}

@GetMapping("/edit-form")
public String editShelter(Model model) {
    model.addAttribute("shelter", new Shelter());
    model.addAttribute("shelters", shelterService.findAll());
    model.addAttribute("statuses", ShelterStatus.values());
    model.addAttribute("conditions", ShelterConditions.values());
    return "edit";
}
```

```
    @PostMapping("/edit")
    public RedirectView editShelterSubmit(@Valid @ModelAttribute Shelter
shelter, RedirectAttributes redirectAttributes) {
        shelterService.updateShelter(shelter);
        redirectAttributes.addFlashAttribute("success", "Сховище успішно
змінено");
        //redirectAttributes.addFlashAttribute("error", "Неможливо змінити
сховище");

        return new RedirectView("/shelter/edit-form", true);
    }

    @GetMapping("/delete-form")
    public String deleteShelter(Model model) {
        Double lat = null, lng = null;
        model.addAttribute("lat", lat);
        model.addAttribute("lng", lng);

        model.addAttribute("shelters", shelterService.findAll());
        return "delete";
    }

    @PostMapping("/delete")
    public RedirectView deleteShelterSubmit(@RequestParam(value = "lat",
required = false) Double lat,
                                            @RequestParam(value = "lng", required = false)
Double lng,
                                            RedirectAttributes redirectAttributes) {
        if (lat == null || lng == null) {
            redirectAttributes.addFlashAttribute("error", "Неможливо видалити
```

```
сховище");

        } else {
            shelterService.deleteShelter(lat, lng);
            redirectAttributes.addFlashAttribute("success", "Сховище успішно
видалено");
        }
        return new RedirectView("/shelter/delete-form", true);
    }
}

        @RestController
@RequestMapping("/api")
public class ShelteredRestController {

    private final ShelterService shelterService;

    @Autowired
    public ShelteredRestController(ShelterService shelterService) {
        this.shelterService = shelterService;
    }

    @GetMapping(value = "/shelter", produces = "application/json")
    public Shelter getShelter(@RequestParam(value = "lat") double lat,
@RequestParam(value = "lng") double lng) {
        return shelterService.findByCoords(lat, lng);
    }

    @GetMapping(value = "/shelters", produces = "application/json")
    public List<Shelter> getShelters() {
```

```
        return shelterService.findAll();
    }

    @GetMapping(value = "/coords", produces = "application/json")
    public List<Coordinates> getSheltersCoordinates() {
        return shelterService.getAllCoords();
    }

}

        @Controller
@RequestMapping("/user")
public class UserController {

    private final UserService userService;
    private final RegistrationService registrationService;

    @Autowired
    public UserController (UserService userService, RegistrationService
registrationService) {
        this.userService = userService;
        this.registrationService = registrationService;
    }

    @GetMapping("/receive-key")
    public String receiveKey(Model model) {
        model.addAttribute("request", new AccessKeyRequest());
        return "access-key";
    }
```

```
    @PostMapping("/receive-key")
    public RedirectView receiveKeySubmit(@Valid @ModelAttribute
AccessKeyRequest request, RedirectAttributes redirectAttributes) {
        registrationService.sendAccessKeyRequest(request);
        redirectAttributes.addFlashAttribute("success", "Заявку успішно надіслано.
Очікуйте відповіді");
        return new RedirectView("/user/receive-key", true);
    }

    @GetMapping("/registration")
    public String registration(Model model) {
        model.addAttribute("request", new RegistrationRequest());
        return "registration";
    }

    @PostMapping("/registration")
    public RedirectView registrationSubmit(@Valid @ModelAttribute
RegistrationRequest request, RedirectAttributes redirectAttributes) {
        String redirectUrl;
        if (registrationService.registerNewUser(request)) {
            redirectUrl = "/shelter/home";
            //автоматичний логін?
        } else {
            redirectAttributes.addFlashAttribute("error", "Неможливо зареєструвати
користувача");
            redirectUrl = "/user/registration";
        }
        return new RedirectView(redirectUrl, true);
    }
```

```
    @RequestMapping(value="/logout", method = RequestMethod.GET)
    public String customLogout(HttpServletRequest request, HttpServletResponse
response) {
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();

        if (authentication != null){
            new SecurityContextLogoutHandler().logout(request, response,
authentication);
        }
        return "redirect:/user/login";
    }

    @GetMapping("/login")
    public String login() {
        return "login";
    }

    /*
    @PostMapping("/login")
    public RedirectView submitLogin() {
        return new RedirectView();
    }

     */
}

        @Entity
public class AccessKey {
```

```
@Id
private String email;
private String key;

public AccessKey() {

}

public AccessKey(String email, String key) {
    this.email = email;
    this.key = key;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getKey() {
    return key;
}

public void setKey(String key) {
    this.key = key;
}

@Override
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        AccessKey accessKey = (AccessKey) o;
        return Objects.equals(email, accessKey.email) && Objects.equals(key,
accessKey.key);
    }

    @Override
    public int hashCode() {
        return Objects.hash(email, key);
    }
}

          public class AccessKeyRequest {

    @Size(min = 1, message = "Ім'я не може бути менше 1 символу")
    @Size(max = 30, message = "Ім'я не може бути більше 30 символів")
    private String firstName;

    @Size(min = 1, message = "Прізвище не може бути менше 1 символу")
    @Size(max = 30, message = "Прізвище не може бути більше 30 символів")
    private String lastName;

    @Size(min = 1, message = "Назва організація не може бути менше 1
символу")
    @Size(max = 30, message = "Назва організація не може бути більше 40
символів")
    private String organisation;
```

```
@Email(message = "Некоректна пошта")
private String email;

@Size(min = 1, message = "Номер телефону не може бути менше за 1")
@Size(max = 14, message = "Номер телефону не може бути менше за 14")
private String phone;

public AccessKeyRequest() {

}

public String getFirstName() {
   return firstName;
}

public void setFirstName(String firstName) {
   this.firstName = firstName;
}

public String getLastName() {
   return lastName;
}

public void setLastName(String lastName) {
   this.lastName = lastName;
}

public String getOrganisation() {
   return organisation;
}
```

```
    public void setOrganisation(String organisation) {
        this.organisation = organisation;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

}

        @Embeddable
public class Coordinates implements Serializable {

    @Max(value = 90, message = "Широта не може бути більшою за 90")
    @Min(value = -90, message = "Широта не може бути меншою за -90")
    private double latitude;
```

```
@Max(value = 180, message = "Довгота не може бути більшою за 180")
@Min(value = -180, message = "Довгота не може бути меншою за -180")
private double longitude;

public Coordinates() {

}

public Coordinates(double latitude, double longitude) {
    this.latitude = latitude;
    this.longitude = longitude;
}

public double getLatitude() {
    return latitude;
}

public void setLatitude(double latitude) {
    this.latitude = latitude;
}

public double getLongitude() {
    return longitude;
}

public void setLongitude(double longitude) {
    this.longitude = longitude;
}

@Override
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Coordinates that = (Coordinates) o;
        return Double.compare(that.latitude, latitude) == 0 &&
Double.compare(that.longitude, longitude) == 0;
    }

    @Override
    public int hashCode() {
        return Objects.hash(latitude, longitude);
    }
}
```

```
        public class RegistrationRequest {

    @Email(message = "Некоректна пошта")
    private String email;

    @Size(min = 8, message = "Мінімальна довжина паролю 8 символів")
    @Size(max = 16, message = "Максимальна довжина паролю 16 символів")
    private String password;

    @Size(min = 36, max = 36, message = "Некоректний ключ")
    private String accessKey;

    public RegistrationRequest() {
    }

    public String getEmail() {
```

```
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getAccessKey() {
        return accessKey;
    }

    public void setAccessKey(String accessKey) {
        this.accessKey = accessKey;
    }
}
```

```
        @Entity
@Table(name = "shelters")
public class Shelter {

    @EmbeddedId
    private Coordinates coordinates;
```

```
    @Enumerated(EnumType.STRING)
    @NotNull(message = "Статус не може бути пустим")
    private ShelterStatus status;

    @ElementCollection(targetClass = ShelterConditions.class)
    @JoinTable(name = "conditions")
    @Enumerated(EnumType.STRING)
    @NotNull(message = "Умови не можуть бути пустими")
    private List<ShelterConditions> conditions;

    @Min(value = 1, message = "Місткість не може бути меншою за 1")
    private int capacity;

    @DecimalMin(value = "0.01", message = "Площа не може бути меншою за
1")
    private double area;

    @Size(max = 250, message = "Додаткова інформація має бути не більше 250
символів")
    private String additional;

    public Shelter() {
    }

    public Coordinates getCoordinates() {
        return coordinates;
    }

    public void setCoordinates(Coordinates coordinates) {
```

```
    this.coordinates = coordinates;
}

public int getCapacity() {
    return capacity;
}

public void setCapacity(int capacity) {
    this.capacity = capacity;
}

public double getArea() {
    return area;
}

public void setArea(double area) {
    this.area = area;
}

public ShelterStatus getStatus() {
    return status;
}

public void setStatus(ShelterStatus status) {
    this.status = status;
}

public List<ShelterConditions> getConditions() {
    return conditions;
}
```

```
    public void setConditions(List<ShelterConditions> conditions) {
        this.conditions = conditions;
    }

    public String getAdditional() {
        return additional;
    }

    public void setAdditional(String additional) {
        this.additional = additional;
    }

    @Override
    public String toString() {
        return "Shelter{" +
                "latitude=" + coordinates.getLatitude() +
                ", longitude=" + coordinates.getLongitude() +
                ", status=" + status +
                ", conditions=" + conditions +
                ", capacity=" + capacity +
                ", area=" + area +
                ", additional='" + additional + '\'' +
                '}';
    }
}

        public enum ShelterConditions {
    WATER("Вода"),
    FOOD("Їжа"),
```

```
    ELECTRICITY("Електрика"),
    SEATS("Місця для сидіння"),
    WIFI("Вай-фай"),
    SOCKETS("Розетки"),
    RADIATION_PROTECTED("Протирадіаційне"),
    LIGHTING("Освітлення"),
    MEDICINES("Медикаменти");

    public final String label;

    ShelterConditions(String label) {
        this.label = label;
    }
}

        @Entity
public class ShelteredUser implements UserDetails {

    @Id
    @Column(name = "email")
    private String username;

    private String password;

    @Enumerated(EnumType.STRING)
    private UserRole role;
    private boolean enabled;
    private boolean locked;

    public ShelteredUser() {
```

```
    }

    public ShelteredUser(String username, String password, UserRole role, boolean
enabled, boolean locked) {
        this.username = username;
        this.password = password;
        this.role = role;
        this.enabled = enabled;
        this.locked = locked;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return Collections.singletonList(new SimpleGrantedAuthority(role.name()));
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
```

```
}

@Override
public boolean isAccountNonLocked() {
    return !locked;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return enabled;
}

public void setUsername(String username) {
    this.username = username;
}

public void setPassword(String password) {
    this.password = password;
}

public UserRole getRole() {
    return role;
}

public void setRole(UserRole role) {
```

```
        this.role = role;
    }

    public void setEnabled(boolean enabled) {
        this.enabled = enabled;
    }

    public boolean isLocked() {
        return locked;
    }

    public void setLocked(boolean locked) {
        this.locked = locked;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        ShelteredUser that = (ShelteredUser) o;
        return enabled == that.enabled && locked == that.locked &&
Objects.equals(username, that.username) && Objects.equals(password,
that.password) && role == that.role;
    }

    @Override
    public int hashCode() {
        return Objects.hash(username, password, role, enabled, locked);
    }
}
```

```
        public enum ShelterStatus {
    IN_USE("Робоче"),
    UNDER_CONSTRUCTION("Будується"),
    BLOCKED("Заблоковане"),
    DESTROYED("Знищене"),
    ABANDONED("Покинуте"),
    UNKNOWN("Невідомо");

    public final String label;

    ShelterStatus(String label) {
        this.label = label;
    }
}

        public enum UserRole {
    ROLE_USER,
    ROLE_ADMIN
}

        @Repository
public interface AccessKeyRepository extends JpaRepository<AccessKey, String>
{

    List<AccessKey> findAllByKey(String key);
    Optional<AccessKey> findByEmail(String email);

}
```

```
        @Repository
public interface ShelterRepository extends JpaRepository<Shelter, Coordinates> {

   @Query("SELECT new
ua.edu.sumdu.nefodov.sheltered.model.Coordinates(s.coordinates.latitude,
s.coordinates.longitude) FROM Shelter s")
   List<Coordinates> findAllCoordinates();
}

        @Repository
@Transactional
public interface UserRepository extends JpaRepository<ShelteredUser, String> {

   Optional<ShelteredUser> findByUsername(String email);
}

        @Service
public class EmailSenderService {

   private JavaMailSender mailSender;

   @Autowired
   public EmailSenderService (JavaMailSender mailSender) {
      this.mailSender = mailSender;
   }

   @Async
   public void send(String to, String subject, String text) {
      try {
         MimeMessage msg = mailSender.createMimeMessage();
```

```
            MimeMessageHelper mmh = new MimeMessageHelper(msg, "utf-8");
            mmh.setTo(to);
            mmh.setSubject(subject);
            mmh.setText(text);
            mailSender.send(msg);
        } catch (MessagingException e) {
            throw new IllegalStateException("Неможливо надіслати листа");
        }
    }
}

        @Service
@Transactional
public class RegistrationService {

    private final UserRepository userRepository;
    private final AccessKeyRepository accessKeyRepository;
    private final EmailSenderService emailSender;

    private  final BCryptPasswordEncoder bCryptPasswordEncoder;

    private final String KEY_REQUEST_MAIL_TO = "nefyodov.nazar@ukr.net";
    private final String KEY_REQUEST_MAIL_SUBJECT = "[Sheltered] Нова
заявка на ключ від ";
    private final UserRole DEFAULT_ROLE = UserRole.ROLE_USER;

    @Autowired
    public RegistrationService(UserRepository userRepository,
                               AccessKeyRepository accessKeyRepository,
                               EmailSenderService emailSender,
```

```
                        BCryptPasswordEncoder bCryptPasswordEncoder) {
        this.userRepository = userRepository;
        this.accessKeyRepository = accessKeyRepository;
        this.emailSender = emailSender;
        this.bCryptPasswordEncoder = bCryptPasswordEncoder;
    }

    public void sendAccessKeyRequest(AccessKeyRequest request) {
        String key;
        do {
            key = UUID.randomUUID().toString();
        } while (!accessKeyRepository.findAllByKey(key).isEmpty());

        if (accessKeyRepository.findByEmail(request.getEmail()).isPresent()) {
            throw new IllegalStateException("Заявку із такою поштою вже
надіслано");
        }
        accessKeyRepository.save(new AccessKey(request.getEmail(), key));

        emailSender.send(KEY_REQUEST_MAIL_TO,
                KEY_REQUEST_MAIL_SUBJECT + request.getOrganisation(),
                prepareKeyRequestText(request, key));
    }

    public boolean registerNewUser(RegistrationRequest request) {

        List<AccessKey> keys =
accessKeyRepository.findAllByKey(request.getAccessKey());
        String email = request.getEmail();
```

```
        if (userRepository.findByUsername(email).isEmpty() &&
                keys.contains(new AccessKey(email, request.getAccessKey()))) {
            String encodedPassword =
bCryptPasswordEncoder.encode(request.getPassword());
            ShelteredUser user = new ShelteredUser(email, encodedPassword,
DEFAULT_ROLE, true, false);
            userRepository.save(user);
            return true;
        } else {
            return false;
        }


    }

    private String prepareKeyRequestText(AccessKeyRequest request, String key) {
        return "Новий запит на отримання ключа доступу до Sheltered!\n\n" +
                "Прізвище та ім'я: " +
                request.getLastName() + " " +
                request.getFirstName() + "\n" +
                "Організація: " + request.getOrganisation() + "\n" +
                "Пошта: " + request.getEmail() + "\n" +
                "Телефон: " + request.getPhone() + "\n" +
                "Згенерований ключ: " + key;
    }
}

        @Service
public class ShelterService {
```

```
    private final ShelterRepository shelterRepo;

    @Autowired
    public ShelterService(ShelterRepository shelterRepo) {
        this.shelterRepo = shelterRepo;
    }

    public List<Shelter> findAll() {
        return shelterRepo.findAll();
    }

    public Shelter findByCoords(double lat, double lng) {
        return shelterRepo.findById(new Coordinates(lat, lng)).orElse(null);
    }

    public void addShelter(Shelter newShelter) {
        shelterRepo.save(newShelter);
    }

    public void updateShelter(Shelter updatedShelter) {
        Coordinates coords = new
Coordinates(updatedShelter.getCoordinates().getLatitude(),
updatedShelter.getCoordinates().getLongitude());
        Shelter shelterToUpdate = shelterRepo.findById(coords).orElse(null);

        if (shelterToUpdate != null) {
            shelterToUpdate.setStatus(updatedShelter.getStatus());
            shelterToUpdate.setConditions(updatedShelter.getConditions());
            shelterToUpdate.setCapacity(updatedShelter.getCapacity());
            shelterToUpdate.setArea(updatedShelter.getArea());
```

```
            shelterToUpdate.setAdditional(updatedShelter.getAdditional());

            shelterRepo.save(shelterToUpdate);
        }
    }

    public void deleteShelter(double lat, double lng) {
        Coordinates coords = new Coordinates(lat, lng);
        shelterRepo.deleteById(coords);
    }

    public List<Coordinates> getAllCoords() {
        return shelterRepo.findAllCoordinates();
    }
}
```

```
        @Service
@Transactional
public class UserService implements UserDetailsService {

    private final UserRepository userRepository;
    private final String USER_NOT_FOUND_MSG = "Користувача не
знайдено";

    private final Logger LOG = LogManager.getLogger(UserService.class);

    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
```

```
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        LOG.info("LOAD USER BY USERNAME");
        LOG.info(username);
        return userRepository.findByUsername(username).orElseThrow(() -> new
UsernameNotFoundException(USER_NOT_FOUND_MSG));
    }
}

        @SpringBootApplication
public class ShelteredApplication {

    public static void main(String[] args) {
        SpringApplication.run(ShelteredApplication.class, args);
    }

}
```