ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

на тему Проектування і розробка системи з мікросервісною архітектурою
Design and development of a system with microservice architecture

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія .
(шифр і назва напряму підготовки, спеціальності)
Освітня програма «Комп'ютерна інженерія»
(назва освітньої програми)
Долгополов Владислав Васильович
(прізвище, ім'я, по-батькові)

Керівник к.ф.-м.н. Антоненко О. С. _____
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент к.ф.-м.н., доц. Шпинарева І.М. _____
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№ 12 від « 09 » 06 2023 р.

Завідувач кафедри
_____ _Євгеній МАЛАХОВ_
(підпис) (ім'я, прізвище)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2023 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК
_____ _Алла КОБОЗЄВА_
(підпис) (ім'я, прізвище)

Одеса - 2023

АНОТАЦІЯ

Мета дипломної роботи – проектування та розробка сервісу "Бюро знахідок" за допомогою мікросервісної архітектури. Реалізація виконана з використанням мови програмування Python, фреймворку FastAPI для бекенду, Vue.js для фронтенду.

Інформація про знахідки та згублені предмети отримується через веб-інтерфейс, де користувачі можуть розміщувати дані про знайдені або втрачені речі.

Для реалізації поставленої мети були спроектовані наступні мікросервіси: сервіс для обробки взаємодії з оголошеннями, користувачами та зустрічами, сервіс для обробки даних пов’язаних з геолокаціями, сервіс для створення рекомендацій та сервіс для обробки скарг та надсилання розсилки поштою.

Результатом роботи є функціонуючий веб-додаток, який включає в себе систему управління базою даних, мікросервіси для обробки запитів від користувачів.

Система має гнучку архітектуру, що дозволяє легко масштабувати її за потреби та розширювати функціонал.

ABSTRACT

The aim of the graduate project is to design and develop the "Lost and Found Bureau" service using a microservices architecture. The implementation is done using the Python programming language, FastAPI framework for the backend, and Vue.js for the frontend.

Information about found and lost items is obtained through a web interface, where users can submit data about the items they found or lost.

To achieve the set goal, the following microservices were designed: a service for processing interactions with ads, users, and meetings; a service for processing geolocation-related data; a recommendation service, and a service for handling complaints and sending mailings.

The result of the project is a functional web application that includes a database management system and microservices for handling user requests.

The system has a flexible architecture that allows for easy scalability and functionality expansion as needed.

ЗМІСТ

ВСТУП 5
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Мікросервісна архітектура 7
1.2 Огляд інфраструктурних рішень мікросервісної архітектури 8
1.3 Переваги мікросервісного підходу 9
1.4 Вимоги до системи, що розробляється 10
2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ 12
2.1 Проектування реляційної бази даних 16
2.2 Проектування бекенд архітектури 19
2.3 Проектування клієнтської підсистеми 22
3 ПРОГРАМНА РЕАЛІЗАЦІЯ БЕКЕНД ДОДАТКУ 24
4 ПРОГРАМНА РЕАЛІЗАЦІЯ ФРОНТЕНД ДОДАТКУ 34
ВИСНОВКИ 42
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 43
ДОДАТОК А 44
ДОДАТОК Б 46

ВСТУП

Сервіси "Бюро знахідок", що допомагають людям відновлювати зв'язок з втраченими речами, виступають важливими елементами сучасного суспільства. Оптимальна ефективність, зручність та доступність цих сервісів в значній мірі залежать від застосовуваних технологій. Використання мікросервісної архітектури у рамках даної дипломної роботи дозволяє створити гнучку, масштабовану та надійну систему, спроможну легко адаптуватися до змінюваних вимог та умов.

У центрі дослідження стоять згублені та знайдені предмети, інформація про які зберігається і обробляється в сервісі. Використання мікросервісної архітектури забезпечує можливість декомпозування системи на окремі служби, що значно спрощує управління, розробку та масштабування сервісу.

Мета даного роботи полягає у проектуванні та розробці високопродуктивного та масштабованого сервісу "Бюро знахідок" на основі мікросервісної архітектури. Планується створити сервіс, що надасть користувачам можливість публікувати інформацію про згублені та знайдені предмети, а також переглядати цю інформацію в зручному форматі.

У рамках даного дослідження необхідно розв'язати наступні завдання:

1) Провести глибокий аналіз предметної області створення сервісів "Бюро знахідок" на основі мікросервісної архітектури, включаючи дослідження найкращих практик, технологій та інструментів;
2) Спроектувати мікросервісну архітектуру для сервісу "Бюро знахідок", яка забезпечить високу продуктивність, масштабованість та надійність системи;
3) Розробити бекенд сервісу за допомогою сучасного та ефективного фреймворку який надає змогу використовувати мікросервіси, а фронтенд — за допомогою потужного і гнучкого фреймворку який надасть можливість користувачу найбільшу зручність;

4) Розгорнути сервіс в хмарному середовищі для забезпечення його стабільної роботи та легкого масштабування;
5) Протестувати функціональність інфраструктури сервісу та його коректну роботу, включаючи надійність збереження даних та здатність системи коректно обробляти запити користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мікросервісна архітектура

Мікросервісна архітектура представляє собою підхід до розробки програмного забезпечення, який став особливо популярним в останні роки. Він дозволяє розбивати більш великі та складні програми на більш малі, незалежні компоненти, що називаються мікросервісами. Ці мікросервіси потім спілкуються один з одним за допомогою API (Application Programming Interface) і виконують окремі бізнес-функції. Це створює ряд переваг, таких як:

1) Масштабованість: Мікросервіси можна масштабувати окремо від інших частин системи. Це означає, що якщо певний компонент системи потребує більше ресурсів, ви можете просто додати ще ресурсів до цього конкретного мікросервісу, замість того щоб масштабувати всю систему;
2) Гнучкість: Кожен мікросервіс може бути написаний на різних мовах програмування і використовувати різні технології зберігання даних. Це означає, що кожен мікросервіс може бути оптимізований для конкретної задачі;
3) Розподілена розробка: Оскільки мікросервіси незалежні один від одного, різні команди можуть працювати над різними мікросервісами одночасно. Це сприяє більш ефективному процесу розробки;
4) Помилки локалізуються: У випадку виникнення проблеми в одному мікросервісі, ця проблема не вплине на решту системи;
5) Легкість оновлень: Кожен мікросервіс може бути оновлений незалежно, що робить процес оновлень менш ризикованим та менш переривчивим для всієї системи.

Мікросервісна архітектура стала одним із ключових підходів в індустрії програмного забезпечення, особливо для великих компаній, що займаються розробкою складних систем. Наступні глобальні компанії та їх сервіси використовують мікросервісний підхід у своїх рішеннях:

1) Netflix: Netflix був одним з перших, хто усвідомив переваги мікросервісів, і є прикладом компанії, яка з успіхом впровадила цю архітектуру на велику масштаб. Netflix використовує сотні мікросервісів, щоб допомогти в управлінні своєю великою глобальною платформою для потокового відео;
2) Amazon: Amazon перейшов від монолітної до мікросервісної архітектури в 2000-х роках, що допомогло їм масштабувати їхній веб-сайт та служби до поточних розмірів;
3) Uber: Uber використовує мікросервіси для керування своєю великою та географічно розподіленою платформою. Вони використовують окремі мікросервіси для різних аспектів свого бізнесу, таких як обробка платежів, управління поїздками і трекінг машин.

1.2 Огляд інфраструктурних рішень мікросервісної архітектури

Інфраструктурні рішення мікросервісної архітектури включають в себе ряд технологій, які забезпечують розробку, розгортання, моніторинг та управління мікросервісами. Прикладами таких технологій є FastAPI, Vue.js і PostgreSQL.

FastAPI [1] – це сучасний, швидкий (високопродуктивний), що побудований на стандартах, веб-фреймворк для Python. Він базується на стандартах Python 3.6 типів. FastAPI добре підходить для побудови мікросервісів, оскільки він має високу продуктивність, легко розширюється і підтримує асинхронну обробку запитів.

Vue.js – це прогресивний фреймворк JavaScript для побудови інтерфейсів користувача. На відміну від інших монолітних фреймворків, Vue розроблений з думкою про поступове впровадження.

PostgreSQL – це потужна, відкрита об'єктно-реляційна система управління базами даних, яка використовує та розширює мову SQL об'єднану з багатьма можливостями, які зберігають найбільш вимогливі розробники веб-додатків.

Ці технології разом дають можливість створити масштабований, високопродуктивний веб-сервіс з зручним інтерфейсом користувача і надійним зберіганням даних.

1.3 Переваги мікросервісного підходу

Мікросервісна архітектура має ряд переваг над монолітною, хоча вибір між цими двома підходами в значній мірі залежить від специфіки проекту, його обсягу, бізнес-вимог і технічних обмежень. Незважаючи на це, ось деякі причини, чому мікросервісна архітектура може бути кращою за монолітну:

1) Незалежність сервісів: Кожен мікросервіс є незалежною одиницею, яка може бути розроблена, випробувана, розгорнута і масштабована незалежно від інших сервісів. Це означає, що команди можуть працювати паралельно і вносити зміни в систему без впливу на інші частини системи;
2) Розширюваність: Мікросервіси дозволяють розширювати окремі компоненти системи незалежно один від одного, що може бути ефективніше, ніж масштабування всього монолітного додатку;
3) Відмовостійкість: У мікросервісній архітектурі, помилка в одному сервісі зазвичай не впливає на роботу інших сервісів. Це робить мікросервісну архітектуру більш відмовостійкою, ніж монолітну;
4) Технологічна гнучкість: У мікросервісах кожен сервіс може використовувати різні технології, такі як мови програмування, фреймворки, бази даних тощо, що забезпечує більшу технологічну гнучкість;
5) Полегшена підтримка та обслуговування: Оскільки мікросервіси є незалежними, це полегшує управління, оновлення та усунення помилок. Крім того, їх можна легко замінити або оновити без впливу на інші частини системи.

1.4 Вимоги до системи, що розробляється

Загальною функціональністю інфраструктури є:

1) надання можливості користувачам публікувати інформацію про згублені та знайдені предмети;
2) надання можливості користувачам переглядати інформацію про згублені та знайдені предмети;
3) система пошуку, щоб користувачі могли шукати втрачені або знайдені предмети за кількома параметрами;
4) перевірка на чесність користувачів;
5) верифікація користувачів для підтвердження їхньої автентичності;
6) надання можливості контактувати з іншими користувачами.

Не функціональні вимоги до інфраструктури:

1) масштабованість – система повинна бути здатна обробляти збільшення обсягу користувачів та операцій без зниження продуктивності;
2) надійність – система повинна бути стійкою до помилок та збоїв, і забезпечувати неперервну роботу сервісу;
3) безпека – дані користувачів повинні бути безпечно збережені, а доступ до системи повинен контролюватися для запобігання несанкціонованого використання;
4) підтримка – система повинна мати можливість легкого оновлення та модифікації для покращення сервісу та додавання нових функцій.

Інфраструктура підтримує дві категорії користувачів:

1) Зареєстровані користувач – це особи, які створили обліковий запис в системі. Вони можуть публікувати оголошення про знайдені або загублені предмети, переглядати та відповідати на інші оголошення, налаштовувати персональні параметри і отримувати сповіщення про потенційні відповідності;

2) Адміністратор – це користувачі з особливими привілеями, які можуть керувати системою. Вони мають можливість видаляти або редагувати оголошення, управляти користувачами (наприклад, блокувати або видаляти облікові записи), та виконувати інші адміністративні функції.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ

Для проектування архітектури інфраструктури сервісу "Бюро знахідок" необхідно спочатку визначити його ключові компоненти. Основні складові частини інфраструктури сервісу "Бюро знахідок" включають:

1) бекенд-додаток – цей компонент відповідає за обробку логіки та бізнес-процесів, а також за збереження та обробку даних. У випадку сервісу "Бюро знахідок", бекенд відповідає за обробку запитів користувачів, управління знахідками та втратами, створення зустрічей, комунікацію з іншими користувачами, збереження і витягування даних з бази даних, аутентифікацію та авторизацію користувачів, комунікацію з іншими компонентами системи;
2) фронтенд-додаток – цей компонент відповідає за взаємодію з користувачем та представлення інтерфейсу користувача, створення веб-сторінок, на яких користувачі можуть переглядати оголошення, створювати нові записи, здійснювати пошук;
3) реляційна база даних – база даних, яка взаємодіє з бекенд-додатком. Вона відповідає за зберігання структурованих даних в табличному форматі.

Зважаючи на складові части при проектуванні архітектури інформаційної системи буде доцільним обрати триланкову архітектуру клієнт-сервер, так як в ній існує три частини, які дуже вдало підходять під наші вимоги:

1) Презентаційний рівень (або рівень користувацького інтерфейсу) відповідає за взаємодію з користувачем. Цей рівень включає в себе всі аспекти, які стосуються відображення даних користувачу, взаємодії користувача з інтерфейсом та представлення даних в зручному для користувача вигляді;
2) Рівень бізнес-логіки (або рівень обробки) містить всю бізнес-логіку, що керує додатком, опрацьовує дані, обробляє запити та інформацію з презентаційного рівня, і комунікує з рівнем даних. Тут знаходяться всі алгоритми, виконуються всі розрахунки, вирішуються бізнес-задачі;

3) Рівень даних займається зберіганням і управлінням даних. Цей рівень містить бази даних, системи управління базами даних (СУБД) та інші служби для зберігання, видалення та вибірки даних.

Також буде доцільним обрати архітектурний стиль REST (рис.2.1). REST (Representational State Transfer) – це архітектурний стиль, який використовується для розробки веб-сервісів. Він заснований на принципах простоти, масштабованості, незалежності від стану і кешування.

Основні принципи REST:

1) Клієнт-серверна архітектура: Система розділена на клієнтську (користувацьку) та серверну (додаткову) частини. Це дозволяє забезпечити незалежність інтерфейсу користувача від серверної логіки та покращити масштабованість системи;
2) Безстановність (Statelessness): Кожен запит клієнта до сервера повинен містити всю необхідну інформацію для обробки цього запиту. Сервер не зберігає стан клієнта між запитами. Це сприяє простоті і підтримці багатоклієнтської взаємодії та покращує масштабованість сервера;
3) Кешування: Сервер може вказувати клієнту, які відповіді можуть бути кешовані для майбутніх запитів. Це дозволяє зменшити навантаження на сервер і покращити продуктивність системи;
4) Єдина точка входу: Система повинна мати єдину точку входу, через яку клієнти отримують доступ до ресурсів. Це дозволяє забезпечити централізоване управління і зробити систему більш простою та зрозумілою;
5) Операції на ресурсах: Клієнт звертається до сервера для виконання операцій над ресурсами, такими як створення, читання, оновлення або видалення. Це дозволяє забезпечити інтуїтивний та стандартизований спосіб взаємодії з системою.

Порівняння з іншими архітектурними стилями:

- SOAP (Simple Object Access Protocol): SOAP є більш складним та важким для розуміння протоколом, порівняно з REST. Він передає

дані за допомогою XML-формату та використовує WSDL (Web Services Description Language) для опису послуг. У порівнянні з REST, SOAP зазвичай використовується для більш складних та формалізованих систем;

- GraphQL – це запитова мова та середовище виконання запитів, яке дозволяє клієнтам отримувати саме ту інформацію, яка їм потрібна. Використовуючи GraphQL, клієнти можуть визначати структуру та обсяг даних, які вони отримують з сервера. GraphQL дозволяє ефективно передавати дані між клієнтом та сервером, дозволяючи зменшити кількість запитів клієнта до сервера та полегшує роботу зі складними даними.

REST, архітектурний стиль, має переваги порівняно з GraphQL та SOAP, включаючи простоту та легкість використання, підтримку кешування, масштабованість та можливість кешування веб-сторінок. REST також відзначається простотою і швидкістю порівняно з SOAP, а також надає більшу гнучкість та прозорість при роботі з ресурсами.

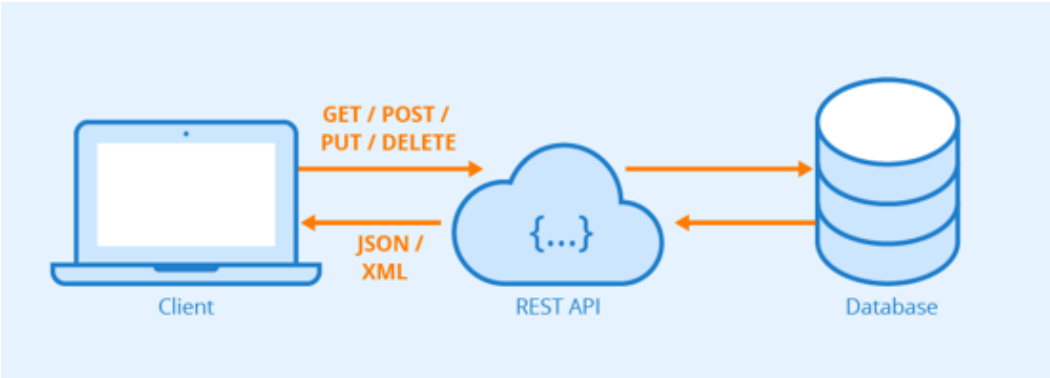

Рисунок 2.1 – Схема архітектури REST

Ґрунтуючись на архітектурі, необхідно також обрати шаблон проектування додатку. Підходящим можна назвати шаблон MVC (рис. 2.2).

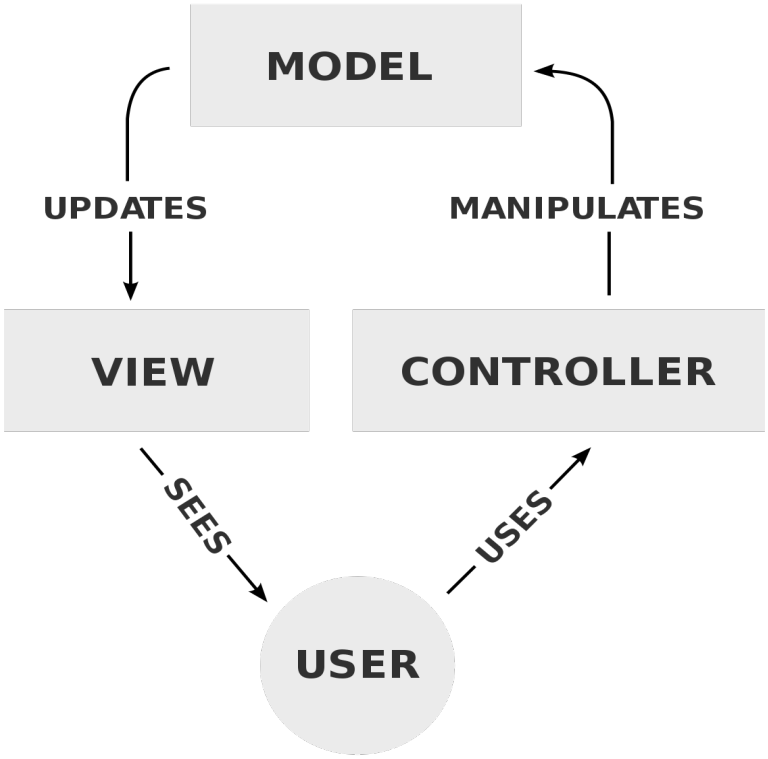

Рисунок 2.2 – Схема шаблону MVC

Патерн MVC (Model-View-Controller) є одним з найпоширеніших архітектурних патернів в розробці програмного забезпечення. Він допомагає розділити логіку додатку на три основні компоненти: Модель (Model), Представлення (View) та Контролер (Controller). Кожен з цих компонентів має свою відповідну відповідальність у системі.

1) Модель (Model): Модель відповідає за управління даними та бізнес-логікою додатку. Вона представляє структуру даних та логіку, яка пов'язана з обробкою та маніпуляцією цих даних. Модель може містити методи для доступу до даних, валідації, обробки подій тощо. Вона не повинна залежати від інших компонентів та забезпечує незалежність даних від їх представлення;
2) Представлення (View): Представлення відповідає за відображення даних користувачу та обробку його взаємодії з додатком. Це можуть бути веб-сторінки, графічні інтерфейси, шаблони тощо. Представлення отримує дані від Моделі та відображає їх у зрозумілому для користувача форматі.

Воно також відповідає за обробку подій та передачу команд до Контролера;

3) Контролер (Controller): Контролер служить посередником між Моделлю та Представленням. Він отримує вхідні події від Представлення, обробляє їх та залучає Модель для оновлення даних. Контролер також відповідає за управління потоком даних та виконанням логіки додатку. Він реагує на дії користувача, виконує необхідні операції з даними та оновлює Представлення.

Основна ідея патерна MVC полягає в розділенні відповідальностей між компонентами, щоб забезпечити модульність, перевикористання коду та зручне управління додатком. Використання патерна MVC допомагає зробити додаток більш структурованим, підтримуваним та розширюваним.

2.1 Проектування реляційної бази даних

Сьогодні існує багато систем керування базами даних (MySQL, Oracle, MariaDB, PostgreSQL тощо). Серед них у проекті обрано PostgreSQL з наступних причин:

- реалізує реляційну модель даних, яка є зрозумілою для кінцевого користувача;
- має гнучкий механізм управління правами користувачів БД (ролі);
- є можливість створювати схеми для відокремлення даних користувачів;
- підтримує мову plpgSQL як розширення стандарту SQL для створення ефективних збережених процедур;

В базі бюро знахідок слід виділити наступні сутності

- користувач – містить інформацію про ім’я у додатку, адресу електронної пошти, рейтинг, кількість знахідок, втрат та скарг, статуси (чи є адміном, заблокованим або активним);

- допис – містить інформацію про заголовок, опис, категорію, дату публікації, дату знахідки, місце знахідки, статус допису(опублікований або збережений, автора поста та чи загублена річ);
- зустріч – містить інформацію про дату зустрічі, користувач, який знайшов, користувач, який загубив річ, місце, реалізована зустріч та успішність зустрічі;
- повідомлення – містить інформацію про текст повідомлення, відправника, отримувача, дату надсилання;
- рейтинг – містить інформацію про оцінку, того, хто залишив рейтинг та отримав.

Між сутностями у базі даних наявні 1 тип зв’язку – «один-до-багатьох». Зв'язок «один-до-багатьох» наявний між декількома парами таблиць у базі, їх буде розглянуто по порядку:

- Користувач та Допис. Кожен користувач може створити допис (або жодного) в певний проміжок часу, він не обмежений у кількості дописів, які може створити. Для формалізації зв'язку у таблиці Post є зовнішній ключ, який посилається на первинний ключ таблиці User;
- Користувач та Повідомлення. Кожен користувач може написати будь-яку кількість повідомлень, проте одне специфічне повідомлення може бути написане лише одним коритсувачем Для формалізації зв'язку у таблиці Chat є зовнішні ключі user_from. user_to, які посилаються на первинний ключ таблиці AbsUser;
- Користувач та Зустріч. Кожен користувач може бути ініціатором будь-якої кількості зустрічей, може ініціювати будь-яку кількість зустрічей, проте у зустрічі є лише два користувача – хто запропонував і хто погодився. Для формалізації зв'язку у таблиці Meetings є зовнішні ключі found_user, lost_user, які посилаються на первинний ключ таблиці User;
- Користувач та Рейтинг. Кожен користувач може бути поставити будь-яку кількість оцінок, може отримати будь-яку кількість оцінок, проте у кожної оцінки є лише два користувача – хто поставив і хто отримав. Для

формалізації зв'язку у таблиці Meetings є зовнішні ключі user_from, user_to, які посилаються на первинний ключ таблиці User.

Таблиці бази даних (рис. 2.3):

1) Users – містить інформацію про користувачів сервісу. Поля: id, amount_of_complaints, amount_of_found, amount_of_lost, is_blocked, password, email, nickname, rating, user_type;
2) Lost_and_found_board – містить інформацію про оголошення втраченого або знайденого. Поля: id, thing_id, User_id, date_if_action, date_if_post, place, is_lost;
3) Things – містить інформацію про речі, які втрачено або знайдено. Поля: id, photo, description;
4) Meetings – містить інформацію про зустрічі для передачі втрачених або знайдених речей. Поля: id, meeting_date, thing_id, founder_id, lost_id, place, meeting_realized, was_necessary_thing;
5) Chat – містить інформацію про обмін повідомленнями між користувачами.

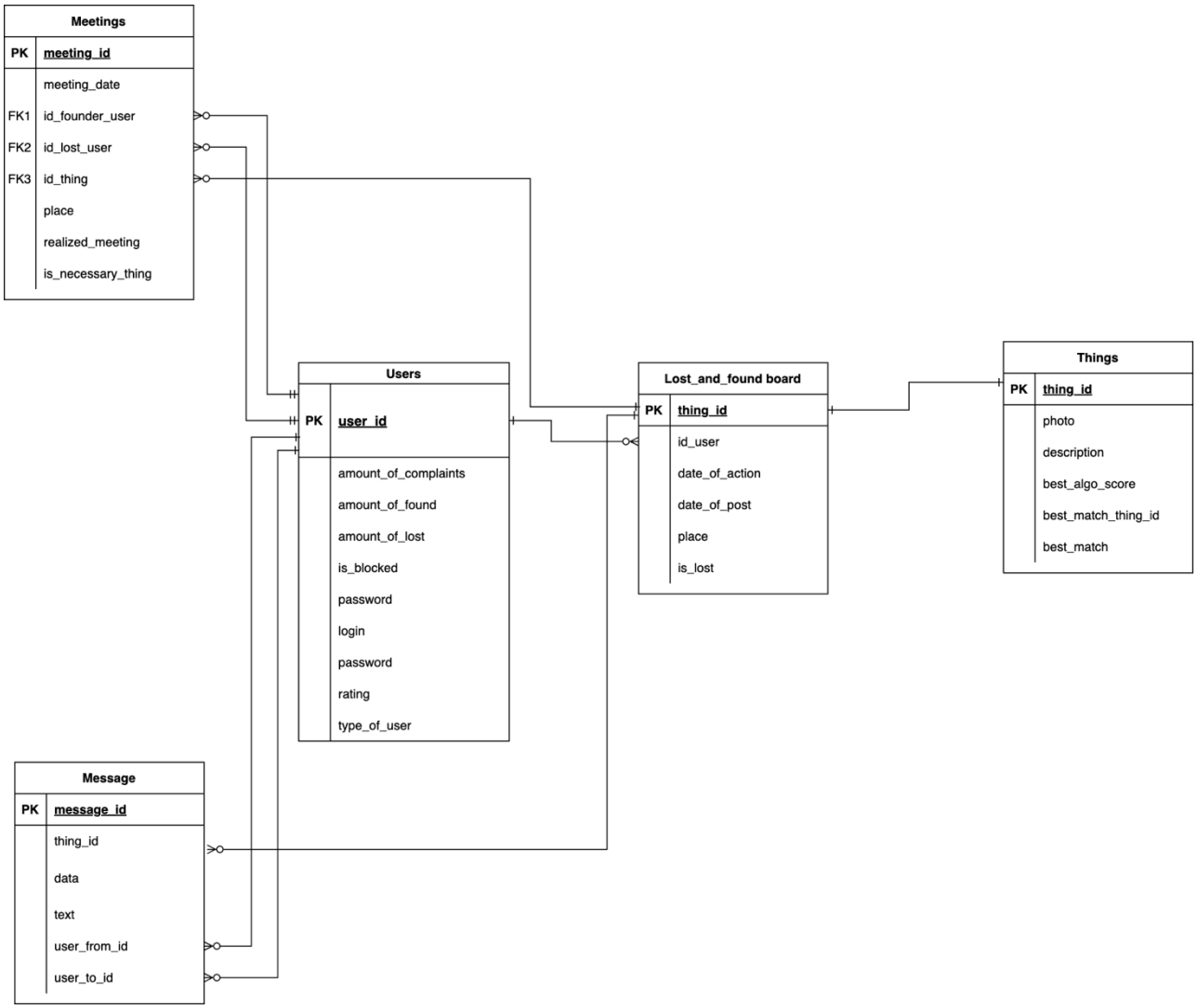

Рисунок 2.3 – Діаграма бази даних

Усі мікросервіси звертаються до однієї бази даних, так як усі необхідні обчислення для забезпечення їхньої правильної роботи були виконані за допомогою використання локальної пам’яті кожного сервісу.

2.2 Проектування бекенд архітектури

Для проектування мікросервісної бекенд архітектури, необхідно окреслити які сервіси за що відповідають:

1) Сервіс користувачів, речей, зустрічей та оголошень: основний сервіс, який найчастіше взаємодіє з іншими сервісами. Він надсилає дані до

сервісу геолокації, сервісу обробки скарг та отримує від них відповіді, в залежності від яких змінює свої дані, також він отримує дані для оновлення порядку виведення оголошень від рекомендаційного сервісу;

2) Рекомендаційний сервіс: цей сервіс цей сервіс зтягує інформацію з головного сервісу для аналізу та формування рекомендацій;
3) Сервіс геолокації: цей сервіс отримує дані від сервісу користувачів, речей, зустрічей та оголошень для обробки географічних даних та запитів щодо геолокації;
4) Сервіс обробки скарг, системи перевірок та стеження статусу, системи повідомлень та підтвердження email адреси: цей сервіс отримує запити від сервісу користувачів, речей, зустрічей та оголошень для обробки скарг, перевірки дотримання правил, стеження за статусом процесів, надсилання повідомлень користувачам та підтвердження email адрес.

На рис. 2.4 схематично проілюстровано відношення сервісів один від одного

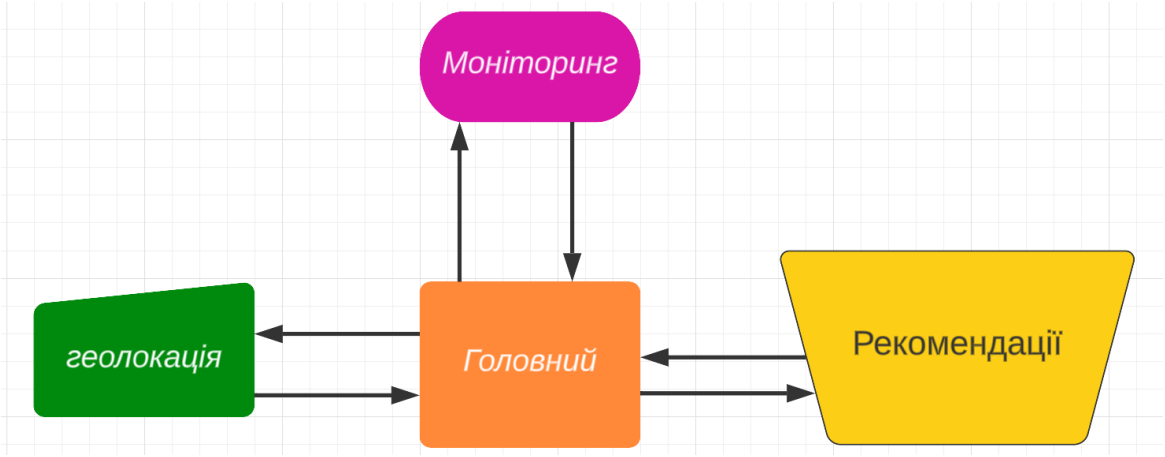

Рисунок 2.4 – Схема відношення сервісів один до одного

Під час проектування системи також необхідно врахувати певні сценарії спілкування сервісів один з одним. З цієї причини були розглянуті наступні кейси:

1) Процес авторизації (рис. 2.5): відбувається наступний процес – користувач починає реєстрацію за допомогою основного сервісу, потім відбувається перевірка листа за допомогою відповідного сервісу, в залежності від правильності відповіді, запит йде або далі до сервісу геолокації, після якого додає своє оголошення до дошки, що викликає зміну інформації у моніторингового сервісу, або закінчує свій шлях;

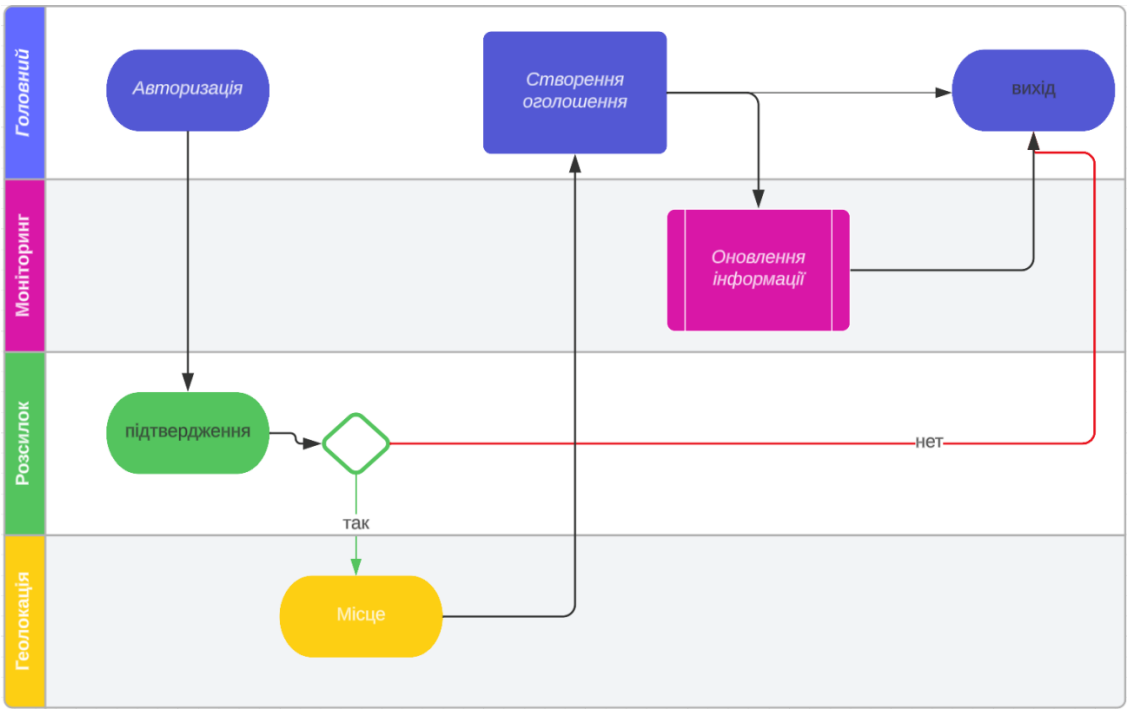

Рисунок 2.5 – Процес авторизації

2) Процес переходу на оголошення (рис. 2.6): Під час натиску на певне оголошення з дошки об’яв, оновлюється показник кількості натисків на клієнтській стороні, що спричиняє оновлення даних для рекомендаційного сервісу, далі у випадку контакту з власником оголошення, стає задіяний ще один сервіс – сервіс розсилки, який повідомляє про новий контакт, а тим часом користувач може запропонувати створити зустріч, або просто скористатись будь-яким іншим функціоналом головного сервісу, у випадку з неконтаком стосовно цього оголошення, користувач повертається назад, оновивши тим самим інформацію про це оголошення для моніторингової системи;

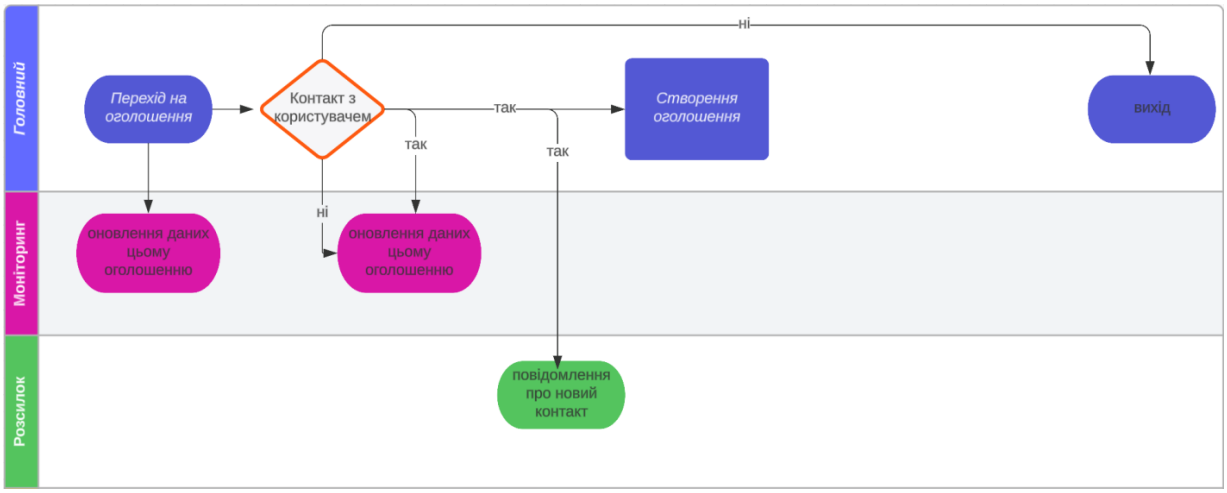

Рисунок 2.6 – Процес запиту оголошення

2.3 Проектування клієнтської підсистеми

Найважливішим аспектом веб-додатку, яким займається клієнтська підсистема, є користувацький інтерфейс. Яка б цінна інформація не знаходилась на сайті і як добре б вона не була структурована на рівні сервера, якщо інформація погано представлена, має нечитабельний формат або дизайн, то користь від такого сайту для звичайного користувача немає – він швидко відмовиться від ідеї шукати потрібну інформацію на сайті з недружнім дизайном.

Відомий факт, що хороший дизайн продукту підвищує його привабливість у вигляді споживача, а отже, і його цінність. Інформаційні продукти це стосується в тому числі і навіть більше, ніж продукти фізичного світу.

Якщо перед людиною, яка не має освіти в області дизайну і не має ресурсів для звернення до спеціаліста, стоїть завдання створення практичного і дружнього дизайну сайту, то хорошим варіантом буде грамотне використання готових рішень від професіоналів цієї справи з відкритих джерел.

З цією метою було прийнято рішення використовувати Vue.js разом з фреймворком Vuetify, який базується на принципах Material Design.

Vuetify – це бібліотека компонентів Vue.js, яка допомагає розробникам створювати веб-додатки, що відповідають принципам Material Design. Він надає широкий набір готових до використання компонентів, що полегшує розробку та підтримку додатків. Vuetify включає такі елементи:

1) Сітку для створення гнучких макетів;

2) Класи для стилізації тексту, зображень, таблиць та іншого контенту;

3) Компоненти для створення кнопок, форм, навігаційних меню, слайдерів, випадаючих списків, "акордеонів", модальних вікон, вспливаючих підказок та інших елементів інтерфейсу;

4) Класи для рішення задач, які найчастіше виникають у веб-розробників (вирівнювання тексту, приховування або відображення елементів, задання кольору і фону елементів, задання margin і padding відступів, рамок елементів, тощо).

3 ПРОГРАМНА РЕАЛІЗАЦІЯ БЕКЕНД ДОДАТКУ

Програмна реалізація бекенд-додатку для сервісу "Бюро знахідок" включає в себе ряд компонентів і технологій, які спільно працюють для забезпечення функціональності та продуктивності системи, таких як:

Python – це об’єктно-орієнтована, інтерпретована мова програмування високого рівня. Python має строгу динамічну типізацію.

Був обраний завдяки тому, що він є об'єктно-орієнтованим, має простий синтаксис та потужні бібліотеки.

Для реалізації бекенд-додатку був використан фреймворк FastAPI. Він забезпечує швидку розробку, масштабованість та підтримку сучасних веб-стандартів. FastAPI використовує асинхронну обробку запитів [2] та надає зручний інтерфейс для роботи з HTTP-запитами, маршрутизацією та валідацією даних.

Для зберігання інформації про користувачів, оголошення та інші дані, використовувані в сервісі "Бюро знахідок", була використана реляційна база даних PostgreSQL.

Для забезпечення безпеки і доступу до даних був використан метод аутентифікації JWT (JSON Web Tokens), для перевірки автентичності користувачів і надання їм доступу до захищених ресурсів.

Розглянемо програмну реалізацію кожного з сервісів, що входять до складу бекенд-додатку проекту:

1) Сервіс користувачів, речей, зустрічей та оголошень: Цей сервіс відповідає за управління всіма операціями, пов'язаними з користувачами, речами, зустрічами та оголошеннями. В програмній реалізації цього сервісу можна врахувати наступні складові:
   - Модуль користувачів: Реалізація функцій для реєстрації нових користувачів, управління профілями, автентифікації та авторизації, зміни паролю, відновлення доступу та інших операцій, пов'язаних з користувачами. На рис. 3.1 зображені усі ендпоінти необхідні для побудови даного модулю;

Рисунок 3.1 – Ендпоінти, необхідні для побудови модуля користувачів

- Модуль речей: Реалізація функцій для додавання, редагування та видалення речей. На рис. 3.2 зображені усі ендпоінти необхідні для побудови даного модулю;

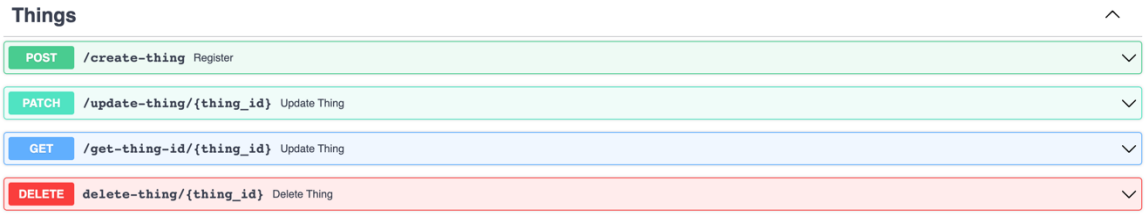

Рисунок 3.2 – Ендпоінти, необхідні для побудови модуля речі

- Модуль зустрічей: Реалізація функцій для планування, редагування та видалення зустрічей, визначення місця та часу зустрічей, спілкування між учасниками та інших операцій, пов'язаних з організацією зустрічей. На рис. 3.3 зображені усі ендпоінти необхідні для побудови даного модулю;

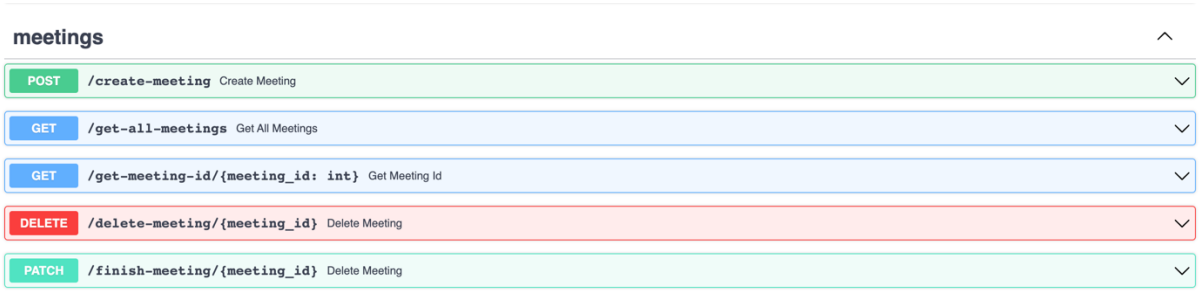

Рисунок 3.3 – Ендпоінти, необхідні для побудови модуля зустрічі

- Модуль оголошень: Реалізація функцій для створення, редагування та видалення оголошень, пошуку та фільтрації за різними критеріями, взаємодії з користувачами та інших операцій, пов'язаних з оголошеннями. На рис. 3.4 зображені усі ендпоінти необхідні для побудови даного модулю;

Рисунок 3.4 – Ендпоінти, модуля дошка оголошень

Для побудови цього сервісу була вжита наступна файлова структура (рис. 3.5, рис. 3.6), вона дозволила розподілити увесь функціонал сервісу між класами, для яких були створені окремі файли, таким чином покращуючи зрозумілість та подальшу можливість вживання коду.

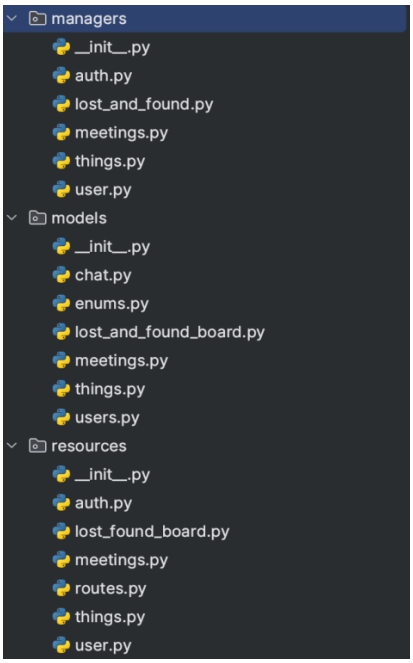

Рисунок 3.5 – Структура головного сервісу

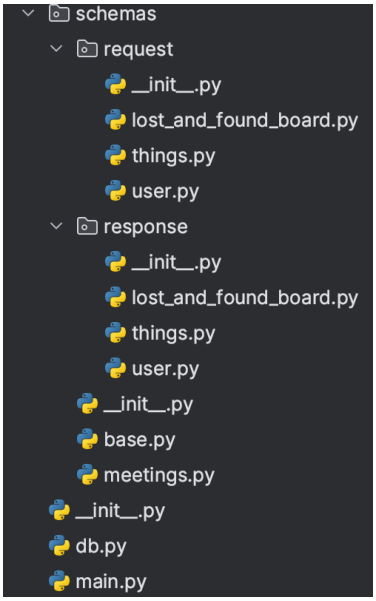

Рисунок 3.6 – Структура головного сервісу ч.2

На прикладі модулю оголошень (рис. 3.7, 3.8, 3.9), слід розглянути структуру коду, який надає можливість реалізувати функціонал, необхідний для впровадження всього функціоналу цього модуля.

Спочатку необхідно розглянути файли з папки schemes – вони відповідають за формат виведення та вводу даних для модуля дошки оголошення, за це відповідають відповідні класи BoardIn – для введення та BoardOut – для виводу, де також для більш зручного виводу інформації використовується для відображення даних про користувача, за допомогою класу UserInsideScheme, це стало можливим завдяки тому, що ці класи наслідуються від класу BaseModel пакету pydantic, який знає інформацію про схеми класів, які використовуються у додатку, таким чином, замість просто user_id при виклику певного ендпоінту, де буде використовуватися BoardOut в якості класу для відображення, буде вся інформація про користувача яка описана у класі UserInsideScheme, а саме: id, username, email.

```python
from datetime import datetime, timezone
from typing import Optional, Dict

from pydantic import BaseModel


2 usages
class BoardIn(BaseModel):
    thing_id: int
    geolocation: Dict[str, float]
    user_id: int
    date_of_action: Optional[datetime]
    place: str
    is_lost: bool


2 usages
class BoardUpdateIn(BaseModel):
    place: str
```

Рисунок 3.7 – Фрагмент файлу відповідального за схеми до модуля дошки оголошень

```python
6 usages
class BoardOut(BaseModel):
    id: int
    thing: Optional[Thing]
    date_of_action: datetime
    date_of_post: datetime
    place: str
    is_lost: bool
    user: Optional[UserInsideScheme]

    class Config:
        orm_mode = True
```

Рисунок 3.8 – Фрагмент файлу відповідального за схеми до модуля дошки оголошень

Також варто відмітити клас BoardManager (рис. 3.9), який відповідає за всю комунікацію з базою даних модуля дошки оголошень та саме за допомогою нього відбувається увесь функціонал, який направляє дані для відображення до класів, які були описані вище, такі як: створити оголошення, отримати оголошення за id, оновити дошку, отримати порядок виводу оголошень з сервісу для рекомендацій, увесь код, який описує усі інші модулі знаходиться у додатку А, так як він є типовим.

```python
5 usages
class BoardManager:
    1 usage
    @staticmethod
    async def create_board_announcement(db: Session, announcement_data):
        new_thing = BoardDB(**announcement_data)
        db.add(new_thing)
        db.commit()
        db.refresh(new_thing)
        return new_thing

    1 usage
    @staticmethod
    async def get_board_by_id(db: Session, announcement_id):
        return db.query(BoardDB).filter(BoardDB.id == announcement_id).first()

    1 usage
    @staticmethod
    async def get_all_board(db: Session):
        print(db.query(BoardDB).all())
        return db.query(BoardDB).all()

    1 usage
    @staticmethod
    async def update_board_announcement(db: Session, announcement_id, thing_data: str):
        board_instance = db.query(BoardDB).filter(BoardDB.id == announcement_id).first()
        if board_instance:
            board_instance.place = thing_data
            db.add(board_instance)
            db.commit()
            db.refresh(board_instance)
            return board_instance
        return None
```

Рисунок 3.9 – Код відповідальний за агрегацію даних та доступ до бази даних модуля дошки оголошень

2) Рекомендаційний сервіс: Цей сервіс відповідає за обробку запитів на рекомендації засновані на описах предметів. Він відповідає за сортування оголошень у певному порядку зважаючи на зібрані та проаналізовані дані про активність їх авторів та тих, кого вони могли зацікавити. Для реалізації даного сервісу була вжита наступна файлова структура проекту (рис. 3.10) та необхідний функціонал був побудований за допомогою наступних ендпоінтів (рис. 3.11).

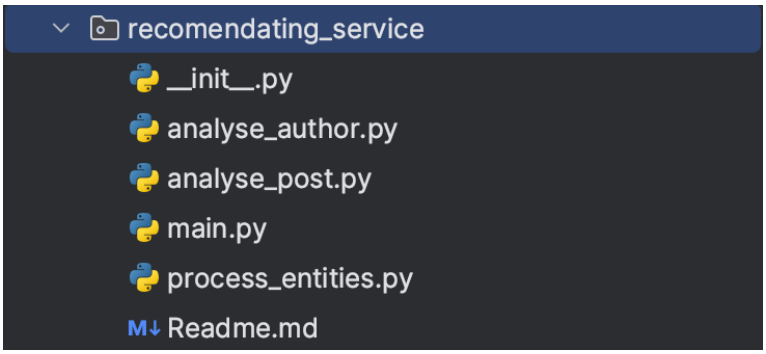

Рисунок 3.10 – Файлова структура рекомендаційного сервісу

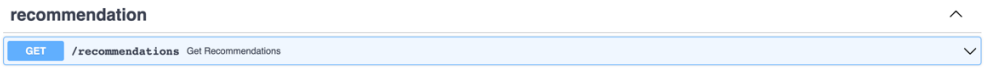

Рисунок 3.11 – Ендпоінти рекомендаційного сервісу

Особливістю цього сервісу є те, що до нього не звертаються інші сервіси, він проводить оновлення системи рекомендацій раз на 2 години, таким чином змінюючи порядок показу оголошень, це було досягнуто за допомогою пакетів schedule та threading, які дозволяють використувати заплановані події та багатопоточність відповідно.

3) Сервіс геолокації: Цей сервіс відповідає за обробку географічних даних та запитів щодо геолокації. В програмній реалізації цього сервісу були враховані наступні складові:

- Модуль збору та зберігання даних: збір та зберігання географічних даних користувачів, речей та інших елементів системи;
- Модуль геокодування та розрахунку відстаней: реалізація геокодування адрес, перетворення географічних координат в адреси та розрахунку відстаней між різними точками на мапі;
- Модуль збору та зберігання даних.

Унікальністю даного сервісу є те, що усі 4 ендпоінти (рис. 3.12) опрацьовуються за домогою одного класу, що дозволяю мінімізувати його файлову структуру (рис. 3.13).

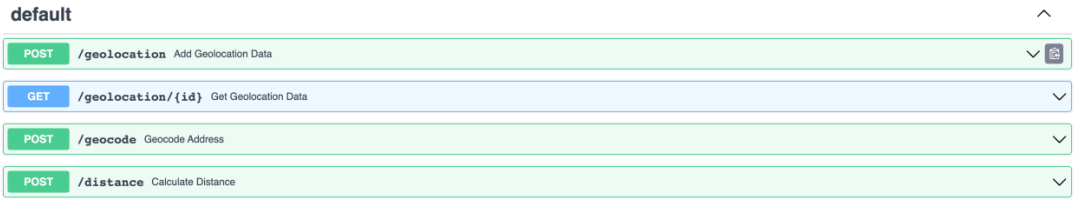

Рисунок 3.12 – Ендпоінти геолокаційного сервісу

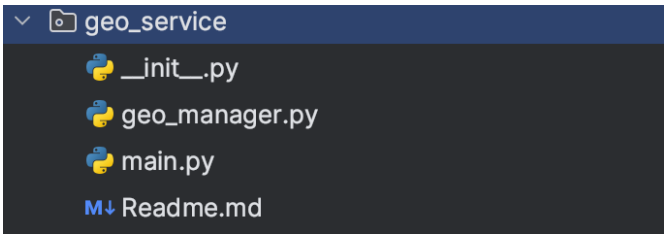

Рисунок 3.13 – Файлова структура геолокаційного сервісу

4) Сервіс обробки скарг, системи перевірок та стеження статусу, системи повідомлень та підтвердження email адреси: цей сервіс відповідає за управління обробкою скарг, системою перевірок та стеженням статусу, системою повідомлень та підтвердженням email адреси. В програмній реалізації цього сервісу були враховані наступні складові:

- Модуль системи стеження статусу: Реалізація системи для стеження статусу процесів, пов'язаних з користувачами, речами, зустрічами та оголошеннями;
- Модуль обробки скарг та перевірок: Прийом та обробка скарг, перевірка дотримання правил та стандартів, реєстрація скарг та результатів перевірок;
- Модуль системи повідомлень: Реалізація системи повідомлень для надсилання сповіщень користувачам про різноманітні події та оновлення;
- Модуль роботи з електронною поштою: Реалізація механізму підтвердження email адреси для нових користувачів, механізму надсилання повідомлення про новий контакт та видалення оголошення.

Основним механізмом комунікації між мікросервісами у FastAPI є взаємодія за допомогою HTTP запитів. Одним з підходів є використання HTTP запитів між мікросервісами. Кожен мікросервіс може мати власні маршрути та ендпоінти, до яких можна здійснювати HTTP запити з інших мікросервісів. Це здійснюється за допомогою модуля , ось приклад (рис. 3.14) встановлення такого зв’язку між сервісами: головним та тим, що відповідає за розсилку пошти під час авторизації користувача, і таким чином з’являється можливість провалідувати існування доступу до пошти у користувача.

```python
@router.post("/register")
async def register(user_data: UserRegisterIn, db: Session = Depends(get_db)):
    token = await UserManager.register(db, user_data.dict())
    response = requests.get(f"http://localhost:5000/verify", params=generated_6_digit())
    if not verify_6_digit(response):
        raise HTTPException(status_code=404, detail="Post not found")

    return {"token": token}
```

Рисунок 3.14 – Авторизація користувача за допомогою додаткового сервісу

Для забезпечення роботи данного сервісу були реалізовані наступні ендпоінти. Для роботи модуля пов’язаного з поштою (рис. 3.15).

Рисунок 3.15 – Ендпоінти модуля пошти

Для роботи модуля пов’язаного з моніторингом (рис. 3.16).

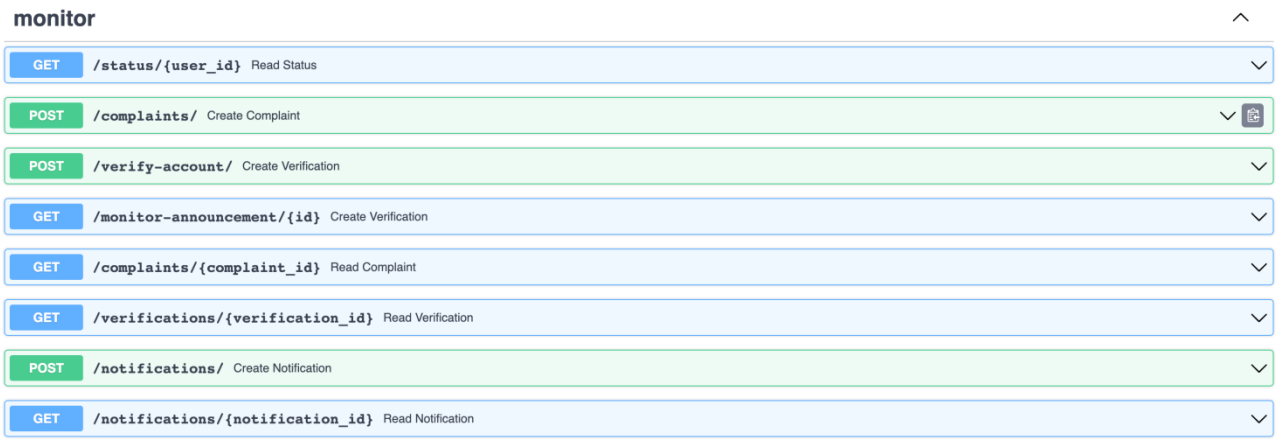

Рисунок 3.16 – Ендпоінти модуля моніторинга

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ФРОНТЕНД ДОДАТКУ

Програмна реалізація фронтенд-додатку для сервісу "Бюро знахідок" включає в себе ряд компонентів і технологій, які спільно працюють для забезпечення функціональності та продуктивності системи, таких як:

1) Vue.js [3] – це прогресивний фреймворк JavaScript, використаний для створення користувацьких інтерфейсів. Його ядро, що зосереджено лише на представленні даних, легко інтегрується з іншими бібліотеками або існуючими проектами. Vue.js також надає можливості для створення складних односторінкових додатків при додаванні додаткових бібліотек і модулів;
2) Vue Router [4] – це офіційна бібліотека маршрутизації для Vue.js. З його допомогою розробники можуть створювати односторінкові додатки, що можуть перемикатись між різними компонентами на основі стану URL. Він дозволяє управляти активними компонентами, на основі поточного шляху URL, а також управляти історією перегляду;
3) Vuex [5] – це стан додатку + бібліотека шаблонів для Vue.js. Він служить як централізований магазин для всіх компонентів в системі, з правилами забезпечення що стан може змінюватися тільки у прогнозований спосіб. Він також інтегрується з розширенням розробника Vue, що надає передові можливості налагодження та перегляду стану;
4) Axios [6] – це популярна бібліотека JavaScript, яка використовується для здійснення HTTP-запитів. Axios пропонує обширний API для роботи з Promise API, що забезпечує гнучкість в обробці HTTP запитів;
5) Vuetify – це Vue UI Library, яка допомагає розробникам при створенні продуктивних та атрактивних інтерфейсів користувачів, які відповідають специфікаціям Material Design. Вона має багатий набір

готових компонентів та інструментів, що дозволяють економити час на розробку та забезпечують консистентність стилю;

6) Vue Google Maps – це компонент для роботи з Google Maps в Vue.js. Він включає в себе ряд підкомпонентів, що дозволяють максимально гнучко використовувати можливості Google Maps API [7]. За допомогою Vue Google Maps можна легко виводити мапи, маркери, полілінії, круги, прямокутники та інші геометричні фігури, а також працювати з подіями на карті. З його допомогою можна реалізувати пошук місця згуби/знахідки, відображення маршрутів, роботу з геолокацією користувача та інші функції, що залучають використання мап.

Скомбінований використання цих інструментів та бібліотек у Vue.js додатку "Бюро знахідок" дозволяє створити продуктивний, інтерактивний та дружній до користувача інтерфейс. Розробка фронтенду з використанням цих технологій не тільки прискорює процес розробки, але і сприяє створенню масштабованих та легко підтримуваних веб-додатків.

У Vue.js код будується за компонентами, приклад його побудови наведений у додатку Б, де кожен з компонент відповідає певній вебсторінці, тому вигляд файлової структури (рис. 4.1) може дати багато інформації стосовно її функціоналу, можна зробити висновок, що є основні компоненти, які ми розглянемо далі: WelcomePage, MapChoose, TheWholeBoard, TheItem, Register, VerifyAccount, FoundLostChoice.

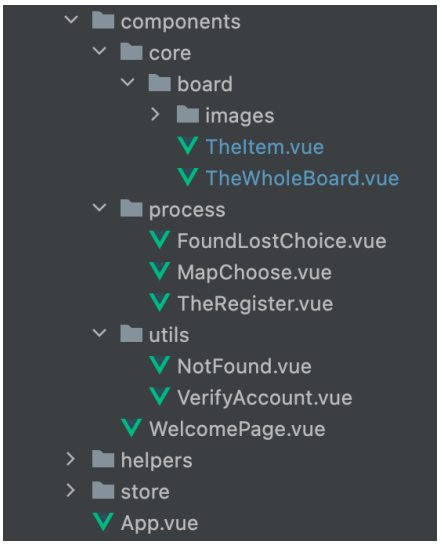

Рисунок 4.1 – Файловая структура фронтенд проекту

За вхідну точку у проект відповідає компонент WelcomePage, який має наступний вигляд (рис. 4.2), за допомогою цього компонента користувач заповнює поле опис речі, він має у собі поле пошуку та основний текст, який допомагає зрозуміти користувачу що потрібно йому робити.

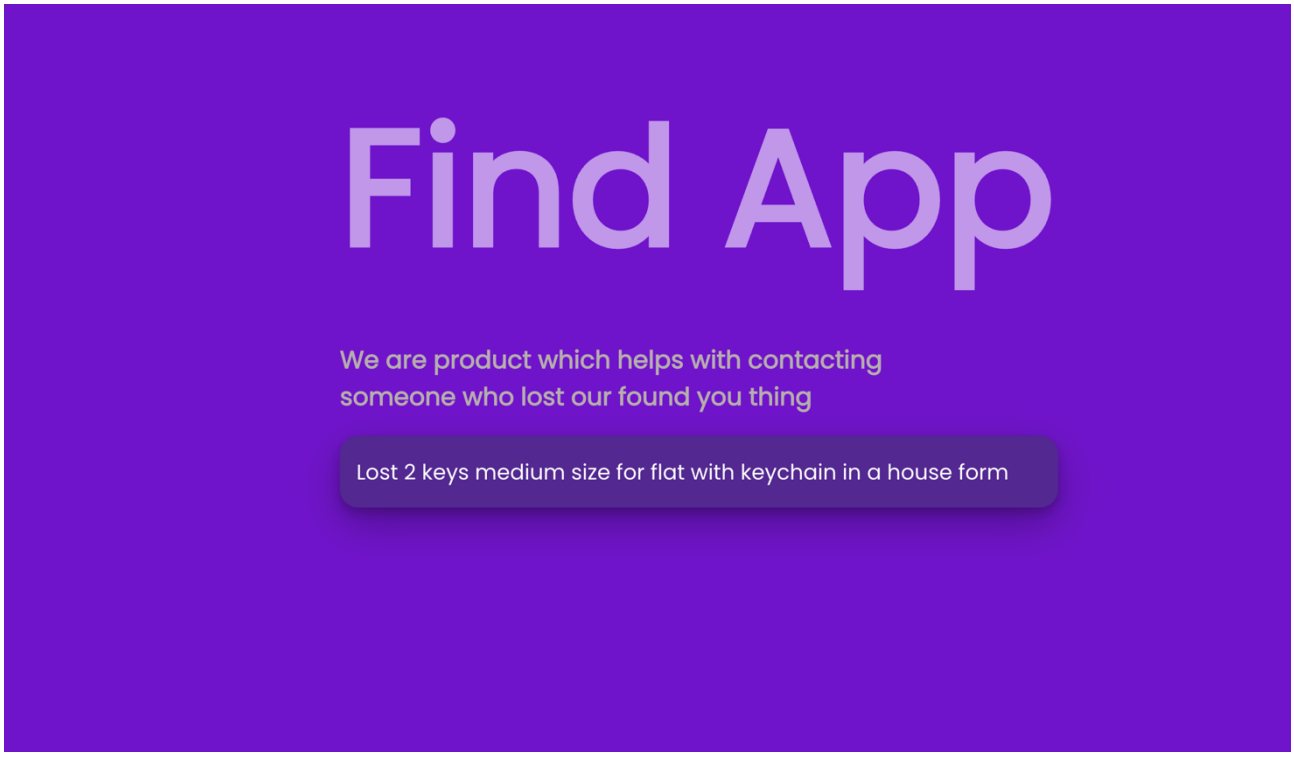

Рисунок 4.2 – Зовнішній вигляд компоненту Welcome Page

Після цього викликається компоненти MapChoose (рис.4.3), який співпрацює з географічним сервісом та використовує Google Maps Api для відмальовування мапи разом з маркером який користувач встановлює та відповідає за обробку наступних показників – статус (загублений/знайдений), місце де це трапилось та фото речі за допомогою відповідних елементів мови HTML.

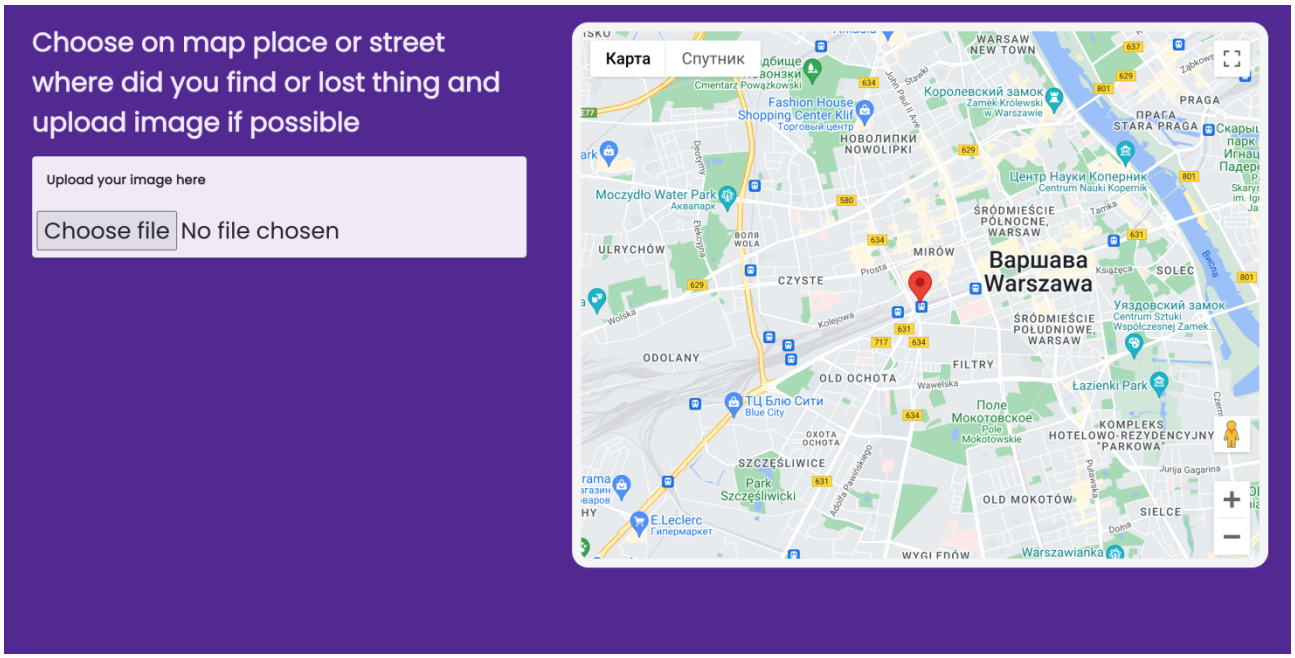

Рисунок 4.3 – Зовнішній вигляд компоненту MapChoose

Після цього викликається компонент реєстрації Register (рис. 4.4), який відповідає за реєстрацію користувача та відправляє запит до головного сервіса, який просто сервіс з надсилання пошти відправити лист для підтвердження на вказану адресу. Цей компонент має у собі 3 поля для вписання даних, та 2 кнопки: одна – для переходу на сторінку авторизацію, інша для надсилання запиту до головного сервісу про створення нового користувача.

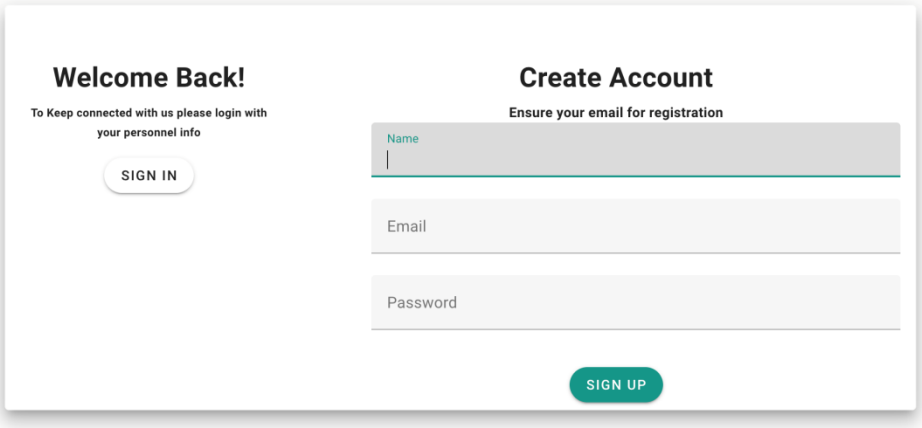

Рисунок 4.4 – Зовнішній вигляд компоненту Register

Далі користувач отримує листа на електрону пошту за допомогою сервісу розсилки (рис. 4.5) та переходить на наступний компонент Verify у якому заповнює форму перевірки, для найкращої користувацького досвіду в цьому компоненті налаштований автоперехід на наступну цифру після введення попередньої (рис. 4.6).

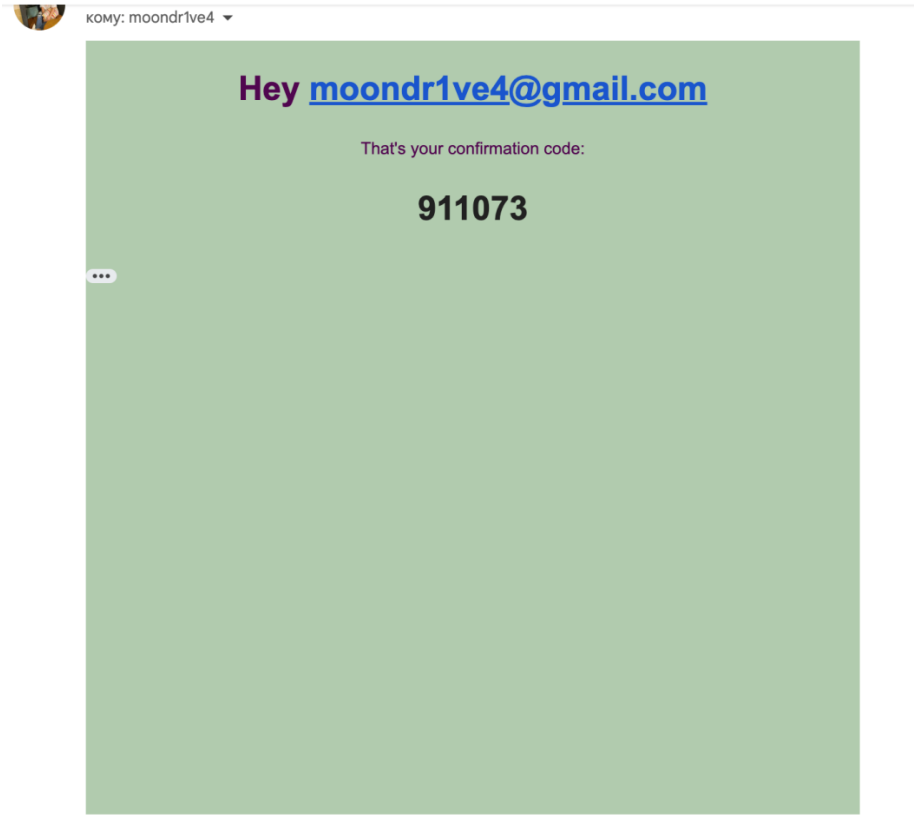

Рисунок 4.5 – Зовнішній вигляд листа, який приходить на пошту користувачу

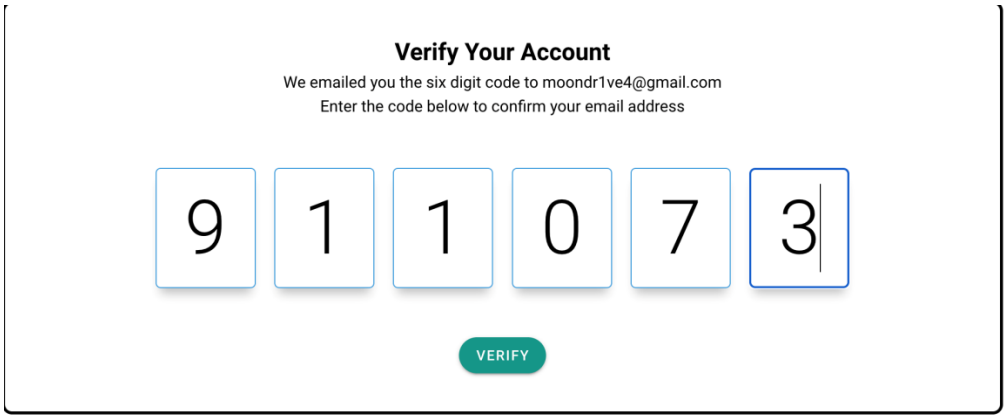

Рисунок 4.6 – Зовнішній вигляд компоненту Verify

Після цих дій, відправляється звертання до головного сервісу та сервісу рекомендацій про створення нової речі і таким чином вона з’являється на дошці об’яв (рис. 4.7), за замочуванням сортування відбувається за датою додання оголошення, внаслідок комунікації фронтенда з головним сервісом, також він отримує доступ до фільтра ції за тегами (рис. 4.8) та перегляду повної інформації про річ (рис. 4.9). Для інтеракції з кожним з цих компонентів відправляється новий запит до головного сервісу за допомогою бібліотеки axios.

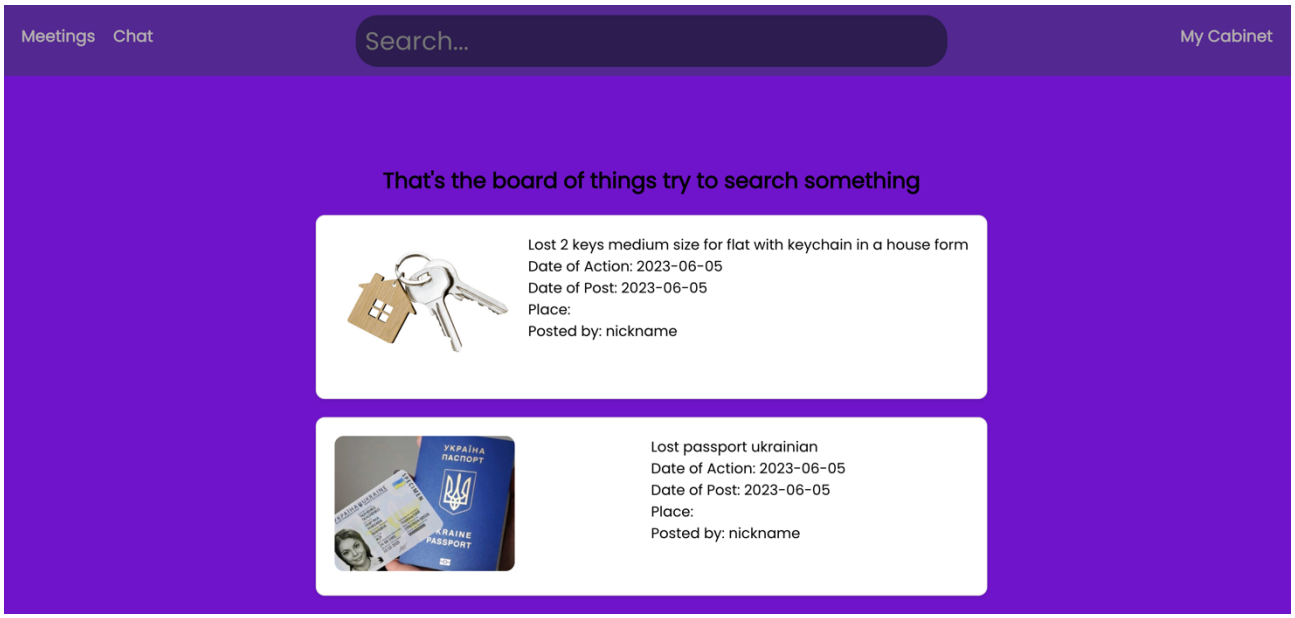

Рисунок 4.7 – Зовнішній вигляд компоненту дошки об’яв

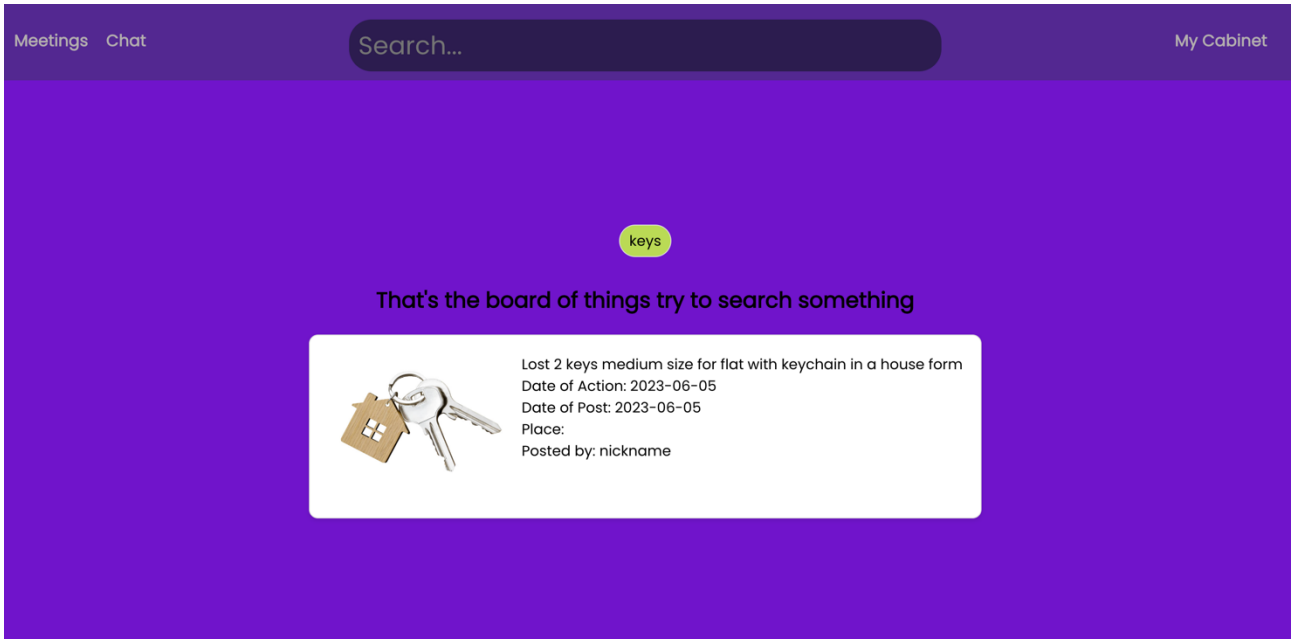

Рисунок 4.8 – Зовнішній вигляд компоненту дошки об’яв з ввімкнутим тегом

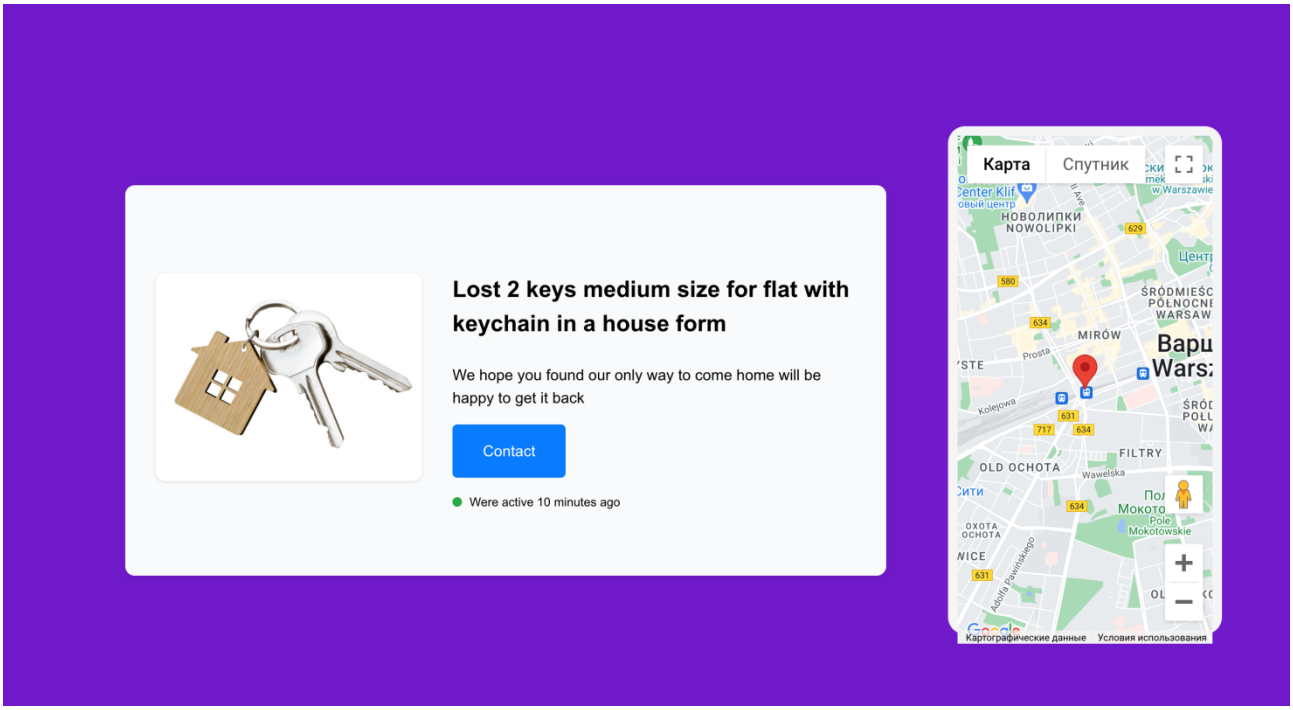

Рисунок 4.9 – Зовнішній вигляд компоненту оголошення

ВИСНОВКИ

У рамках дипломної роботи були виконані наступні завдання:

1) проведений аналіз предметної області розробки інфраструктури сервісу Бюро знахідок;
2) спроектована інфраструктура сервісу Бюро знахідок;
3) розроблена інфраструктура сервісу Бюро знахідок;
4) протестована функціональність інфраструктури сервісу Бюро знахідок.

Розроблена програмна реалізація бекенд-додатку з використанням мови програмування Python та фреймворку FastAPI. Спроектована та розгорнута база даних PostgreSQL для зберігання даних користувачів, знайдених та загублених речей, оголошень, скарг, перевірок, повідомлень та іншої інформації. Розроблено API для взаємодії з бекенд-додатком.

Розроблена програмна реалізація фронтенд-додатку за допомогою фреймворку Vue.js. Використані бібліотеки Vue Router для реалізації маршрутизації між компонентами, Vuex для управління станом додатку, Vuetify для реалізації інтерфейсу, Axios для виконання HTTP запитів до сервера. Також, використано Vue Google Maps для візуалізації місць знахідок та згуб.

Забезпечена безпека і доступ до даних за допомогою методу аутентифікації JWT (JSON Web Tokens). Реалізовано функціонал реєстрації, авторизації та аутентифікації користувачів.

Розроблені автоматизовані тести для перевірки функціональності та стійкості системи.

В результаті було створено гнучку, продуктивну, інтерактивну та дружню до користувача систему, що масштабується і легко підтримується. Проект має потенціал для подальшого розвитку та вдосконалення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FastAPI. Офіційна документація. [Електронний ресурс]. — Режим доступу: https://fastapi.tiangolo.com/

2. Building APIs with FastAPI. [Електронний ресурс]. — Режим доступу: https://www.udemy.com/course/building-apis-with-fastapi-and-mongodb/

3. Vue.js. Офіційна документація. [Електронний ресурс]. — Режим доступу: https://vuejs.org/v2/guide/

4. Vue Router. Офіційна документація. [Електронний ресурс]. — Режим доступу: https://router.vuejs.org/

5. Vuex. Офіційна документація. [Електронний ресурс]. — Режим доступу: https://vuex.vuejs.org/

6. Axios: Зробити HTTP-запити з Vue.js. [Електронний ресурс]. — Режим доступу: https://vuejs.org/v2/cookbook/using-axios-to-consume-apis.html

7. Vue.js і аутентифікація користувачів. [Електронний ресурс]. — Режим доступу: https://auth0.com/blog/beginner-vuejs-tutorial-with-user-login/

8. Vue Google Maps. Офіційна документація. [Електронний ресурс]. — Режим доступу: https://vue-map.netlify.app/

ДОДАТОК А
Код необхідний для побудови модуля дошка оголошень

```
@router.post("/create-announcement")
async def create_lost_and_found(board_data: BoardIn, db: Session =
Depends(get_db)):
    """
    :param board_data: Data necessary to create new announcement\n
    :param db: >>>\n
    :return: New announcement
    """
    lost_and_found = await BoardManager.create_board_announcement(db,
board_data.dict())
    return {"data": lost_and_found}


@router.get("/get-whole-board", response_model=List[BoardOut])
async def get_whole_board(db: Session = Depends(get_db)) -> List[BoardOut]:
    return await BoardManager.get_all_board(db)


@router.patch("/update-announcement-place/{board_id: int}",
response_model=BoardOut)
async def update_announcement(board_id: int, board: BoardUpdateIn, db: Session =
Depends(get_db)):
    return await BoardManager.update_board_announcement(db, board_id,
board.place)


@router.get("/get-announcement-id/{board_id: int}")
async def get_announcement_id(board_id: int, db: Session = Depends(get_db)):
    if (board_item := await BoardManager.get_board_by_id(db, board_id)) is None:
        raise HTTPException(status_code=404, detail="Announcement not found")
    return board_item
```

Лістинг А.1 – Файл який описує ендпоінти, необхідні для роботи з дошкою оголошень

```
import requests
from sqlalchemy.orm import Session

from app.models import BoardDB


class BoardManager:
    @staticmethod
    async def create_board_announcement(db: Session, announcement_data):
        new_thing = BoardDB(**announcement_data)
        db.add(new_thing)
        db.commit()
        db.refresh(new_thing)
        return new_thing

    @staticmethod
    async def get_board_by_id(db: Session, announcement_id):
        return db.query(BoardDB).filter(BoardDB.id == announcement_id).first()

    @staticmethod
    async def get_all_board(db: Session):
```

```
        print(db.query(BoardDB).all())
        return db.query(BoardDB).all()

    @staticmethod
    async def update_board_announcement(db: Session, announcement_id,
thing_data: str):
        board_instance = db.query(BoardDB).filter(BoardDB.id ==
announcement_id).first()
        if board_instance:
            board_instance.place = thing_data
            db.add(board_instance)
            db.commit()
            db.refresh(board_instance)
            return board_instance
        return None

    @staticmethod
    async def update_order(db: Session):
        order = requests.get('http://localhost:5001/get-order')
        return db.query(BoardDB).sort(order=order)
```

Лістинг А.2 – Файл який описує яким чином відбувається виклик даних з ендпоінтів

```
class UserInsideScheme(UserBase):
    id: int
    nickname: str
    amount_of_complaints: int
    amount_of_found: int
    amount_of_lost: int
    is_blocked: bool
    rating: Optional[int] = None
    class Config:
        orm_mode = True

class Thing(BaseModel):
    id: int
    photo: str
    description: str
    best_match: Optional[str] = None
    class Config:
        orm_mode = True

class BoardOut(BaseModel):
    id: int
    thing: Optional[Thing]
    date_of_action: datetime
    date_of_post: datetime
    place: str
    is_lost: bool
    user: Optional[UserInsideScheme]
    class Config:
        orm_mode = True
```

Лістинг А.3 – Файл, який зображує принцип виводу інформації за допомогою схем

ДОДАТОК Б

Код необхідний для побудови компонентів Vue.js

```
<template>
  <div class="main-container">
  <section class="product">
    <div class="product__image">
      <img
        :src="image"
        alt="Razer Kraken Kitty Edt Gaming Headset Quartz"
        class="product__img"
      />
    </div>

    <div class="product__details">

      <h2 class="product__title">
        Lost 2 keys medium size for flat with keychain in a house form
      </h2>

      <div class="product__pricing">

        <p class="product__offer">
          We hope you found our only way to come home will be happy to get it
back
        </p>
      </div>

      <button class="product__add-btn">
              Contact</button>

      <div class="product__stock">
        <div class="product__stock-indicator"></div>
        <span class="product__stock-text">Were active 10 minutes ago</span>
      </div>


    </div>
  </section>


    <div class="map-container">
      <GoogleMap
        ref="map"
        api-key=""
        style="width: 100%; height: 65vh;"
        :center="center"
        :zoom="13"
        @click="addMarker"
      >
        <MarkerCluster>
          <Marker
            v-for="(location, i) in locations"
            :options="{ position: location }"
            :key="i"
          />
          <Marker
            v-for="(marker, index) in markers"
            :options="{ position: marker.position }"
            :key="'marker_' + index"
            @click="showPopup(marker)"
          />
        </MarkerCluster>
```

```
      </GoogleMap>
    </div>
  </div>


</template>

<script>
import {defineComponent, ref } from 'vue'
import {GoogleMap, Marker, MarkerCluster} from 'vue3-google-map'
import image from '@/components/core/board/images/headphone.jpeg';
import image2 from '@/components/core/board/images/heart.jpeg';
import image3 from '@/components/core/board/images/weight.png';

export default defineComponent({
  name: 'TheItem',
  components: {GoogleMap, Marker, MarkerCluster},
  setup() {
    const center = {lat: 52.225955, lng: 20.989853}

    const locations = [
      {lat: 54.218878, lng: 24.017450},
      {lat: 52.418878, lng: 21.017450},
      {lat: 52.818878, lng: 21.017450},
      {lat: 52.118878, lng: 21.017450},
      {lat: 52.918878, lng: 21.017450},
      {lat: 53.218878, lng: 21.017450},
      {lat: 52.225955, lng: 20.989853},
      {lat: 51.298878, lng: 21.017450},
      {lat: 51.718878, lng: 21.017450},
    ]

    const markers = ref([])
    const showInfo = ref(false)
    const selectedMarker = ref(null)
    const showPopupOverlay = ref(false)

    const addMarker = (event) => {
      console.log('added marker')
      markers.value.push({position: event.latLng})
      const marker = markers.value[markers.value.length - 1];
      showPopup(marker)
    }

    const showPopup = (marker) => {
      console.log('show popup')

      selectedMarker.value = marker
      showInfo.value = true
    }

    const hidePopup = () => {
      console.log('hide popup')

      selectedMarker.value = null
      showInfo.value = false
    }

    const blockMap = () => {
      console.log('block map')

      showPopupOverlay.value = false
    }
```

```
    const unblockMap = () => {
      console.log('unblock map')

      showPopupOverlay.value = false
    }

    return {
      center,
      locations,
      markers,
      showInfo,
      selectedMarker,
      addMarker,
      showPopup,
      hidePopup,
      blockMap,
      unblockMap,
      showPopupOverlay,
      image,
      image2,
      image3
    }
  },
})
</script>


<style scoped>

.map-container {
  float: right;
  width: 20%;
  height: 65vh;
  border-radius: 18px;
  border: 10px solid #f5f5f5;
  margin-left:65px;
}

.main-container{
  display: flex;
  justify-content: center;
  align-items: center;
  position: fixed;
  background-color: rgb(112, 21, 203); /* purple background */
  margin: 0;
  width: 100%;
  height: 100vh;
  //max-width:5000px;
  box-sizing: border-box;

}

.product {
  background-color: white;
  border-radius: 10px;
  overflow: hidden;
  max-width: 800px; /* adjust as needed */
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  display: flex;
  align-items: center;
  justify-content: center;
  min-height: 50vh;
  max-height: 100vh;
```

```
  background-color: #f8f9fa;
  padding: 2em;
  box-sizing: border-box;
  font-family: Arial, sans-serif;
}

.product__image {
  margin-right: 2em;
}

.product__img {
  width: 280px;
  object-fit: cover;
  border-radius: 10px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
  transition: all 0.2s ease-in-out;
}

.product__img:hover {
  transform: scale(1.05);
}

.product__details {
  flex: 1;
}

.product__tag {
  display: inline-block;
  padding: 0.5em 1em;
  font-size: 0.8em;
  color: white;
  background-color: #007bff;
  border-radius: 50px;
}

.product__title {
  margin: 1em 0;
  font-size: 1.5em;
}

.product__price {
  font-size: 1em;
}

.product__price--old {
  text-decoration: line-through;
  color: #6c757d;
}

.product__price--new {
  font-size: 2em;
  color: #28a745;
  font-weight: bold;
}

.product__quote("#offer-info {\n  font-size: 14px;\n", "color: #6c757d;\n}")

quote("#add-button {\n  width: 100%;\n", "transition: all 0.15s;\n}")

quote("#add-button:hover {\n", "box-shadow: 0px 10px 15px rgba(0,0,0,0.1);\n}")

quote("#secondary-buttons{\n  flex-direction: row;\n", "justify-content: space-
evenly;\n}")
```

```
quote(".secondary-button {\n  display: flex;\n", "transition: all 0.15s;\n}")

quote(".secondary-button:hover {\n", "transform: translateY(-2px);\n}")

quote("#cart-icon,\n#wishlist-icon {\n", "width: 32px;\n}")
.offer {
  font-size: 0.8em;
  color: #6c757d;
}

.product__add-btn {
  display: inline-block;
  margin: 1em 0;
  padding: 1em 2em;
  font-size: 1em;
  color: white;
  background-color: #007bff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: all 0.15s ease-in-out;
}

.product__add-btn:hover {
  background-color: #0056b3;
}

.product__stock {
  display: flex;
  align-items: center;
  gap: 0.5em;
}

.product__stock-indicator {
  width: 10px;
  height: 10px;
  background-color: #28a745;
  border-radius: 50%;
}

.product__stock-text {
  font-size: 0.8em;
}

.product__actions {
  display: flex;
  gap: 1em;
  margin-top: 2em;
}

.product__action-btn {
  display: flex;
  align-items: center;
  gap: 0.5em;
  padding: 1em;
  font-size: 0.8em;
  color: #343a40;
  background-color: white;
  border: 1px solid #ced4da;
  border-radius: 5px;
  cursor: pointer;
  transition: all 0.15s ease-in-out;
}
```

```
.product__action-btn:hover {
  background-color: #f8f9fa;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}

.product__action-icon {
  width: 20px;
  height: 20px;
}

</style>
```

Лістинг Б.1 – Код для роботи класу TheItem.vue, який відповідає за зображення певного оголошення з дошки оголошень

```
<template>
  <div class="main-container">
  <div class="navbar">
      <nav>
        <ul class="left-nav">
          <li><a href="#meetings">Meetings</a></li>
          <li><a href="#chat">Chat</a></li>
        </ul>
      </nav>
      <div class="search-container">
              <input class="search-bar" type="text" v-model="searchTerm"
placeholder="Search...">
              <div v-if="searchTerm" class="suggestions" v-click-
outside="hideSuggestions">
                <div
                    v-for="tag in filteredTags"
                    :key="tag"
                    class="suggestion"
                    @click="addTag(tag)"
                >
                  {{ tag }}
                </div>
              </div>
      </div>

      <nav>
        <ul class="right-nav">
          <li><a href="#mycabinet">My Cabinet</a></li>
        </ul>
      </nav>
    </div>


    <div class="tag-container">
      <div
          v-for="tag in selectedTags"
          :key="tag"
          class="tag selected"
          @click="removeTag(tag)">
        {{ tag }}
      </div>
    </div>

    <div class="board-info">

    <h2>
      That's the board of things try to search something
    </h2>
```

```
    </div>


    <div class="announcement-board">
      <div class="announcement" v-for="announcement in filteredAnnouncements"
:key="announcement.id" @click="handleItemClick(announcement.id)">
        <div class="photo">
          <img :src="announcement.thing.photo" alt="Photo">
        </div>
        <div class="details">
          <p>{{ announcement.thing.description }}</p>
          <p>Date of Action: {{ addDateFormatting(announcement.date_of_action)
}}</p>
          <p>Date of Post: {{ addDateFormatting(announcement.date_of_post)
}}</p>
          <p>Place: {{ announcement.place }}</p>
          <p>Posted by: {{ announcement.user.nickname }}</p>
        </div>
      </div>
    </div>
  </div>
</template>

<script>
import { formatDate } from "@/helpers/formatDate";
import image from '@/components/core/board/images/headphone.jpeg';

export default {
  data() {
    return {
      tags: ['keys', 'id', 'wallet', 'bag', 'money'],
      selectedTags: [],
      searchTerm: '',
    };
  },
  computed: {
    filteredAnnouncements() {
        return this.announcements;
      },

    filteredTags() {
      let search = this.searchTerm.toLowerCase();
      return this.tags.filter(tag => tag.toLowerCase().includes(search));
    },

    computedDateOfAction(date) {
      return formatDate(date)
    },

    computedDateOfPost(date) {
      return formatDate(date)
    }
  },
  methods: {
    handleItemClick(id) {
      this.$router.push(`/board-item/${id}`)
    },
    addTag(tag) {
      if (!this.selectedTags.includes(tag)) {
        this.selectedTags.push(tag);
      }
      this.searchTerm = '';
    },
    removeTag(tag){
```

```
      let index = this.selectedTags.indexOf(tag);
      if (index > -1) {
        this.selectedTags.splice(index, 1);
      }
    },
    hideSuggestions() {
      console.log("Should be hidden")
      this.searchTerm = '';
    },
    addDateFormatting(date) {
      return formatDate(date)
    }
  }
}
</script>
<style scoped>
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@200;400&display=swap'
);

* {
  box-sizing: border-box;
  font-family: 'Poppins', sans-serif;

}
.navbar {
  display: flex;
  justify-content: space-between;
  align-items: center;
  position: fixed;
  top: 0;
  width: 100%;
  z-index: 1000;
  background-color: #532891;
  padding: 10px;
  font-size:2.2vh

}

.search-container {
  padding-top: 10px;
}


nav {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px;
  width: 90%;
  margin: auto;
}

nav ul {
  display: flex;
  gap: 20px;
}

nav ul.left-nav li,
nav ul.right-nav li {
  list-style: none;
}

nav ul.left-nav li a,
```

```
nav ul.right-nav li a {
  text-decoration: none;
  color: #b4afaf;
  font-weight: 600;
}
.right-nav {
  margin-left: auto;
  margin-right: 10px
}
.announcement-board {
  display: grid;
  gap: 20px;
  padding: 20px;
  grid-template-columns: repeat(auto-fill, minmax(500px, 1fr));
}

.announcement {
  display: grid;
  gap: 15px;
  grid-template-areas:
      'photo details';
  border: 1px solid #ddd;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
  background-color: #fff;
  transition: all 0.3s ease;
}

.announcement .photo {
  grid-area: photo;
  max-width: 200px;
}

.announcement .details {
  grid-area: details;
}

.announcement img {
  max-width: 100%;
  height: auto;
  border-radius: 10px;
}


.search-bar {
  padding: 10px;
  border: none;
  border-radius: 25px;
  width: 80vh;
  height: 7vh;
  outline: none;
  background-color: #2d1c4f;
  color: #b4afaf;
  font-size:3.4vh
}
.search-bar input::placeholder {
  color: #b4afaf;
;
}



.main-container{
```

```
  background-color: rgb(112, 21, 203);
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  height: 100vh;
  margin: 0;
  padding: 0;
}

.board-info{
  display: flex;
  justify-content: center;
  margin-top: 10px;
  padding-top: 20px;
}
.tag-container {
  margin-top:20px;
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
}
.tag {
  padding: 5px 10px;
  border: 1px solid #ddd;
  border-radius: 20px;
  cursor: pointer;
  background: #eee;
  transition: all 0.3s;
}
.tag:hover {
  background: #ddd;
}
.tag.selected {
  background: #bada55;
}
.suggestions {
  position: absolute;
  width: 80vh;
  max-height: 150px;
  overflow-y: auto;
  background: #fff;
  border: 1px solid #ddd;
  border-radius: 5px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.1);
  z-index: 1000;
}
.suggestion {
  padding: 5px 10px;
  cursor: pointer;
  border-bottom: 1px solid #ddd;
}
.suggestion:last-child {
  border-bottom: 0;
}
.suggestion:hover {
  background: #ddd;
}
</style>
```

Лістинг Б.2 – Код для роботи класу TheWholeBoard.vue, який відповідає за зображення дошки оголошень