КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра Математичної Інформатики

«До захисту допущено»
Завідувач кафедри
В. М. Терещенко
_____ (підпис)
«_____» _____ 2025 р.

Дипломна робота

на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА ВЕБ-ПЛАТФОРМИ ДЛЯ КОМУНІКАЦІЇ ТА ОБМІНУ ЗНАННЯМИ В СПІЛЬНОТІ БДЖОЛЯРІВ

Виконав студент 2 курсу
Андращук Едуард Олександрович

_____ (підпис)

Науковий керівник:
доктор технічних наук
Заславський Володимир Анатолійович

_____ (підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____ (підпис)

Київ – 2025

Реферат

Магістерська робота присвячена проектуванню та розробці повностекового веб-застосунку *Beekeepers Community Platform* — інтегрованої платформи, призначеної для української спільноти бджолярів з метою сприяння ефективній комунікації, обміну спеціалізованими знаннями та надання інструментів для управління пасіками і полями.

У роботі здійснено аналіз предметної області, розглянуто існуючі рішення для нішевих онлайн-спільнот та інструменти для агросектору. Обґрунтовано вибір сучасного технологічного стеку, що включає React (з TypeScript, Vite, Material-UI, Redux Toolkit) для розробки клієнтської частини, NestJS (Node.js, Fastify, TypeScript) для серверної логіки, та MongoDB (з Mongoose) як документо-орієнтовану базу даних з підтримкою GeoJSON.

Спроектовано архітектуру системи, що базується на принципах модульності та RESTful API. Визначено функціональні та нефункціональні вимоги. Розроблено ключові модулі: система автентифікації користувачів (локальна реєстрація, верифікація email з можливістю повторного надсилання листа, Google OAuth, JWT для авторизації), форум для обговорень, структурована база знань, та багатофункціональна інтерактивна карта. Картографічний модуль дозволяє користувачам додавати, переглядати, редагувати метадані та видаляти об’єкти пасік (вулики) і полів, а також візуалізує попередження про заплановані обробки полів шляхом динамічного кольорового кодування.

Описано процес розробки, організацію кодової бази, ключові аспекти реалізації API та інтерфейсу користувача, включаючи використання Leaflet для картографії. Розглянуто питання безпеки, валідації даних та інтернаціоналізації.

Проведено тестування основних функціональних можливостей. За результатами роботи сформульовано висновки щодо успішної реалізації прототипу платформи та окреслено перспективні напрямки для її подальшого розвитку та вдосконалення.

Ключові слова: веб-застосунок, спільнота бджолярів, React, NestJS, MongoDB, інтерактивна карта, Leaflet, GeoJSON, управління пасіками, автентифікація, форум, база знань, RTK Query, Material-UI.

Зміст

Вступ

Актуальність теми магістерської роботи зумовлена зростаючою потребою у спеціалізованих онлайн-платформах для нішевих спільнот, зокрема для бджолярів. Бджільництво є важливою галуззю сільського господарства та екології, і ефективний обмін знаннями, досвідом та оперативною інформацією між пасічниками може суттєво сприяти його розвитку. Існуючі загальні соціальні мережі та форуми не завжди враховують специфічні потреби бджолярської спільноти, такі як обговорення хвороб бджіл, методів догляду, медоносних рослин, а також координація дій щодо обробки полів та розташування пасік.

Метою даної магістерської роботи є розробка фулстек веб-застосунку *Beekeepers Community Platform*, що надасть бджолярам зручні інструменти для комунікації, обміну інформацією, доступу до бази знань та управління даними про власні пасіки та сільськогосподарські угіддя.

Для досягнення поставленої мети було визначено наступні завдання:

- Провести аналіз предметної області та існуючих рішень.
- Обґрунтувати вибір технологічного стеку для розробки.
- Спроектувати архітектуру клієнтської та серверної частин застосунку, а також схему бази даних.
- Реалізувати основний функціонал платформи, включаючи систему реєстрації та автентифікації, форум, базу знань та інтерактивну карту.
- Забезпечити базові механізми безпеки та валідації даних.
- Розробити інтуїтивно зрозумілий та адаптивний користувацький інтерфейс.

Об’єктом дослідження є процес проектування та розробки повностекового веб-застосунку для нішевої спільноти.

Предметом дослідження є архітектурні рішення, технології та інструменти для створення інтерактивної та функціональної платформи для бджолярів, що включає засоби комунікації, обміну знаннями та геоінформаційні функції.

Методи дослідження, що використовувалися в роботі, включають:

- Аналіз науково-технічної літератури та існуючих аналогів.
- Системний аналіз та проектування програмного забезпечення.
- Об’єктно-орієнтоване програмування.

- Використання сучасних веб-технологій та фреймворків: React для розробки клієнтської частини, NestJS (Node.js) для серверної частини, MongoDB як система управління базами даних.

- Застосування бібліотек Material-UI для користувацького інтерфейсу та Leaflet для реалізації картографічного функціоналу.

Наукова новизна

Наукова новизна даної роботи полягає у наступному:

- Розроблено концепцію та реалізовано прототип інтегрованої веб-платформи для нішевої спільноти бджолярів, що поєднує засоби соціальної комунікації (форум), обміну спеціалізованими знаннями (база знань) та інструменти геоінформаційного менеджменту (інтерактивна карта пасік та полів). На відміну від загальних соціальних мереж або окремих ГІС-інструментів, запропонована платформа надає комплексне рішення, адаптоване до специфічних потреб бджолярів України.

- Запропоновано та реалізовано підхід до візуалізації на інтерактивній карті критично важливої для бджолярів інформації, такої як заплановані дати обробки сільськогосподарських полів, шляхом динамічного кольорового кодування полігонів, що підвищує обізнаність та сприяє попередженню ризиків для бджолосімей.

- Обґрунтовано вибір та продемонстровано ефективність застосування сучасного стеку веб-технологій (React, NestJS, Leaflet, MongoDB) для створення масштабованої та функціональної платформи, що підтримує специфічні вимоги агро-геоінформатики в контексті бджільництва, спираючись на існуючі дослідження щодо цифрової трансформації сільського господарства та управління онлайн-спільнотами [17, 8, 6].

Практичне значення отриманих результатів полягає у створенні готового до використання прототипу веб-платформи, що може бути впроваджена для підтримки спільноти бджолярів, покращення їх взаємодії та доступу до актуальної інформації.

Апробація результатів роботи

Результати роботи були представлені у вигляді тез доповіді на конференції ”Формування сучасної науки: методика та практика 23.05.2025, м. Київ.

Структура роботи: Магістерська робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1

Аналіз предметної області

1.1 Огляд існуючих рішень

Даний розділ присвячено аналізу існуючих веб-платформ та мобільних застосунків, що можуть використовуватися спільнотами для обміну інформацією, а також спеціалізованих рішень для бджолярів та ширшого агросектору. Розглядаються популярні форумні системи, соціальні мережі для груп за інтересами, а також застосунки, що пропонують інструменти для ведення пасіки чи моніторингу умов медозбору. Важливість розробки таких систем підкреслюється дослідженнями в галузі агро-геоінформатики, що фокусуються на зборі, обробці та візуалізації геопросторових даних для підтримки прийняття рішень в сільському господарстві [5]. Прикладами спеціалізованих ІКТ-проектів для бджільництва є ініціативи зі створення відкритих екосистем обміну даними [6], дослідження цифрової трансформації галузі через архітектури підтримки прийняття рішень [8], а також проекти, спрямовані на розробку сервісів для розумного управління пасіками, зокрема в країнах, що розвиваються [20]. Дослідження в галузі онлайн-спільнот підкреслюють важливість дизайну, теорії та практики для їх успішного функціонування та розвитку [17]. Аналіз включає порівняння їх функціональних можливостей, переваг та недоліків у контексті потреб української спільноти бджолярів.

1.2 Порівняння технологій для веб-розробки

Вибір технологічного стеку є ключовим етапом розробки будь-якого програмного продукту. Для проекту *Beekeepers Community Platform* розглядалися наступні категорії технологій:

1.2.1 Технології фронтенду

Для розробки клієнтської частини веб-застосунку основними кандидатами були такі JavaScript-бібліотеки та фреймворки як React, Angular та Vue.js.

1.2.1.1 React

React є популярною JavaScript-бібліотекою для створення користувацьких інтерфейсів, розробленою Facebook [11]. Її перевагами є компонентний підхід, велика екосистема, значна спільнота розробників, гнучкість та висока продуктивність завдяки використанню

віртуального DOM. Для даного проекту React був обраний через його здатність ефективно будувати складні та інтерактивні інтерфейси, велику кількість готових компонентів (зокрема, інтеграція з Material-UI та картографічними бібліотеками як Leaflet), а також через наявний досвід розробника з цією технологією.

1.2.1.2 Angular

Angular — це комплексний фреймворк від Google, що надає повний набір інструментів для створення великих корпоративних застосунків. Хоча Angular є потужним рішенням, для даного проекту його всеохоплююча структура та дещо вищий поріг входження були розцінені як надлишкові.

1.2.1.3 Vue.js

Vue.js — прогресивний JavaScript-фреймворк, відомий своєю простотою інтеграції та низьким порогом входження. Він є гарним вибором для багатьох проектів, однак екосистема React та доступність спеціалізованих бібліотек для React виявилися більш привабливими для специфічних завдань проекту (наприклад, використання RTK Query для управління станом API).

1.2.2 Технології бекенду

Для серверної частини розглядалися Node.js-фреймворки NestJS та Express.js, а також Django на Python.

1.2.2.1 NestJS (Node.js)

NestJS — це прогресивний фреймворк для створення ефективних, масштабованих серверних застосунків на Node.js [14]. Він побудований з використанням TypeScript та поєднує елементи об'єктно-орієнтованого програмування, функціонального програмування та функціонально-реактивного програмування. Архітектура NestJS, що базується на модулях, контролерах та сервісах, сприяє чіткій організації коду та його підтримці. Вбудована підтримка TypeScript, легка інтеграція з Passport.js для автентифікації, Swagger для документації API та TypeORM/Mongoose для роботи з базами даних роблять його чудовим вибором для розробки REST API. Обраний для проекту завдяки модульній структурі, підтримці TypeScript та хорошій інтеграції з інструментами, необхідними для проекту.

1.2.2.2 Express.js (Node.js)

Express.js є мінімалістичним та гнучким Node.js веб-фреймворком. Він надає базовий набір функцій для веб-застосунків та API, але вимагає більше ручного налаштування та структурування проекту порівняно з NestJS. Для проекту, що передбачає розширення функціоналу, більш структурований підхід NestJS був визнаний кращим.

1.2.2.3 Django (Python)

Django — високорівневий Python веб-фреймворк, що заохочує швидку розробку та чистий, прагматичний дизайн. Він включає багато готових компонентів, таких як ORM, адміністративна панель тощо. Однак, для даного проекту перевага була надана стеку на JavaScript/TypeScri для забезпечення однорідності технологій на фронтенді та бекенді.

1.2.3 Системи управління базами даних

Вибір СУБД ґрунтувався на потребах гнучкості схеми даних та роботи з геопросторовими даними.

1.2.3.1 MongoDB

MongoDB — це документо-орієнтована NoSQL база даних [12]. Її перевагами є гнучка схема, що дозволяє легко еволюціонувати структуру даних, горизонтальна масштабованість та вбудована підтримка геопросторових запитів, що є важливим для функціоналу карти вуликів та полів. Обрана для проекту завдяки гнучкості, підтримці GeoJSON та добрій інтеграції з Node.js через Mongoose.

1.2.3.2 PostgreSQL

PostgreSQL — потужна об'єктно-реляційна СУБД, відома своєю надійністю, відповідністю стандартам SQL та розширюваністю (наприклад, PostGIS для геоданих). Хоча PostgreSQL є чудовим вибором для багатьох застосунків, для даного проекту з потенційно змінною структурою даних спільноти та геопросторовими об'єктами, гнучкість MongoDB була визнана більш пріоритетною.

1.3 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку для проекту *Beekeepers Community Platform* ґрунтувався на аналізі вимог, доступності інструментів, досвіді розробника та прагненні створити сучасний, масштабований та підтримуваний веб-застосунок. Було обрано наступний стек:

- **Клієнтська частина (Frontend):** React [11] з TypeScript, зібраний за допомогою Vite [21], обрано через його популярність, велику екосистему, компонентний підхід та ефективність у створенні динамічних користувацьких інтерфейсів. Material-UI (MUI) [13] використано як бібліотеку UI компонентів для швидкої розробки адаптивного та естетично привабливого дизайну, що відповідає сучасним веб-стандартам. React Router використано для навігації, а Redux Toolkit (зокрема RTK Query) [18] — для управління станом та взаємодії з API.

- **Серверна частина (Backend):** NestJS (побудований на Node.js та Express/Fastify [4]) з TypeScript обрано завдяки його модульній архітектурі, яка сприяє чіткій організації коду, вбудованій підтримці TypeScript, що підвищує надійність, та легкій інтегра-

ції з іншими інструментами, такими як Passport.js [7] для автентифікації та Swagger (на основі OpenAPI [16]) для автоматичної генерації документації API. Використання Fastify адаптера забезпечує високу продуктивність.

- **База даних:** MongoDB [12] обрано як NoSQL документо-орієнтовану базу даних. Її гнучка схема даних є перевагою для проекту, де структура інформації може розвиватися. Вбудована підтримка GeoJSON та геопросторових індексів є критично важливою для реалізації функціоналу інтерактивної карти. Mongoose [10] використано як ODM для взаємодії з MongoDB з боку NestJS.

- **Картографічний сервіс:** Leaflet разом з React-Leaflet обрано як легку та гнучку бібліотеку для відображення інтерактивних карт та маніпуляції геопросторовими даними (маркери, полігони) [1], що відповідає сучасним тенденціям застосування геоінформаційних технологій для візуалізації даних в агросекторі [5].

- **Інтернаціоналізація:** i18next [9] з react-i18next використано для підтримки багатомовності інтерфейсу.

- **Контейнеризація:** Docker [2] та Docker Compose використовуються для створення консистентного середовища розробки та спрощення розгортання застосунку.

- **Платформа розгортання та база даних у хмарі:** Для розгортання застосунку було обрано PaaS-платформу Render [19] завдяки її простоті використання, інтеграції з GitHub для автоматичного розгортання (CI/CD), автоматичному налаштуванню HTTPS та можливості розгортання як статичних сайтів (фронтенд), так і Docker-контейнерів (бекенд). Для бази даних було обрано MongoDB Atlas [12] (як хмарний сервіс) через його переваги як керованої бази даних, що включають автоматичне резервне копіювання, масштабування та вбудовані засоби безпеки, що дозволило зосередитися на розробці застосунку, а не на адмініструванні інфраструктури.

Даний стек технологій дозволяє ефективно розробляти як клієнтську, так і серверну частини, забезпечуючи при цьому хорошу продуктивність, масштабованість та можливості для подальшого розвитку платформи.

РОЗДІЛ 2

Проектування системи

2.1 Функціональні та нефункціональні вимоги

Проектування веб-застосунку *Beekeepers Community Platform* базувалося на визначенні ключових функціональних та нефункціональних вимог, що забезпечують його корисність, надійність та зручність для користувачів.

2.1.1 Функціональні вимоги

- **Реєстрація та автентифікація користувачів:** Можливість створення облікового запису з використанням електронної пошти та паролю, верифікація email через надсилання підтверджувального листа (токен дійсний 1 годину), можливість повторного надсилання листа верифікації, а також автентифікація за допомогою облікового запису Google (OAuth 2.0).
- **Управління профілем користувача:** Перегляд та редагування базової інформації профілю (наприклад, біографія, місцезнаходження, експертиза).
- **Форум для обговорень:** Створення нових тем для обговорення, публікація повідомлень у темах, можливість залишати коментарі до повідомлень, система вподобань (лайків) для постів.
- **База знань:** Доступ до каталогу статей та ресурсів з бджільництва, можливість пошуку та фільтрації матеріалів за категоріями (поточна реалізація з mock-даними).
- **Інтерактивна карта (Управління пасіками та полями):**
  - Відображення карти (Україна за замовчуванням) з використанням Leaflet.
  - Додавання точкових маркерів для вуликів із зазначенням назви, нотаток та автоматичним визначенням геолокації.
  - Відображення вуликів з кастомними іконками (MUI HiveIcon).
  - Видалення маркерів вуликів з картографічного інтерфейсу (з підтвердженням).
  - Додавання полігональних об’єктів для полів із зазначенням назви, типу культури, періоду цвітіння та списку запланованих дат обробки.
  - Редагування метаданих існуючих полів (назва, тип культури, період цвітіння, дати обробки).

  - Динамічне візуальне виділення полів на карті різними кольорами (наприклад, червоний, помаранчевий, синій) залежно від статусу та терміновості запланованих обробок.
  - Відображення детальної інформації (метаданих) при виборі об'єкта на карті у спливаючих вікнах (Popups).
  - Фільтрація об'єктів на карті за різними критеріями (тип культури, період цвітіння для полів; тип вулика, стан для пасік) – (заплановано/майбутній функціонал).
  - Можливість отримання сповіщень про заплановані обробки полів поблизу пасік (майбутній функціонал).
- **Адміністрування користувачів (для ролі Адміністратор):**
  - Перегляд списку всіх зареєстрованих користувачів системи.
  - Можливість зміни ролі користувача (надання/скасування прав адміністратора).
- **Інтелектуальний помічник FAQ (на базі ШІ):**
  - Можливість для користувачів ставити питання природною мовою стосовно функціоналу платформи або загальних тем бджільництва (обмежено наданим контекстом).
  - Отримання відповідей, згенерованих моделлю OpenAI (наприклад, GPT-3.5-turbo) на основі попередньо визначеного набору ЧаПи (FAQ) або даних з бази знань.
  - Інтерфейс для введення питання та відображення відповіді.
- **Інтернаціоналізація:** Підтримка декількох мов інтерфейсу (українська, англійська).

2.1.2 Нефункціональні вимоги

- **Продуктивність:** Забезпечення прийнятного часу завантаження сторінок та швидкої відповіді сервера на запити користувачів.
- **Безпека:** Захист облікових записів користувачів (хешування паролів), валідація вхідних даних на клієнті та сервері, використання HTTPS у продакшн-середовищі, захист від поширених веб-вразливостей (наприклад, XSS через використання React, який екранує дані за замовчуванням).
- **Масштабованість:** Архітектура застосунку повинна дозволяти майбутнє масштабування для обслуговування зростаючої кількості користувачів та обсягів даних.
- **Надійність:** Система повинна бути доступною та стабільно працювати.
- **Зручність використання (Usability):** Інтерфейс має бути інтуїтивно зрозумілим, легким у навігації та адаптивним для різних розмірів екранів (десктоп, мобільні пристрої).
- **Підтримуваність коду:** Кодова база повинна бути добре структурованою, документованою (де необхідно) та легкою для модифікації та розширення.

2.2 Діаграми варіантів використання (Use Case Diagrams)

Тут мають бути представлені діаграми варіантів використання, що ілюструють взаємодію користувачів (акторів) із системою. Наприклад, для незареєстрованого користувача (перегляд публічного контенту, реєстрація), зареєстрованого користувача (вхід, участь у форумі, робота з картою, перегляд бази знань, управління профілем) та адміністратора (якщо передбачено).

2.3 Архітектура системи

Розроблений веб-застосунок має класичну клієнт-серверну архітектуру.

2.3.1 Загальна архітектура

Система складається з трьох основних компонентів:

- Клієнтська частина (Frontend): односторінковий застосунок (SPA), розроблений на React, відповідає за користувацький інтерфейс та взаємодію з користувачем.
- Серверна частина (Backend): REST API, розроблене на NestJS, відповідає за бізнес-логіку, обробку запитів, взаємодію з базою даних та автентифікацію.
- База даних: MongoDB, документо-орієнтована NoSQL база даних, використовується для зберігання всієї інформації застосунку, включаючи дані користувачів, пости форуму, статті бази знань, а також геопросторові дані для карти.

Взаємодія між клієнтом та сервером відбувається за протоколом HTTP(S) через RESTful API. Для розгортання використовується Docker, що забезпечує ізоляцію та портативність середовища.

2.3.2 Архітектура фронтенду

Клієнтська частина, розроблена на React [11] з використанням TypeScript, становить основу користувацького досвіду платформи. TypeScript забезпечує статичну типізацію, що суттєво підвищує надійність коду та спрощує рефакторинг у великому проекті, особливо при роботі з типами пропсів компонентів, станів та структур даних API. Компонентний підхід React дозволив створити модульну та легко підтримувану структуру UI, де складні інтерфейси, такі як інтерактивна карта або діалогові вікна, розбиваються на менші, незалежні та повторно використовувані компоненти (наприклад, `AddHiveDialog`, `EditFieldDialog`).

Управління станом, особливо станом, що надходить з сервера, реалізовано за допомогою Redux Toolkit [18], зокрема його інструменту RTK Query. Цей вибір дозволив значно спростити логіку взаємодії з REST API бекенду, автоматизувавши процеси запиту даних, їх кешування, оновлення при змінах (invalidatesTags) та обробку станів завантаження та помилок. Для таких сутностей, як вулики, поля та профілі користувачів, RTK Query ав-

томатично генерує хуки (наприклад, `useGetHivesQuery`, `useAddHiveMutation`), що мінімізує кількість шаблонного коду.

Для побудови користувацького інтерфейсу було обрано бібліотеку компонентів Material-UI (MUI) [13]. Широкий набір готових, кастомізованих та адаптивних компонентів (кнопки, форми, діалоги, сітки, іконки) прискорив розробку та забезпечив консистентний візуальний стиль, що відповідає принципам Material Design. Можливості темизації MUI також були використані для адаптації колірної схеми до тематики платформи.

Навігація в рамках односторінкового застосунку (SPA) реалізована за допомогою бібліотеки React Router. Вона дозволяє визначати маршрути для різних сторінок (наприклад, `/map`, `/profile`, `/forums`) та управляти переходами між ними без перезавантаження сторінки, що є стандартом для сучасних веб-застосунків.

Картографічний функціонал, що є центральним для платформи, побудований на базі бібліотеки Leaflet [1] та її React-обгортки React-Leaflet. Це поєднання надає декларативний спосіб інтеграції інтерактивних карт в React-компоненти, дозволяючи легко управляти шарами, маркерами (як стандартними, так і кастомними), полігонами та спливаючими вікнами.

Інтернаціоналізація інтерфейсу для підтримки української та англійської мов реалізована за допомогою бібліотеки i18next [9] та її інтеграції з React (react-i18next), що дозволяє зберігати текстові ресурси в окремих файлах та динамічно змінювати мову застосунку.

Загальна структура коду фронтенду організована за функціональними та типовими ознаками, включаючи директорії для компонентів (загальних та специфічних для модулів, наприклад `map/`), сторінок, API-сервісів (зрізів RTK Query у `store/api/`), кастомних хуків, контекстів (наприклад, `AuthContext`) та утиліт.

2.3.3 Архітектура бекенду

Серверна частина розроблена на платформі Node.js з використанням фреймворку NestJS [14] та TypeScript. Ключовою перевагою NestJS для даного проекту є його модульна архітектура, що забезпечує чітке розділення відповідальностей та сприяє високій підтримуваності коду. Кожен основний функціональний блок платформи, такий як автентифікація (`AuthModule`), управління користувачами (`UsersModule`), форумом (`ForumModule`), картографічними об'єктами (`HivesModule`, `FieldsModule`), та нещодавно доданий FAQ-асистент (`FaqModule`), реалізований як окремий модуль NestJS.

Кожен такий модуль інкапсулює власні компоненти:

- **Контролери (Controllers):** Обробляють вхідні HTTP-запити, валідують їх (часто за допомогою DTO та автоматичних пайпів валідації NestJS), викликають відповідні методи сервісів та повертають відповіді клієнту. Наприклад, `HivesController` містить ендпоінти для створення, отримання, оновлення та видалення вуликів.

- **Сервіси (Services):** Містять основну бізнес-логіку модуля. Вони взаємодіють з базою даних через моделі Mongoose, виконують операції з даними, реалізують специфічні для домену правила та логіку. Наприклад, `AuthService` відповідає за валідацію ко-

ристувачів, генерацію JWT токенів та взаємодію з `UsersService` для створення нових користувачів або перевірки їх статусу.

- **Об'єкти Передачі Даних (Data Transfer Objects - DTOs):** Використовуються для визначення структури даних, що передаються між клієнтом та сервером (в тілах запитів) або між різними шарами застосунку. DTO, визначені за допомогою класів TypeScript та декораторів з бібліотеки `class-validator`, дозволяють автоматично валідувати вхідні дані на рівні контролера за допомогою вбудованого в NestJS `ValidationPipe`. Це забезпечує, що до сервісів потрапляють лише коректно сформовані дані, підвищуючи надійність API. Прикладом є `CreateHiveDto`, що валідує поля, необхідні для створення нового вулика.

- **Схеми Mongoose (Schemas):** Визначають структуру документів у відповідних колекціях MongoDB та правила їх валідації на рівні бази даних. Наприклад, `HiveSchema` визначає поля для назви, нотаток, геокоординат та посилання на користувача.

- **Guards, Strategies, Decorators (в основному в AuthModule):** NestJS надає потужні механізми для реалізації автентифікації та авторизації. В `AuthModule` використовуються Passport.js стратегії (`LocalStrategy`, `JwtStrategy`, `GoogleStrategy`), guards (`JwtAuthGuard`, `AdminGuard`) для захисту маршрутів, та кастомні декоратори (наприклад, `@GetUser()`) для зручного доступу до даних користувача в контролерах.

Для взаємодії з базою даних MongoDB [12] використовується ODM Mongoose [10], що дозволяє працювати з даними в об'єктно-орієнтованому стилі. Автоматична генерація документації API за допомогою Swagger (на базі специфікації OpenAPI [16]) значно спрощує тестування та інтеграцію з клієнтською частиною. Використання Fastify [4] як HTTP-адаптера для NestJS було обрано з метою підвищення продуктивності обробки запитів порівняно зі стандартним Express адаптером.

2.3.4 Схема бази даних

Для зберігання даних застосунку *Beekeepers Community Platform* використовується документо-орієнтована NoSQL база даних MongoDB [12], а взаємодія з нею на рівні NestJS-сервісів реалізована за допомогою ODM (Object Document Mapper) Mongoose [10]. Такий підхід забезпечує гнучкість у структуруванні даних та зручні інструменти для їх валідації та маніпуляції. Нижче описано структуру основних колекцій бази даних.

Колекція `users`

Зберігає інформацію про зареєстрованих користувачів платформи. Кожен документ у колекції має наступні ключові поля:

- `email` (String): Електронна пошта користувача, використовується як логін. Поле є обов'язковим та унікальним.

- `password` (String): Хеш паролю користувача. Поле є обов'язковим.
- `username` (String): Ім'я користувача, що відображається на платформі. Поле є обов'язковим.
- `bio` (String, optional): Коротка біографія або опис користувача.
- `location` (String, optional): Місцезнаходження користувача.
- `expertise` (Array of Strings, optional): Список сфер експертизи бджоляра.
- `isEmailVerified` (Boolean): Прапорець, що вказує, чи підтвердив користувач свою електронну пошту. За замовчуванням `false`.
- `emailVerificationToken` (String, optional): Токен для верифікації email. Не вибирається за замовчуванням при запитах.
- `emailVerificationExpires` (Date, optional): Термін дії токена верифікації. Не вибирається за замовчуванням.
- `isAdmin` (Boolean): Прапорець адміністратора. За замовчуванням `false`.
- `createdAt`, `updatedAt` (Date): Часові мітки, що автоматично додаються Mongoose завдяки опції `timestamps: true`.

Колекція `hives`

Призначена для зберігання інформації про вулики, додані користувачами на інтерактивну карту.

- `name` (String): Назва вулика. Поле є обов'язковим.
- `notes` (String, optional): Додаткові нотатки або опис вулика.
- `location` (Object): Геопросторові дані про місцезнаходження вулика. Вбудований об'єкт типу `Point` (GeoJSON), що містить:
  - `type` (String): Тип геометрії, фіксоване значення `'Point'`. Обов'язкове.
  - `coordinates` (Array of Numbers): Масив з двох чисел [довгота, широта]. Обов'язкове.

  Поле `location` індексується за допомогою `2dsphere` індексу для ефективних геопросторових запитів.
- `user` (ObjectId): Ідентифікатор користувача-власника вулика, посилається на колекцію `users`. Поле є обов'язковим та індексованим.
- `createdAt`, `updatedAt` (Date): Автоматичні часові мітки.

Колекція `fields`

Зберігає інформацію про сільськогосподарські поля, які користувачі позначають на карті.

- `name` (String): Назва поля. Поле є обов'язковим.
- `cropType` (String): Тип культури, що вирощується на полі. Поле є обов'язковим.
- `bloomingPeriodStart` (Date): Дата початку періоду цвітіння культури. Поле є обов'язковим
- `bloomingPeriodEnd` (Date): Дата кінця періоду цвітіння культури. Поле є обов'язковим.
- `treatmentDates` (Array of Dates, optional): Список запланованих дат обробки поля. За замовчуванням порожній масив.
- `geometry` (Object): Геометрія поля. Вбудований об'єкт типу `Polygon` (GeoJSON), що містить:
  - `type` (String): Тип геометрії, фіксоване значення `'Polygon'`. Обов'язкове.
  - `coordinates` (Array of Array of Array of Numbers): Масив координат, що визначають полігон (стандарт GeoJSON [[[lng, lat], ...]]). Обов'язкове.

  Поле `geometry` індексується за допомогою `2dsphere` індексу.
- `user` (ObjectId): Ідентифікатор користувача, який додав поле. Посилається на колекцію `users`. Поле є обов'язковим та індексованим.
- `createdAt`, `updatedAt` (Date): Автоматичні часові мітки.

Колекція `forumposts`

Містить пости, створені користувачами на форумі платформи.

- `title` (String): Заголовок поста. Поле є обов'язковим.
- `content` (String): Основний зміст поста. Поле є обов'язковим.
- `author` (ObjectId): Ідентифікатор автора поста, посилається на колекцію `users`. Поле є обов'язковим.
- `likes` (Array of ObjectId, optional): Масив ідентифікаторів користувачів, які вподобали пост. Посилається на колекцію `users`. За замовчуванням порожній масив.
- `comments` (Array of Objects, optional): Масив коментарів до поста. Кожен об'єкт коментаря містить:
  - `content` (String): Текст коментаря. Обов'язкове.

- `author` (ObjectId): Ідентифікатор автора коментаря, посилається на колекцію `users`. Обов'язкове.
- `createdAt` (Date): Дата створення коментаря. За замовчуванням поточна дата.

- `createdAt`, `updatedAt` (Date): Часові мітки для самого поста, автоматично керовані Mongoose (також можуть бути явно визначені в схемі, як вказано).

Така деталізована схема даних забезпечує зберігання всієї необхідної інформації для функціонування платформи та підтримує специфічні вимоги, такі як геопросторові запити та зв'язки між різними сутностями.

2.4 Проектування UI/UX

Проектування користувацького інтерфейсу (UI) та досвіду взаємодії (UX) було спрямоване на створення інтуїтивно зрозумілої, зручної та візуально привабливої платформи. Основним інструментом для реалізації UI стала бібліотека компонентів Material-UI, яка надає широкий набір готових елементів дизайну, що відповідають сучасним стандартам Material Design.

Ключові рішення:

- **Адаптивний дизайн:** Забезпечення коректного відображення та функціонування на різних пристроях (десктопи, планшети, мобільні телефони).
- **Інтуїтивна навігація:** Використання бічної панелі навігації для доступу до основних розділів сайту та чіткої ієрархії сторінок.
- **Консистентність інтерфейсу:** Дотримання єдиного стилю оформлення елементів на всіх сторінках застосунку.
- **Інтерактивність:** Надання користувачам можливості легко взаємодіяти з елементами, такими як карта, форми, кнопки.
- **Зворотний зв'язок:** Інформування користувача про результати його дій (успіх, помилка, процес завантаження) за допомогою повідомлень та індикаторів.

2.4.0.1 Вимоги до форуму

- Реєстрація користувачів з верифікацією електронної пошти.
- Автентифікація (логін/пароль, можливість входу через Google OAuth).
- Зберігання даних користувачів (email, хеш паролю, ім'я користувача, роль, інформація про пасіку за бажанням).
- Управління профілем користувача.
- Відновлення паролю.

2.4.0.2 Вимоги до бази знань

- Створення, редагування та видалення статей/ресурсів адміністраторами.
- Категоризація та тегування матеріалів.
- Пошук по базі знань.
- Система коментарів до статей (за бажанням).
- Рейтинг статей (за бажанням).

2.4.0.3 Загальні нефункціональні вимоги

- **Безпека:** Захист від основних веб-вразливостей (XSS, CSRF, SQL/NoSQL ін'єкції), безпечне зберігання паролів, використання HTTPS.
- **Продуктивність:** Швидке завантаження сторінок та відгук інтерфейсу.
- **Масштабованість:** Архітектура повинна дозволяти додавання нового функціоналу та витримувати зростання кількості користувачів.
- **Надійність:** Система повинна бути доступною та стабільно працювати.
- **Зручність використання (Usability):** Інтуїтивно зрозумілий та легкий у використанні інтерфейс.
- **Адаптивність (Responsiveness):** Коректне відображення на різних пристроях (десктопи, планшети, мобільні телефони).
- **Інтернаціоналізація (i18n):** Підтримка української та англійської мов інтерфейсу.
- **Зворотний зв'язок:** Інформування користувача про результати його дій, помилки.

РОЗДІЛ 3

Реалізація системи

3.1 Огляд процесу розробки

Розробка веб-застосунку *Beekeepers Community Platform* велася з використанням гнучких підходів, що дозволяли ітеративно додавати функціонал та вносити корективи. Основними інструментами розробки були Visual Studio Code як інтегроване середовище розробки, Git для системи контролю версій (з використанням GitHub для хостингу репозиторію), Docker та Docker Compose для контейнеризації та управління середовищем розробки та розгортання. Робочий процес включав регулярні коміти, розробку у функціональних гілках (за потреби) та тестування ключових функцій на локальному середовищі перед розгортанням.

3.2 Організація коду

Структура коду проекту організована для забезпечення модульності, легкості підтримки та масштабування.

3.2.1 Структура проекту фронтенду

Клієнтська частина (директорія `client/`) розроблена на React з використанням Vite як інструменту для збірки. Основні директорії в `src/`:

- `components/`: Містить UI компоненти, розділені за функціональними ознаками (наприклад, `forum/`, `map/`, загальні компоненти).
- `pages/`: Компоненти, що відповідають за окремі сторінки застосунку (наприклад, `Home.tsx`, `Forums.tsx`, `MapPage.tsx`).
- `services/` (або `store/api/`): API-зрізи (slices) для RTK Query, що описують взаємодію з бекенд API (наприклад, `authApi.ts`, `forumApi.ts`, `mapApi.ts`).
- `store/`: Конфігурація Redux store (`store.ts`).
- `context/`: React Context API для управління глобальним станом, таким як автентифікація (`AuthContext.tsx`).
- `hooks/`: Спеціалізовані React хуки.
- `types/`: Загальні TypeScript типи та інтерфейси.

- `locales/`: Файли перекладів для інтернаціоналізації (`en/translation.json`, `uk/transl`
- `utils/`: Допоміжні функції та утиліти.
- `App.tsx`: Головний компонент застосунку, що налаштовує маршрутизацію.
- `index.tsx` (або `main.tsx`): Вхідна точка застосунку.

3.2.2 Структура проекту бекенду

Серверна частина (директорія `server/`) розроблена на NestJS. Проект структурований за модульним принципом:

- `src/`: Містить основний код застосунку.
- `src/MODULE_NAME/`: Кожен функціональний блок (наприклад, `auth/`, `users/`, `forum/`, `hives/`, `fields/`, `health/`, `email/`) виділений в окремий модуль NestJS.
- Кожен модуль зазвичай містить:
  - `*.module.ts`: Файл визначення модуля.
  - `*.controller.ts`: Контролер, що обробляє HTTP-запити.
  - `*.service.ts`: Сервіс, що інкапсулює бізнес-логіку.
  - `schemas/`: Mongoose схеми для визначення структури даних MongoDB.
  - `dto/`: Data Transfer Objects для валідації вхідних даних.
  - `guards/`, `strategies/`, `decorators/`, `types/` (для специфічних типів модуля, наприклад, в `auth/`).
- `src/main.ts`: Вхідна точка серверного застосунку, ініціалізація NestJS та Fastify.
- `src/app.module.ts`: Кореневий модуль застосунку.

3.2.3 Проектування API

API спроектовано за принципами RESTful. Всі ендпоінти мають префікс `/api/v1/`, що забезпечує версіонування. Для валідації даних, що надходять від клієнта, використовуються Data Transfer Objects (DTO) з декораторами `class-validator`. Документація API автоматично генерується за допомогою Swagger (OpenAPI) і доступна на ендпоінті `/docs` (у середовищі розробки), що спрощує тестування та інтеграцію.

3.3 Опис ключових компонентів та модулів

3.3.1 Автентифікація та авторизація

Система автентифікації реалізована з використанням Passport.js. Підтримуються:

- **Локальна стратегія:** Автентифікація за логіном (email) та паролем. Паролі зберігаються у хешованому вигляді (з використанням `crypto.pbkdf2Sync`).
- **JWT (JSON Web Tokens):** Використовуються для авторизації користувачів після успішного входу. Генеруються access та refresh токени.
- **Верифікація Email:** При реєстрації користувачу надсилається лист з унікальним токеном для підтвердження електронної пошти. Доступ до функціоналу обмежений до верифікації. Термін дії токена – 1 година.
- **Повторне надсилання листа верифікації:** Реалізовано можливість для користувача запросити повторне надсилання листа для верифікації email, якщо попередній токен сплив або лист не було отримано. Цей функціонал включає новий ендпоінт API (`/auth/resend-verification-email`) та відповідний інтерфейс на сторінці верифікації (`VerifyEmailPage.tsx`).
- **Google OAuth 2.0:** Реалізована можливість входу та реєстрації через обліковий запис Google.
- **Захист маршрутів (Guards):** `JwtAuthGuard` використовується для захисту ендпоінтів, що вимагають автентифікації. `LocalAuthGuard` використовується для обробки локального входу.

3.3.2 Форум

Модуль форуму дозволяє користувачам створювати теми для обговорення, залишати повідомлення та коментарі, а також висловлювати свою думку за допомогою лайків. Дані зберігаються в колекції `forumposts`. Фронтенд реалізований з використанням компонентів для списку постів, окремого поста та форми створення нового поста. Взаємодія з бекендом відбувається через відповідні API ендпоінти, реалізовані в `ForumController` та `ForumService`.

3.3.3 База знань

Модуль бази знань (на даний момент з демонстраційними даними на клієнті) передбачає зберігання та відображення статей, посібників та інших корисних матеріалів для бджолярів. Планується розширення функціоналу для управління контентом через адміністративну панель та реалізація повноцінного API для роботи з ресурсами бази знань.

3.3.4 Інтеграція інтерактивної карти

Одним з ключових функціональних блоків є інтерактивна карта, реалізована за допомогою Leaflet та React-Leaflet на фронтенді та GeoJSON з індексами `2dsphere` на бекенді (MongoDB). Компонент `MapPage.tsx` є центральним для цієї функціональності.

3.3.4.1 Управління вуликами (Hives)

- **Додавання вуликів:** Користувачі можуть додавати на карту точкові маркери для позначення вуликів. Через діалогове вікно `AddHiveDialog.tsx` вказуються назва та нотатки. Геолокація визначається кліком на карті. Дані надсилаються на бекенд за допомогою RTK Query мутації `useAddHiveMutation`.

- **Відображення вуликів з кастомними іконками:** Стандартні маркери бібліотеки Leaflet є досить загальними. Для покращення візуального сприйняття та тематичної відповідності було реалізовано відображення вуликів на карті за допомогою кастомних іконок. В якості візуального образу було обрано іконку `HiveIcon` з бібліотеки Material-UI, що забезпечує стилістичну єдність з іншими елементами інтерфейсу платформи.

  Оскільки Leaflet очікує HTML-рядок або DOM-елемент для кастомних маркерів через `L.divIcon`, а `HiveIcon` є React-компонентом, було застосовано наступне рішення. За допомогою функції `ReactDOMServer.renderToString()` з пакету `react-dom/serv` (яка зазвичай використовується для серверного рендерингу) React-компонент іконки перетворюється на статичний HTML-рядок на стороні клієнта. Цей рядок потім передається у властивість `html` об'єкта конфігурації `L.divIcon`.

  При створенні кастомної іконки також налаштовуються важливі параметри `L.divIcon`, такі як `className` (для можливості додаткової CSS-стилізації), `iconSize` (розмір іконки), `iconAnchor` (точка прив'язки іконки до географічних координат, щоб вістря іконки вказувало на точне місце), та `popupAnchor` (позиціонування спливаючого вікна відносно іконки). Сформований таким чином об'єкт `hiveLeafletIcon` потім передається у пропс `icon` компонента `<Marker>` з бібліотеки React-Leaflet. Такий підхід дозволив ефективно інтегрувати React-компоненти Material-UI в картографічний контекст Leaflet, забезпечуючи кращий користувацький досвід та візуальну привабливість карти.

- **Видалення вуликів:** Користувачі можуть видаляти свої вулики. У спливаючому вікні (Popup) маркера вулика розміщена кнопка видалення. Натискання на неї активує діалогове вікно для підтвердження дії. Після підтвердження, мутація `useDeleteHiveMutation` з RTK Query відправляє запит на бекенд для видалення відповідного запису з колекції `hives`.

- **Перегляд інформації:** У спливаючому вікні (Popup) кожного вулика відображається його назва та нотатки.

3.3.4.2 Управління полями (Fields)

- **Додавання полів:** Користувачі можуть малювати полігони на карті для позначення полів. Діалогове вікно `AddFieldDialog.tsx` дозволяє вказати назву поля, тип культури, період цвітіння та список запланованих дат обробки. Геометрія полігону передається на бекенд у форматі GeoJSON. Використовується мутація `useAddFieldMutation`.

- **Відображення полів та візуалізація статусу обробки:** Поля відображаються як полігони на карті за допомогою компонента `<Polygon>` з бібліотеки React-Leaflet. Ключовою особливістю є динамічна зміна кольору полігону для візуального інформування бджолярів про заплановані хімічні обробки полів, що можуть становити загрозу для бджіл. Ця логіка реалізована в компоненті `MapPage.tsx` у функції `getFieldTreatmentSta`

  Функція приймає масив дат обробок для конкретного поля. Вона порівнює кожну дату обробки з поточною датою. Якщо обробка запланована на сьогодні, полю присвоюється червоний колір (високий ризик). Якщо обробка запланована протягом наступних семи днів (визначено константою `TREATMENT_SOON_DAYS`), полю присвоюється помаранчевий колір (середній ризик). В інших випадках використовується стандартний синій колір. Перевірка на "сьогодні"здійснюється за допомогою допоміжної функції `isSameDay`, яка порівнює рік, місяць та день двох дат. Результатом роботи `getFieldTreatmentStatus` є об'єкт, що містить властивість `color`, значення якої динамічно передається в атрибут `pathOptions` компонента `<Polygon>`. Для оптимізації та уникнення зайвих перерахунків при кожному рендері компонента, функція `getFieldTreatmentStatus` обгорнута в React хук `useMemo` (хоча в поточній реалізації без залежностей вона створюється один раз). Це забезпечує користувачам наочне та оперативне сповіщення про потенційні загрози.

- **Редагування метаданих полів:** Для модифікації атрибутів існуючих полів було розроблено спеціалізований діалоговий компонент `EditFieldDialog.tsx`. Активація цього діалогу відбувається при натисканні користувачем кнопки редагування у спливаючому вікні (Popup) відповідного полігону поля на карті. Компонент отримує через пропси поточні дані поля (`initialData`), прапорець видимості (`open`), обробники закриття (`onClose`) та відправки форми (`onSubmit`), а також стан завантаження (`isLoading`) для індикації процесу взаємодії з API.

  Внутрішній стан форми (`formData`), що включає назву, тип культури, дати початку та кінця цвітіння, а також список дат обробки, управляється за допомогою хука `useState`. При відкритті діалогу або зміні `initialData`, хук `useEffect` відповідає за ініціалізацію стану форми даними обраного поля. Важливим аспектом є перетворення форматів дат: рядкові ISO-дати, отримані з бекенду, форматуються у вигляд `YYYY-MM-DD`, сумісний з HTML-елементами вводу типу `date`.

  Обробка змін у текстових полях форми реалізована через універсальний обробник `handleChange`. Для управління динамічним списком дат обробки передбачені окремі функції: `addTreatmentDate` для додавання нового поля вводу дати, `removeTreatmentD` для видалення існуючого, та `handleTreatmentDateChange` для оновлення конкретної дати у списку.

  При відправці форми функція `handleSubmit` виконує базову валідацію введених даних. Якщо валідація успішна, викликається переданий через пропси обробник `onSubmit` (визначений у `MapPage.tsx`). Цей обробник, у свою чергу, активує RTK Query му-

тацію `useUpdateFieldMutation`, передаючи ідентифікатор поля (`_id`) та об'єкт `formData` з оновленими даними на сервер. Протягом виконання асинхронного запиту до API, пропс `isLoading` використовується для блокування елементів форми та відображення індикатора завантаження на кнопці збереження, забезпечуючи користувачеві зворотний зв'язок.

- **Перегляд інформації:** У спливаючому вікні (Popup) кожного поля відображається його назва, тип культури, період цвітіння та список дат обробки.

Бекенд надає відповідні CRUD API ендпоінти для управління вуликами та полями, забезпечуючи збереження, отримання, оновлення та видалення геопросторових об'єктів та їх метаданих (колекції `hives` та `fields` у MongoDB).

3.3.5 Реалізація функціоналу адміністрування користувачів

Для забезпечення можливостей управління користувачами платформи було розроблено адміністративний функціонал, доступний лише користувачам з роллю адміністратора. Цей функціонал включає перегляд списку всіх користувачів та можливість зміни їх адміністративного статусу.

3.3.5.1 Захист адміністративних маршрутів

На серверній стороні доступ до адміністративних ендпоінтів контролюється за допомогою спеціалізованого NestJS Guard – `AdminGuard` (файл `server/src/auth/guards/admin.`
Цей guard активується після успішної автентифікації користувача через `JwtAuthGuard`. `AdminGuard` перевіряє наявність поля `isAdmin` зі значенням `true` в об'єкті користувача, що додається до запиту `JwtAuthGuard`. Якщо користувач не є адміністратором, guard генерує виняток `ForbiddenException`, блокуючи доступ.

3.3.5.2 Сервісні методи та контролери для управління користувачами

У сервісі `UsersService` (`server/src/users/users.service.ts`) було додано два ключових методи для адміністративних потреб:

- `findAllAdmin()`: Повертає повний список всіх документів користувачів з бази даних. На відміну від потенційних публічних методів отримання користувачів, цей метод призначений для надання адміністратору повного огляду.
- `updateUserAdminStatus(userIdToUpdate: string, isAdmin: boolean)`: Оновлює поле `isAdmin` для вказаного користувача. Метод знаходить користувача за `userIdToUpdate` та встановлює нове значення для його адміністративного статусу, повертаючи оновлений документ користувача.

Відповідні ендпоінти були додані до `UsersController` (`server/src/users/users.control`

- `GET /users/admin/all`: Викликає `usersService.findAllAdmin()` та захищений `JwtAuthGuard` і `AdminGuard`.

- `PATCH /users/admin/:userId/set-admin-status`: Приймає `userId` як параметр маршруту та тіло запиту типу `SetUserAdminStatusDto` (що містить поле `isAdmin: boolean`). Викликає `usersService.updateUserAdminStatus()` та також захищений обома guards.

Для валідації тіла запиту при зміні статусу адміністратора використовується DTO `SetUserAdminSt` з відповідними декораторами `class-validator`.

3.3.5.3 Клієнтська реалізація панелі адміністратора

На клієнтській стороні було створено сторінку `AdminPage.tsx` (розташовану в `client/sr` яка слугує інтерфейсом для управління користувачами. Доступ до цієї сторінки контролюється на рівні компонента: перевіряється, чи поточний автентифікований користувач (отриманий за допомогою хука `useAuth` з `AuthContext`) має прапорець `isAdmin`, встановлений в `true`. Якщо користувач не є адміністратором, або дані про користувача ще завантажуються, відображається відповідне повідомлення про відмову в доступі або індикатор завантаження.

Для отримання списку всіх користувачів сторінка використовує RTK Query хук `useGetAllUs` з `usersApi.ts`. Запит пропускається (опція `skip`), якщо поточний користувач не визначений як адміністратор. Отриманий список користувачів відображається за допомогою компонентів Material-UI `<List>`, `<ListItem>` та `<ListItemText>`, показуючи ім'я користувача, його email та ID. Поточний залогінений адміністратор візуально виділяється у списку.

Зміна адміністративного статусу користувача реалізована за допомогою компонента `<Switch>` (Material-UI) для кожного користувача у списку. Обробник `handleAdminStatusChang` викликається при зміні стану перемикача. Цей обробник ініціює RTK Query мутацію `useUpdateUse` передаючи ID користувача та нове значення статусу `isAdmin`. Для запобігання випадковій зміні власних прав, перемикач для поточного адміністратора деактивується. Після успішного виконання мутації (оновлення даних на сервері), викликається функція `refetch` (надана хуком `useGetAllUsersAdminQuery`) для оновлення списку користувачів на сторінці, що забезпечує актуальність відображених даних. Стани завантаження під час отримання списку (`isLoadingUsers`) та оновлення статусу (`isUpdatingStatus`) використовуються для відображення індикатора `<CircularProgress>`, надаючи користувачеві зворотний зв'язок про хід операцій. Текстові елементи на сторінці інтернаціоналізовані за допомогою хука `useTranslation` з `react-i18next`.

3.3.6 Налаштування доставки транзакційних електронних листів

Для забезпечення надійної та безпечної доставки транзакційних електронних листів (наприклад, для верифікації email), платформа *Beekeepers Community Platform* інтегрується з сервісом Mailgun. Правильне налаштування DNS є критично важливим для доставки та відповідності сучасним галузевим стандартам, особливо тим, що висуваються великими провайдерами, такими як Google та Yahoo.

3.3.6.1 Автентифікація домену

Для автентифікації вихідних електронних листів до DNS-конфігурації домену проекту були додані наступні записи:

- **SPF (Sender Policy Framework):** TXT-запис, що вказує, які поштові сервери мають право надсилати електронні листи від імені домену. Це допомагає запобігти несанкціонованим відправникам (спуфінгу).
- **DKIM (DomainKeys Identified Mail):** TXT-запис, що містить публічний криптографічний ключ. Вихідні листи підписуються приватним ключем Mailgun, а одержувачі можуть перевірити підпис за допомогою публічного ключа в DNS, забезпечуючи цілісність та автентичність повідомлення.
- **DMARC (Domain-based Message Authentication, Reporting, and Conformance):** TXT-запис, який інструктує приймаючі поштові сервери, як обробляти листи, що не пройшли перевірки SPF або DKIM.
  - **Опції політики:**
    - `p=none`: Тільки моніторинг, без примусових дій.
    - `p=quarantine`: Відправляти листи, що не пройшли перевірку, до папки "Спам".
    - `p=reject`: Повністю блокувати листи, що не пройшли перевірку.
  - **Сучасна практика:** Через нові вимоги Google та Yahoo (з 2024 року), політика DMARC повинна бути встановлена щонайменше на `quarantine` або `reject` для використання у виробничому середовищі. Це захищає користувачів від фішингу та покращує доставку.

3.3.6.2 Поширення та верифікація DNS

Після додавання необхідних записів, поширення змін у DNS може зайняти до 48 годин. Верифікація виконується за допомогою панелі керування Mailgun та сторонніх інструментів (наприклад, MXToolbox), щоб переконатися, що всі записи коректно опубліковані та доступні глобально.

3.3.6.3 Безпека та відповідність стандартам

- **Відсутність жорстко закодованих облікових даних:** Усі API-ключі Mailgun та конфіденційні налаштування зберігаються у змінних середовища, а не в коді.
- **Постійний моніторинг:** Звіти DMARC надсилаються на визначену електронну адресу для моніторингу проблем автентифікації та потенційного зловживання.
- **Ескалація політики:** Проект спочатку використовує `p=none` для DMARC для моніторингу легітимного трафіку, а потім переходить до `quarantine` або `reject` для повної відповідності стандартам та захисту.

3.3.6.4 Підсумкова таблиця DNS-записів

Запис	Призначення	Приклад значення / Політика
SPF	Автентифікація відправника	`v=spf1 include:mailgun.org ~all`
DKIM	Підпис повідомлення	(Публічний ключ, наданий Mailgun)
DMARC	Політика та звіти	`v=DMARC1; p=quarantine; rua=mailto:admin@`

Таблиця 3.1. Приклади DNS-записів для автентифікації email.

Така конфігурація гарантує, що всі вихідні електронні листи автентифіковані, знижуючи ризик спаму та фішингу та відповідаючи останнім вимогам основних поштових провайдерів. Вона також підтримує моніторинг та постійне вдосконалення безпеки електронної пошти.

3.3.7 Реалізація інтелектуального FAQ-асистента

Для надання користувачам швидких відповідей на поширені питання було реалізовано прототип інтелектуального FAQ-асистента, що використовує можливості великих мовних моделей (LLM), зокрема GPT-3.5-turbo від OpenAI [15]. Метою було дослідження потенціалу застосування LLM для покращення користувацького досвіду та оперативності надання інформації в рамках платформи.

3.3.7.1 Серверна частина (Backend): Промпт-інжиніринг та взаємодія з OpenAI

На бекенді, в модулі `FaqModule`, ключову роль відіграє `FaqService`. Цей сервіс відповідає за підготовку даних, конструювання промпту та взаємодію з API OpenAI.

- **DTO запиту та Контролер:** Як і в інших модулях, використовується DTO `AskFaqDto` для валідації вхідного питання від користувача, яке надходить на ендпоінт `POST /api/v1/fa` оброблюваний `FaqController`.

- **Управління API-ключем:** API-ключ для доступу до OpenAI завантажується з конфігураційних файлів (змінних середовища) через `ConfigService`. Реалізовано перевірку наявності ключа при ініціалізації сервісу; у разі його відсутності, функціонал FAQ стає недоступним, про що логується попередження та інформується користувач при спробі запиту.

- **Побудова промпту – ядро логіки сервісу:** Якість відповіді LLM значною мірою залежить від якості та структури наданого промпту. У поточній реалізації прототипу застосовано наступний підхід до промпт-інжинірингу в методі `answerQuestion`:

    1. **Формування контексту з ЧаПи (FAQ):** На даному етапі використовується статичний масив `FAQ_DATA`, що містить пари «питання-відповідь». Перед відправкою до LLM, цей масив перетворюється на текстовий блок, де кожне питання та відповідь чітко позначені (наприклад, Q1:/A1:, Q2:/A2:). Це допомагає моделі

розрізняти окремі елементи ЧаПи. *Обмеження поточного підходу до контексту:* Основне обмеження полягає в тому, що весь обсяг `FAQ_DATA` включається в кожен запит. Зі зростанням кількості ЧаПи, це призведе до перевищення ліміту токенів, який підтримує обрана модель (наприклад, gpt-3.5-turbo має ліміт близько 4096 токенів для промпту та відповіді разом). Це робить поточне рішення не масштабованим для великих баз знань і слугує обґрунтуванням для впровадження семантичного пошуку в майбутньому.

2. **Розробка системного повідомлення (System Prompt):** Системне повідомлення є критично важливим для керування поведінкою LLM. Для даного асистента воно було сформульовано таким чином, щоб чітко визначити роль моделі, обмежити джерело її знань та задати стиль відповіді:

   > *Ви є корисним асистентом для платформи "Beekeepers Community Platform". Ваше завдання - відповідати на питання користувачів, базуючись ВИКЛЮЧНО на наданому контексті з ЧаПи (Часто Задаваних Питань). Якщо відповіді немає в контексті, чітко вкажіть, що ви не можете надати відповідь на основі наявної інформації. Не вигадуйте відповіді. Будьте коротким та чітким. Відповідайте українською мовою.*

   Ключові елементи цього промпту – вимога базуватися **виключно** на наданому контексті та інструкція щодо поведінки у випадку відсутності інформації – спрямовані на мінімізацію «галюцинацій» моделі (генерування неправдивої або нерелевантної інформації) та забезпечення того, що користувачі отримують відповіді, що стосуються саме платформи.

3. **Конструювання фінального запиту до моделі:** Питання користувача, попередньо підготовлений контекст ЧаПи та системне повідомлення об'єднуються в структурований запит (у форматі повідомлень для Chat Completions API OpenAI), який надсилається до моделі. Це забезпечує чітке розділення інструкцій, контекстуальних даних та запиту користувача.

- **Взаємодія з OpenAI API та параметризація:** Для взаємодії з API використовується офіційна бібліотека `openai` для Node.js. Було обрано модель `gpt-3.5-turbo` як збалансований варіант за співвідношенням можливостей та вартості. Ключові параметри запиту до API включають:
  - `model`: Назва використовуваної моделі.
  - `messages`: Масив об'єктів, що включає системне повідомлення та повідомлення користувача (з контекстом та питанням).
  - `temperature`: Встановлено на низьке значення (наприклад, 0.2 з діапазону 0 до 2) для зменшення випадковості та отримання більш детермінованих, фактологічних відповідей, що є бажаним для FAQ-системи.

  - `max_tokens`: Обмежує максимальну довжину генерованої відповіді (наприклад, 150-250 токенів), що допомагає контролювати витрати та стисліть відповіді.

- **Обробка відповіді та помилок:** Отримана від OpenAI відповідь проходить базову обробку (наприклад, видалення зайвих пробілів) перед поверненням на клієнт. Реалізовано логування запитів та відповідей, а також обробку потенційних помилок під час виклику API, з поверненням відповідного повідомлення користувачу.

На момент підготовки тез, для контрольованого запуску та уникнення непередбачених витрат, `FaqModule` закоментовано в основному модулі серверного застосунку (`AppModule`), що робить ендпоінт `/api/v1/faq/ask` тимчасово недоступним.

3.3.7.2 Клієнтська частина (Frontend)

На фронтенді реалізовано компонент `FaqSection.tsx` (у `client/src/components/fa` який надає користувацький інтерфейс для взаємодії з FAQ-асистентом.

- **Інтерфейс користувача:** Компонент містить текстове поле (MUI `<TextField>`) для введення питання та кнопку (MUI `<Button>`) для відправки запиту. Відповідь від асистента відображається в окремому блоці.

- **Взаємодія з API:** Для надсилання запиту на бекенд та отримання відповіді створено окремий RTK Query API slice `faqApi.ts` (у `client/src/store/api/`). Він визначає мутацію `useAskFaqMutation`.

- **Управління станом:** Компонент `FaqSection` використовує хук `useState` для зберігання поточного питання та отриманої відповіді. Стан завантаження (`isLoading`) та можливі помилки (`error`), що повертаються хуком `useAskFaqMutation`, використовуються для відображення індикатора завантаження (MUI `<CircularProgress>`) та повідомлень про помилки.

- **Інтеграція:** Компонент `FaqSection` інтегровано на сторінку Бази Знань (`KnowledgeBase.` хоча його рендеринг може бути керований feature-прапорцем для поступового впровадження.

3.4 Безпека системи

При розробці увага приділялася аспектам безпеки:

- **Автентифікація та авторизація:** Використання JWT, захист маршрутів, OAuth 2.0.

- **Зберігання паролів:** Паролі користувачів хешуються на стороні сервера перед збереженням у базу даних (використано модуль `crypto` Node.js, зокрема `pbkdf2Sync`).

- **Валідація вхідних даних:** Усі дані, що надходять від клієнта на бекенд, валідуються за допомогою DTO та `class-validator`, що запобігає некоректним даним та потенційним атакам (наприклад, NoSQL ін'єкції на рівні структури даних).

- **Захист від XSS:** Використання React на фронтенді за замовчуванням екранує дані, що вставляються в DOM, що знижує ризик XSS-атак.
- **CORS:** Налаштовано політику Cross-Origin Resource Sharing для контролю доступу до API з боку фронтенду.
- **Використання HTTPS:** У продакшн-середовищі необхідно використовувати HTTPS для шифрування трафіку.
- **Управління секретами:** Чутливі дані (секрети JWT, ключі API для зовнішніх сервісів, рядок підключення до БД) зберігаються у змінних середовища та не включаються до системи контролю версій.

РОЗДІЛ 4

Тестування та розгортання

4.1 Тестування системи

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, спрямованою на виявлення помилок та забезпечення відповідності функціональним та нефункціональним вимогам. Для веб-застосунку *Beekeepers Community Platform* було проведено кілька видів тестування.

4.1.1 Модульне тестування (Unit Testing)

Модульне тестування було зосереджено на перевірці окремих компонентів та функцій серверної частини (NestJS [14]). Використовувався вбудований в NestJS тестовий фреймворк, що базується на Jest [3]. Тестувалися сервіси, контролери (частково) та допоміжні утиліти. Основна увага приділялася тестуванню бізнес-логіки в сервісах, валідації даних та коректності відповідей API.

4.1.2 Інтеграційне тестування

Інтеграційне тестування передбачало перевірку взаємодії між різними модулями системи, зокрема між фронтендом та бекендом (API ендпоінти), а також взаємодію бекенду з базою даних MongoDB. На цьому етапі перевірялася коректність обробки запитів, передачі даних та їх збереження/отримання.

4.1.3 Тестування користувацького інтерфейсу (UI Testing)

На фронтенді проводилося ручне тестування користувацького інтерфейсу на різних пристроях та в різних браузерах (Chrome, Firefox, Safari) для забезпечення адаптивності та коректного відображення. Перевірялася робота інтерактивних елементів, форм, навігації та картографічного функціоналу.

4.1.4 Тестування безпеки

Здійснювалася базова перевірка на поширені веб-вразливості, такі як XSS (здебільшого покривається React), валідація вхідних даних для запобігання ін'єкціям на рівні API. Також перевірялася робота системи автентифікації та авторизації, зокрема захист маршрутів та валідність JWT токенів.

4.2 Розгортання застосунку

Для розгортання веб-застосунку *Beekeepers Community Platform* було обрано платформу як сервіс (PaaS) Render [19], що дозволило автоматизувати та спростити процес виведення застосунку в продуктивне середовище. Контейнеризація за допомогою Docker [2] та Docker Compose, описана на етапі локальної розробки, лягла в основу конфігурації сервісів на Render.

4.2.1 Конфігурація сервісів на Render

Платформа була розділена на два основні сервіси, розгорнуті на Render:

- **Клієнтська частина (Frontend):** Розгорнута як ”Static Site”. Render було підключено до GitHub репозиторію проекту. При кожному оновленні основної гілки (наприклад, `main` або `master`) Render автоматично ініціював процес збірки, виконуючи команду `npm run build` (визначену у `package.json` Vite-проекту). Зібрані статичні активи з директорії `client/dist` публікувалися та ставали доступними через наданий Render домен. Налаштування Render для статичних сайтів також дозволило легко конфігурувати правила перенаправлення для коректної роботи односторінкового застосунку (SPA) з React Router.

- **Серверна частина (Backend):** Розгорнута як ”Web Service”з використанням Docker-контейнера. Файл `Dockerfile`, що знаходився в директорії `server/`, використовувався Render для побудови образу. Цей Dockerfile був оптимізований для продуктивного середовища, потенційно включаючи багатоетапну збірку для зменшення розміру кінцевого образу. Командою запуску сервісу була вказана `npm run start:prod`. Render автоматично обробляв мапінг портів, роблячи внутрішній порт NestJS-застосунку (наприклад, 4000) доступним для зовнішнього трафіку.

- **База даних:** Для зберігання даних використовувалася хмарна служба MongoDB Atlas [12]. Було створено безкоштовний кластер, отримано рядок підключення (SRV-запис), який потім був безпечно доданий як змінна середовища як `DATABASE_URL` в налаштуваннях бекенд-сервісу на Render. Налаштування мережевого доступу в MongoDB Atlas було сконфігуровано для дозволу з’єднань з IP-адрес сервісів Render, або використовувалися загальнодоступні IP (0.0.0.0/0) на час розробки з подальшим посиленням безпеки.

4.2.2 Управління конфігурацією, безпека та CI/CD

Ключові аспекти конфігурації, безпеки та автоматизації при розгортанні на Render включали:

- **Змінні середовища:** Усі чутливі дані – секрети JWT, ключі API для сервісу Mailgun, URL бази даних MongoDB Atlas, порт сервера, тощо – були конфігуровані виключно

через змінні середовища в панелі керування Render для кожного сервісу. Це забезпечує надійний захист конфігураційної інформації та унеможливлює її потрапляння до системи контролю версій.

- **HTTPS:** Render автоматично надає та управляє SSL-сертифікатами (через Let's Encrypt) для всіх веб-сервісів та статичних сайтів, розгорнутих на платформі. Це забезпечило шифрування всього трафіку між клієнтами та серверами за протоколом HTTPS без необхідності ручного налаштування сертифікатів.
- **Автоматичне розгортання (CI/CD з Render):** Інтеграція Render з GitHub репозиторієм забезпечила базовий, але ефективний процес безперервної інтеграції та доставки. Кожен push або merge в основну гілку (наприклад, `main` або `master`) автоматично ініціював нову збірку та розгортання відповідного сервісу (фронтенд чи бекенд) на Render, що значно прискорило ітераційний процес розробки та оновлення застосунку.
- **Автоматизовані перевірки якості коду (GitHub Actions):** Для забезпечення високої якості коду та раннього виявлення потенційних проблем було налаштовано робочий процес GitHub Actions (файл `.github/workflows/lint-pr.yml`). Цей процес автоматично запускається при створенні або оновленні кожного запиту на злиття (Pull Request) до основних гілок репозиторію (наприклад, `master`). Робочий процес включає наступні кроки: вивантаження коду, налаштування середовища Node.js (версія 22.x), встановлення залежностей для клієнтської та серверної частин за допомогою `npm ci`, та запуск команди `make lint_ci`. Команда `make lint_ci`, визначена у файлі `Makefile`, послідовно виконує скрипти лінтингу (ESLint з відповідними конфігураціями) для обох частин проекту, причому для серверної частини використовується прапорець `--max-warnings 0` для забезпечення неуспішного завершення при наявності будь-яких попереджень. Успішне проходження цього CI-завдання є однією з умов для злиття змін, що сприяє підтримці чистоти та консистентності кодової бази.
- **Моніторинг та Логування:** Render надає вбудовані інструменти для перегляду логів розгорнутих сервісів в реальному часі, що використовувалося для моніторингу стану застосунку та діагностики можливих проблем під час роботи.

Використання PaaS-платформи Render значно спростило інфраструктурні аспекти розгортання, дозволивши зосередитися на розробці самого застосунку. Перевагами такого підходу стали легкість налаштування, глибока інтеграція з Git, автоматичне масштабування (в межах можливостей обраного тарифного плану Render), забезпечення безпечного HTTPS-з'єднання, та зручне управління змінними середовища. Це дозволило швидко отримати робочий прототип, доступний онлайн.

4.3 Майбутні напрямки розвитку

Платформа *Beekeepers Community Platform* має потенціал для подальшого розвитку та розширення функціоналу. Можливі напрямки включають:

- Розширення функціоналу бази знань: додавання можливості користувачам пропонувати статті, система рецензування, коментарі.
- Розвиток картографічного сервісу: фільтри за типами культур, періодами цвітіння, сповіщення про обробку полів, інтеграція з погодними даними.
- Система приватних повідомлень між користувачами.
- Календар подій для бджолярів (виставки, ярмарки, семінари).
- **Вдосконалення та повна інтеграція інтелектуального FAQ-асистента:**
  - Заміна статичного масиву ЧаПи (`FAQ_DATA`) на динамічне завантаження даних з модуля "База знань"або окремої колекції в MongoDB, що дозволить адміністраторам легко оновлювати та розширювати контент для асистента.
  - Впровадження механізму семантичного пошуку (наприклад, з використанням векторних вкладень тексту та векторної бази даних) для відбору найбільш релевантних фрагментів з бази знань для формування контексту для LLM. Це дозволить значно підвищити точність відповідей та подолати обмеження на максимальну довжину промпту при великих обсягах інформації.
  - Додавання можливості для користувачів оцінювати корисність відповідей асистента (наприклад, лайк/дизлайк) для збору зворотного зв'язку та подальшого покращення якості.
  - Розгляд можливості тонкого налаштування (fine-tuning) меншої, можливо відкритої, мовної моделі на специфічних даних платформи для оптимізації витрат та підвищення контролю над відповідями.
  - Інтеграція механізму кешування для часто задаваних питань для зменшення кількості запитів до OpenAI API та прискорення отримання відповідей.
  - Моніторинг використання та аналіз запитів до FAQ-асистента для виявлення прогалин у базі знань та оптимізації роботи сервісу.
- Мобільний застосунок (React Native або нативні технології).
- Розширена аналітика та статистика для користувачів (наприклад, продуктивність пасік).
- Інтеграція з іншими сервісами (наприклад, маркетплейси для продукції бджільництва).

Висновки

У даній магістерській роботі було розроблено повностековий веб-додаток *Beekeepers Community Platform*. Метою роботи було створення платформи для спілкування, обміну досвідом та знаннями серед бджолярів, а також надання інструментів для управління пасіками та полями.

Основні досягнуті результати:

- Проведено аналіз предметної області та обґрунтовано вибір сучасного стеку технологій (React, NestJS, MongoDB).
- Спроектовано та реалізовано ключові функціональні модулі: система автентифікації (включаючи email верифікацію та Google OAuth), форум, база знань, інтерактивна карта.
- Забезпечено базові механізми безпеки та валідації даних.
- Створено адаптивний користувацький інтерфейс з використанням Material-UI.

Розроблений додаток успішно вирішує поставлені завдання, надаючи зручну та функціональну платформу для спільноти бджолярів.

Напрямки для подальшого розвитку включають розширення функціоналу карти (фільтрація, аналітика), впровадження системи сповіщень, розробку мобільного додатку та інтеграцію з іншими сервісами для бджолярів.

Список використаних джерел

[1] Volodymyr Agafonkin and contributors. Leaflet - an open-source javascript library for mobile-friendly interactive maps. https://leafletjs.com/, 2024. Accessed: 2025-05-12.

[2] Docker, Inc. Docker - empowering app development for developers and teams. https://www.docker.com/, 2024. Accessed: 2025-05-12.

[3] Facebook, Inc. and contributors. Jest - delightful javascript testing. https://jestjs.io, 2024. Accessed: 2025-05-15.

[4] Fastify Maintainers. Fastify - fast and low overhead web framework, for node.js. https://www.fastify.io/, 2024. Accessed: 2025-05-12.

[5] Carlos Granell, Sven Casteleyn, and Clement Atzberger. Editorial: Geospatial Data Capturing, Processing, Analysis, and Visualization in Agro-Geoinformatics. *Frontiers in Environmental Science*, 3(78), 2015. Accessed: 2025-05-15.

[6] Shreyas M. Guruprasad and Benjamin Leiding. BeeOpen—An Open Data Sharing Ecosystem for Apiculture. *Agriculture*, 14(3):470, 2024. Accessed: 2025-05-15.

[7] Jared Hanson and Passport contributors. Passport.js - simple, unobtrusive authentication for node.js. https://www.passportjs.org/, 2024. Accessed: 2025-05-12.

[8] Jean-Christophe Huet, Lamine Bougueroua, Yassine Kriouile, Katarzyna Wegrzyn-Wolska, and C'edric Ancourt. Digital Transformation of Beekeeping through the Use of a Decision Making Architecture. *Applied Sciences*, 12(21):11179, 2022. Accessed: 2025-05-15.

[9] i18next. i18next - internationalization-framework written in and for javascript. https://www.i18next.com/, 2024. Accessed: 2025-05-12.

[10] Valeri Karpov and contributors. Mongoose odm - elegant mongodb object modeling for node.js. https://mongoosejs.com/, 2024. Accessed: 2025-05-12.

[11] Meta Platforms, Inc. React - a javascript library for building user interfaces. https://react.dev/, 2024. Accessed: 2025-05-12.

[12] MongoDB, Inc. Mongodb atlas - the multi-cloud developer data platform. https://www.mongodb.com/, 2024. Accessed: 2025-05-12.

[13] MUI. Mui - the react component library you always wanted. https://mui.com/, 2024. Accessed: 2025-05-12.

[14] NestJS. Nestjs - a progressive node.js framework. https://nestjs.com/, 2024. Accessed: 2025-05-12.

[15] OpenAI. OpenAI API Platform. https://platform.openai.com/docs/overview, 2024. Accessed: 2025-05-15.

[16] OpenAPI Initiative. Openapi specification - the industry standard for rest apis. https://www.openapis.org/, 2024. Accessed: 2025-05-12.

[17] Jenny Preece and Diane Maloney-Krichmar. Online communities: Design, theory, and practice. *Journal of Computer-Mediated Communication*, 10(4), 2005. Accessed: 2025-05-15.

[18] Redux Maintainers. Redux toolkit - the official, opinionated, batteries-included toolset for efficient redux development. https://redux-toolkit.js.org/, 2024. Accessed: 2025-05-12.

[19] Render Inc. Render - the easiest way to build and run all your apps and websites. https://render.com/, 2024. Accessed: 2025-05-15.

[20] Kibebew Wakjira, Taye Negera, Aleksejs Zacepins, Armands Kviesis, Vitalijs Komasilovs, Sascha Fiedler, Sascha Kirchner, Oliver Hensel, Dwi Purnomo, Marlis Nawawi, Amanda Paramita, Okie Fauzi Rachman, Aditya Pratama, Nur Al Faizah, Markos Lemma, Stefanie Schaedlich, Angela Zur, Magdalena Sperl, Katrin Proschek, Kristina Gratzer, and Robert Brodschneider. Smart apiculture management services for developing countries—the case of SAMS project in Ethiopia and Indonesia. *PeerJ Computer Science*, 7:e484, 2021. Accessed: 2025-05-15.

[21] Evan You and Vite contributors. Vite - next generation frontend tooling. https://vitejs.dev/, 2024. Accessed: 2025-05-12.

ДОДАТКИ А

Приклад коду API

Замість простого методу створення, наведемо приклад методу для повторного надсилання листа верифікації з `AuthController`, оскільки він демонструє взаємодію з декількома сервісами та обробку специфічного сценарію користувача:

```
// Backend - AuthController - resendVerificationEmail method
@Post('resend-verification-email')
@Version('1')
@ApiOperation({ summary: 'Resend email verification link' })
@ApiBody({ schema: { properties: { email: { type: 'string', format: 'em
async resendVerificationEmail(@Body('email') email: string) {
  return this.authService.resendVerificationEmail(email);
}

// Corresponding AuthService method snippet (conceptual)
async resendVerificationEmail(email: string): Promise<{ message: string
  const user = await this.usersService.findByEmailWithDocument(email);
  if (!user) {
    throw new BadRequestException('User with this email does not exist.
  }
  if (user.isEmailVerified) {
    throw new BadRequestException('Email is already verified.');
  }
  const verificationToken = crypto.randomBytes(32).toString('hex');
  const verificationExpires = new Date();
  verificationExpires.setHours(verificationExpires.getHours() + 1);
  await this.usersService.updateEmailVerificationToken(
    user._id.toString(), verificationToken, verificationExpires
  );
  await this.emailService.sendVerificationEmail(
    user.email, user.username, verificationToken
  );
  return { message: 'A new verification email has been sent...' };
}
```

ДОДАТКИ Б

Приклад коду клієнтської частини

Наведемо приклад функції з компонента `MapPage.tsx`, що відповідає за визначення кольору полігону поля залежно від дат обробки:

```
// Frontend - MapPage.tsx - getFieldTreatmentStatus function
const getFieldTreatmentStatus = (treatmentDates?: string[]) => {
  if (!treatmentDates || treatmentDates.length === 0) {
    return { color: 'blue', status: 'normal' }; // FIELD_COLOR_DEFAULT
  }
  const today = new Date();
  let isSoon = false;
  const TREATMENT_SOON_DAYS = 7;

  for (const dateString of treatmentDates) {
    const treatmentDate = new Date(dateString);
    // isSameDay helper function compares year, month, day
    if (isSameDay(treatmentDate, today)) {
      return { color: 'red', status: 'today' }; // FIELD_COLOR_TREATMEN
    }
    const diffTime = treatmentDate.getTime() - today.getTime();
    const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));
    if (diffDays >= 0 && diffDays <= TREATMENT_SOON_DAYS) {
      isSoon = true;
    }
  }
  if (isSoon) {
    return { color: 'orange', status: 'soon' }; // FIELD_COLOR_TREATMEN
  }
  return { color: 'blue', status: 'normal' };
};
```

Також, приклад створення кастомної іконки для вуликів з використанням MUI та Leaflet:

```
// Frontend - MapPage.tsx - Custom hive icon definition
import ReactDOMServer from 'react-dom/server';
```

```
import HiveIcon from '@mui/icons-material/Hive';
import L from 'leaflet';

const hiveLeafletIcon = L.divIcon({
  html: ReactDOMServer.renderToString(<HiveIcon sx={{ fontSize: 30, col
  className: 'leaflet-mui-icon',
  iconSize: [30, 30],
  iconAnchor: [15, 30],
  popupAnchor: [0, -30]
});
// This icon is then used in <Marker icon={hiveLeafletIcon} ... />
```

ДОДАТКИ В

Документація API (Swagger)

Документація API була автоматично згенерована за допомогою Swagger (OpenAPI) і доступна за ендпоінтом /docs на сервері розробки.

Placeholder for Swagger UI Screenshot

Рис. В.1. Приклад інтерфейсу Swagger UI для API

ДОДАТКИ Г

Інструкція користувача

Реєстрація та вхід

Опис процесу реєстрації та входу, включаючи верифікацію email, можливість повторного запиту листа верифікації, та вхід через Google.

Використання форуму

Як створювати теми, писати повідомлення, коментувати.

Робота з картою

Інтерактивна карта дозволяє користувачам управляти інформацією про свої вулики та поля.

- **Додавання вулика:** Натисніть кнопку ”Додати вулик потім клікніть на карті у місці розташування вулика. У діалоговому вікні введіть назву та нотатки, потім збережіть.
- **Додавання поля:** Натисніть кнопку ”Додати поле”. Кліками на карті позначте кути вашого поля (мінімум 3 точки). Після завершення малювання натисніть ”Завершити малювання”. У діалоговому вікні введіть назву поля, тип культури, період цвітіння та дати запланованих обробок. Збережіть.
- **Перегляд інформації:** Клікніть на маркер вулика або полігон поля на карті, щоб відкрити спливаюче вікно з детальною інформацією.
- **Редагування поля:** У спливаючому вікні поля натисніть на іконку редагування (олівець). У діалоговому вікні, що відкриється, змініть необхідні дані та збережіть.
- **Видалення вулика:** У спливаючому вікні вулика натисніть на іконку видалення (кошик). Підтвердіть дію у діалоговому вікні.
- **Кольорове кодування полів:** Поля на карті автоматично змінюють колір для індикації статусу обробки: синій – стандартний, помаранчевий – обробка запланована протягом найближчих 7 днів, червоний – обробка запланована на сьогодні.