

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Система управління спільнотами та подіями»

Виконав студент групи ПЗ-20-1
СКАКОВСЬКИЙ Владислав Олександрович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
МОРОЗОВ Андрій Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СКАКОВСЬКИЙ Владислав Олександрович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Система управління спільнотами та подіями»,**

затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74/с

Вихідні дані для роботи: **веборієнтована система управління спільнотами та подіями з використанням принципів чистої архітектури, хмарних технологій та пошукового двигуна.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С	27.02.2024р.	12.04.2024р.
2	Савіцький Р.С	27.02.2024р.	16.05.2024р.
3	Савіцький Р.С	27.02.2024р.	30.05.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку системи для управління спільнотами та подіями та пояснювальної записки. Пояснювальна записка до випускної роботи складається із вступу, 3-х розділів, висновків, списку використаних джерел із 18 найменувань, 1-го додатку, 31 таблиці та 96-х ілюстрацій по тексту. Повний обсяг роботи становить 100 сторінок, з них містить 84 сторінок основного тексту.

Метою роботи є створення системи для управління спільнотами та подіями із використанням технології контейнеризації, інтеграцією хмарних сервісів та пошукового двигуна.

В першому розділі проаналізовано проєкт та поставлено задачу на кваліфікаційну роботу. Виконано порівняння аналогічних продуктів та визначено основний функціонал додатку, обґрунтовано вибір технологій.

В другому розділі спроектовано систему для управління спільнотами та подіями, визначено об'єктно-орієнтовану структуру програмного продукту. Складено алгоритми роботи системи, розроблено та описано програмний код застосунку.

В третьому розділі описано схему розгортання системи. Наведено приклади інтерфейсу додатку. Проведено тестування, використовуючи підходи юніт- та мануального тестування та браузерні інструменти.

В результаті було розроблено програмний продукт із зручним адаптивним інтерфейсом, який дозволяє користувачам організовувати спільноти та події, вступати у них та вести комунікацію із учасниками.

КЛЮЧОВІ СЛОВА: СПІЛЬНОТА, ПОДІЯ, ЧИСТА АРХІТЕКТУРА, SPA, WEB.

					ІПЗ.КР.Б-121-24-ІПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Скаковський В.О.			Система управління спільнотами та подіями Пояснювальна записка	Літ.	Арк.	Аркуші
Керівник		Савіцький Р.С.					2	100
н. контр.		Тичина О.П.				Житомирська політехніка, група ІПЗ-20-1		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor thesis consists of a community and event management system web application and an explanatory note. The explanatory note to the final thesis consists of an introduction, 3 chapters, conclusions, a list of used sources with 18 titles, 1 appendix, 31 tables and 94 illustrations in the text. The full volume of the work is 99 pages, of which 84 pages contain the main text.

The goal of the work is to create a system for managing communities and events using containerization technology, integration of cloud services and a search engine.

In the first section, the project was analyzed and the task for the qualification work was set. A comparison of similar products was made, the main functionality of the application was determined, and the choice of technologies was justified.

In the second section, a system for managing communities and events is designed, and the object-oriented structure of the software product is defined. Algorithms for the system's operation were compiled, and the software code of the application was developed and described.

The third section describes the system deployment scheme. Screenshots of the application interface are provided. Testing was conducted using unit and manual testing approaches and browser tools.

The result was a software product with a user-friendly, responsive interface that allows users to organize, join, and communicate with communities and events.

KEYWORDS: COMMUNITY, EVENT, CLEAN ARCHITECTURE, SPA, WEB.

					III3.KP.Б-121-24-П3	
						3

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ	8
1.1. АНАЛІЗ ПРОЄКТУ ТА ПОСТАНОВКА ЗАДАЧІ	8
1.2. АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ	9
1.3. ВИБІР АРХІТЕКТУРИ СИСТЕМИ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ	15
1.4. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	21
Висновки до першого розділу	25
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ	26
2.1. ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ТА ОБ’ЄКТНО-ОРІЄНТОВАНОЇ СТРУКТУРИ СИСТЕМИ.....	26
2.2. Розробка бази даних системи.....	34
2.3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АЛГОРИТМІВ РОБОТИ СИСТЕМИ.....	43
2.4. РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ СИСТЕМИ ОРГАНІЗАЦІЇ СПІЛЬНОТ ТА ПОДІЙ	45
Висновки до другого розділу.....	52
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СИСТЕМОЮ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ	53
3.1. ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ ПАРАМЕТРІВ ПЛАТФОРМИ	53
3.2. СТРУКТУРА ІНТЕРФЕЙСУ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ	54
3.3. ТЕСТУВАННЯ ПЛАТФОРМИ.....	79
Висновки до третього розділу	81
ВИСНОВКИ.....	82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	83
ДОДАТКИ.....	85

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

IT – інформаційні технології

ASR – Architecturally Significant Requirements

CQRS – command and query responsibility segregation

CRUD – create, read, update, delete

DI – dependency injection

EF – Entity Framework

JSX – JavaScript Syntax eXtension

LINQ – language integrated query

ORM – object-relational mapping

SPA – single-page application

SQL – structured query language

SSR – server-side rendering

ВСТУП

Актуальність теми полягає в розробці додатку з використанням технологій для виконання повнотекстового пошуку та структурного логування, інтеграції хмарних сервісів та технології контейнеризації застосунку та слідуванням засад чистої архітектури. Повнотекстовий пошуковий двигун забезпечує високу швидкодію пошукових запитів у системі для управління спільнотами та подіями. Структурне логування допоможе у відстеженні та діагностиці проблем в системі. Використання хмарних технологій надасть гнучкість, масштабованість та надійність. Чиста архітектура допомагає створити стійку, модульну та легку в обслуговуванні систему.

Поширення дистанційної роботи та навчання призвело до кратного збільшення потреби у проведенні онлайн подій та організації інтернет-спільнот, а культура самонавчання ще більше підняла інтерес у відвідуванні таких заходів.

Із початком повномасштабного вторгнення РФ в Україну, волонтерський рух почав розвиватися стрімкими темпами. Люди з усіх куточків країни почали робити збори на потреби воїнів Збройних Сил України. Щоб заохочувати інших підтримувати збори, вони почали організовувати благодійні розіграші, концерти, аукціони, та інші події.

Тому, розробка системи для управління спільнотами та подіями є на часі.

Метою випускної кваліфікаційної роботи бакалавра є розробка системи для управління спільнотами та подіями.

Завдання обумовлені встановленою метою:

- проведення аналізу систем для управління спільнотами та подіями;
- визначення архітектури та узагальненої структури системи;
- вибір засобів реалізації;
- проектування об'єктно-орієнтованої структури системи;
- розробка бази даних системи;
- проектування та реалізація алгоритмів системи;
- реалізація функціоналу;
- розгортання системи;

					ІПЗ.КР.Б-121-24-ПЗ	
						6

- опис структури інтерфейсу;
- тестування платформи.

Об'єктом дослідження є веборієнтована система управління спільнотами та подіями.

Предметом дослідження є використання принципів чистої архітектури, хмарних технологій та пошукового двигуна для розробки системи для управління спільнотами та подіями.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези [1].

					ІПЗ.КР.Б-121-24-ПЗ	
						7

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ

1.1. Аналіз проєкту та постановка задачі

Метою роботи є створення системи для управління спільнотами та подіями із використанням технології контейнеризації, інтеграцією хмарних сервісів та пошукового двигуна та використанням принципів та патернів розробки ПЗ задля забезпечення стабільності та високої продуктивності системи.

Використання вебзасобів для управління спільнотами та подіями стало широко розповсюджуватися із поширенням дистанційної роботи та навчання, культури відвідуванням подій у вигляді вебінарів, конференцій та презентацій та обміну досвідом і інтересами онлайн. Особливо бурхливо ця сфера почала розвиватися за часи пандемії коронавірусу. Зважаючи на плюси та мінуси проведення онлайн подій, заперечення ефективності офлайн подій є недоцільним. Що в одному, що в іншому випадку, мати сторінку спільноти чи події в інтернеті стало невід’ємною частиною в їх рекламі чи просунені.

Подібна система може мати широкий функціонал, але для її використання, потрібні наступні базові модулі:

- модуль авторизації, в якому користувач повинен мати змогу зареєструватися, щоб користуватися системою як Учасник та Організатор;
- модуль пошуку спільноти або події допоможе знайти спільноту або подію, яка задовольняє деякі критерії, наприклад, місце проведення, дата або сфера діяльності, потрібен ефективний пошук;
- в модулі детальної інформації про спільноту, Організатор має мати змогу додати усю необхідну інформацію, налаштувати посилання на групи та чати, щоб учасники могли ознайомитися із ними;
- модуль детальної інформації про подію дасть змогу Організатору створювати події від спільноти, додавати медіа-файли для брендбука, спікерів, розклад доповідей;

- модуль вступу до спільноти та реєстрації на відвідування події, в якому Учасник має мати змогу вступити у спільноту та зареєструватися як відвідувач події, щоб отримати більш детальну інформацію про її місце проведення.

Також, для більш ефективної роботи із системою, вона має мати наступні додаткові модулі, які кратно підвищують зручність для користувачів:

- модуль спілкування дає змогу Організатору спілкуватися із учасниками спільноти та події, наприклад, методом відправки листів на електронну пошту;
- модулі скасування події, в якому Організатор повинен мати змогу скасувати подію і автоматично сповістити учасників про це;
- модуль створення форм дає можливість Організатору спільноти створити форму, яку користувачі мають заповнити, щоб вступити у спільноту;
- модуль підтвердження участі у події дає доступ Організатору налаштувати подію таким чином, щоб брати участь у ній могли лише учасники, яких він підтвердить.

Таким чином система управління спільнотами та подіями матиме достатній функціонал для зручного користування.

1.2. Аналіз аналогів програмного продукту

На ринку доволі велика кількість аналогів, які надають функціонал створення подій та спільнот.

DOU (<https://dou.ua>) – найбільша IT-спільнота в Україні (рис. 1.1). На DOU зареєстровано 600 тис. користувачів, більш як 14 тис. компаній, а щомісяця статті й топіки збирають понад 2,5 млн переглядів (а весь сайт – 9 млн) [2].

Також DOU відомий проведенням різноманітних опитувань та досліджень, таких як аналітика зарплат українських IT-спеціалістів, портрет IT-спеціаліста, кількість співробітників в українських IT-компаніях, рейтинг роботодавця, рейтинг мов програмування та іншими [3].

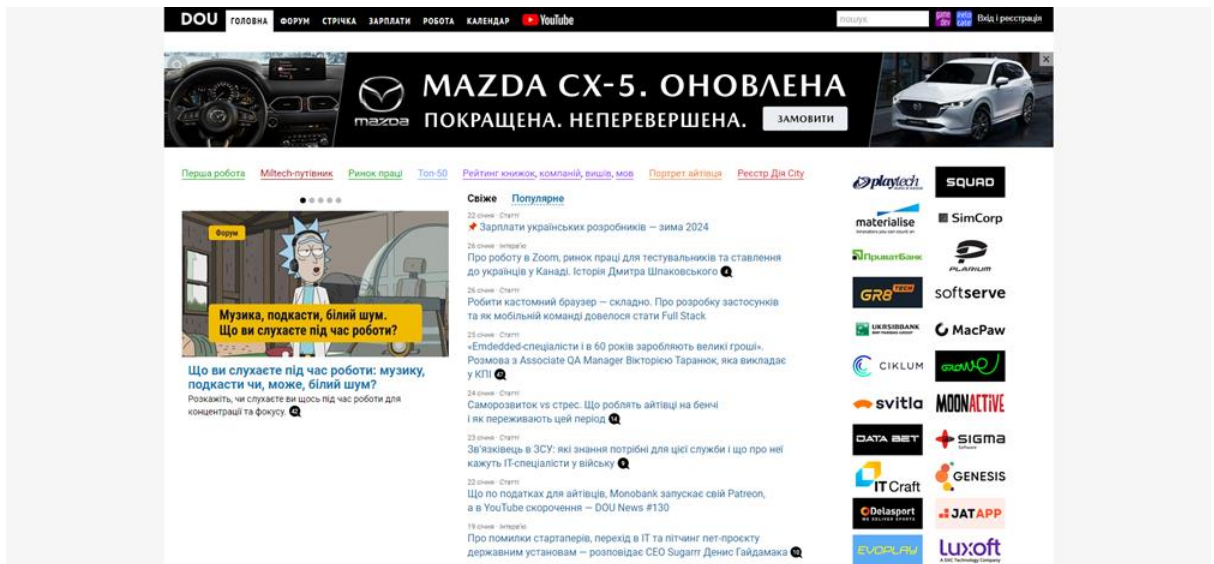


Рис. 1.1 – Домашня сторінка Dou.ua

DOU має розділ «Календар», де є можливість переглянути заплановані та вже проведені події (рис. 1.2 – 1.3). Також є можливість фільтрації за містом проведення або онлайн та за темою. Зважаючи на те що DOU це сайт з основною тематикою IT, тому і теми подій є відповідними.

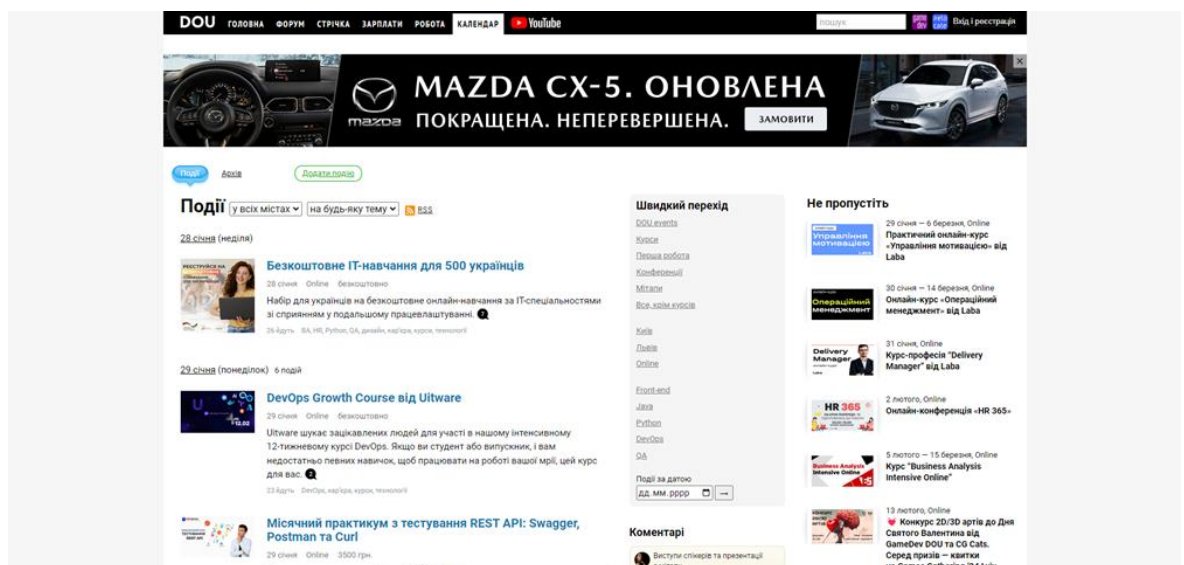
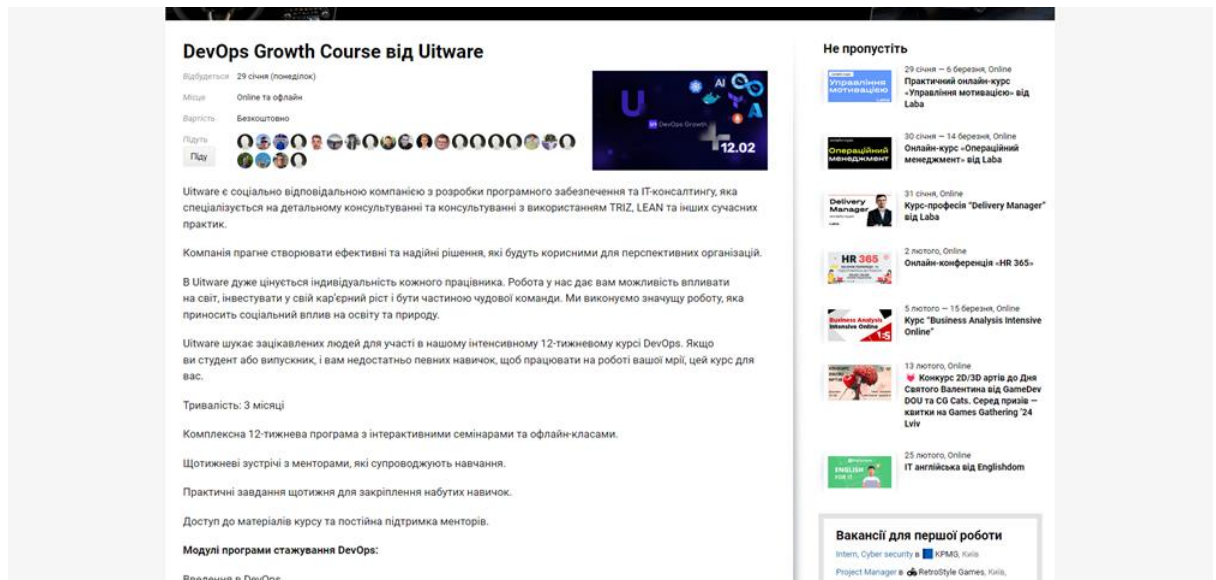


Рис. 1.2 – Сторінка зі списком подій Dou.ua



Додати подію на сайт може будь-хто. Розміщення анонсів некомерційних ІТ-подій і великих ІТ-конференцій і заходів безкоштовне, потрібно просто заповнити форму (рис. 1.4). Редакція календаря лишає за собою право відмовляти у безкоштовній публікації без пояснення причин. Анонси платних курсів (і промо-вебінарів/семінарів до них), тренінгів, майстер-класів і т.д. публікуються за умови придбання розміщення за 40\$ в Календарі [4].

Після додавання події, вона відправляється на модерацію.

Преваги:

- простий інтерфейс;
- можливість створювати події із платними квитками;

- можливість гнучко налаштовувати опис події.

Недоліки:

- немає функціоналу спільнот;
- запис на подію є формальним, бо лише відображає факт запису;
- немає можливості створення форми для запису на подію.

Kommunity (<https://kommunity.com>) – сайт, який спеціалізується на створенні платформ для ком'юніті та організації подій (рис. 1.5). Вона включає в себе менеджмент ком'юніті та подій (рис. 1.6), call for papers, продажу квитків і безкоштовну реєстрацію (рис. 1.7), менеджмент розкладу та спікерів події, можливість спілкування зі спільнотою, транслявання події онлайн, збір відгуків, завантаження матеріалів події та інше. Kommunity включає в себе велику кількість інструментів необхідних задля ефективної організації подій.

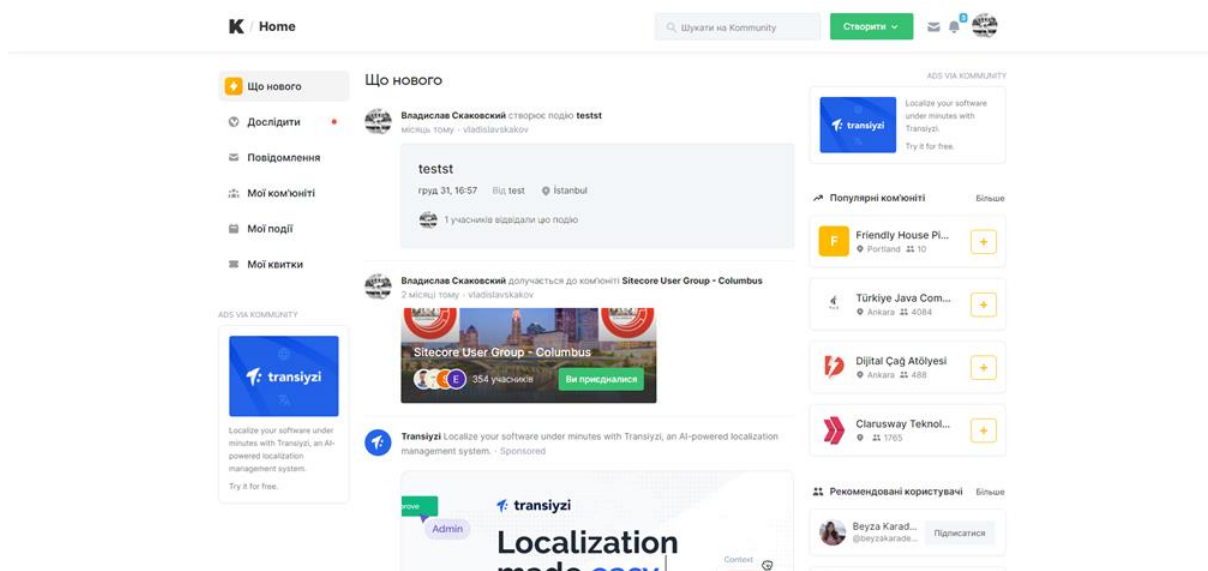


Рис. 1.5 – Домашня сторінка Kommunity

Рис. 1.6 – Сторінка створення події Community

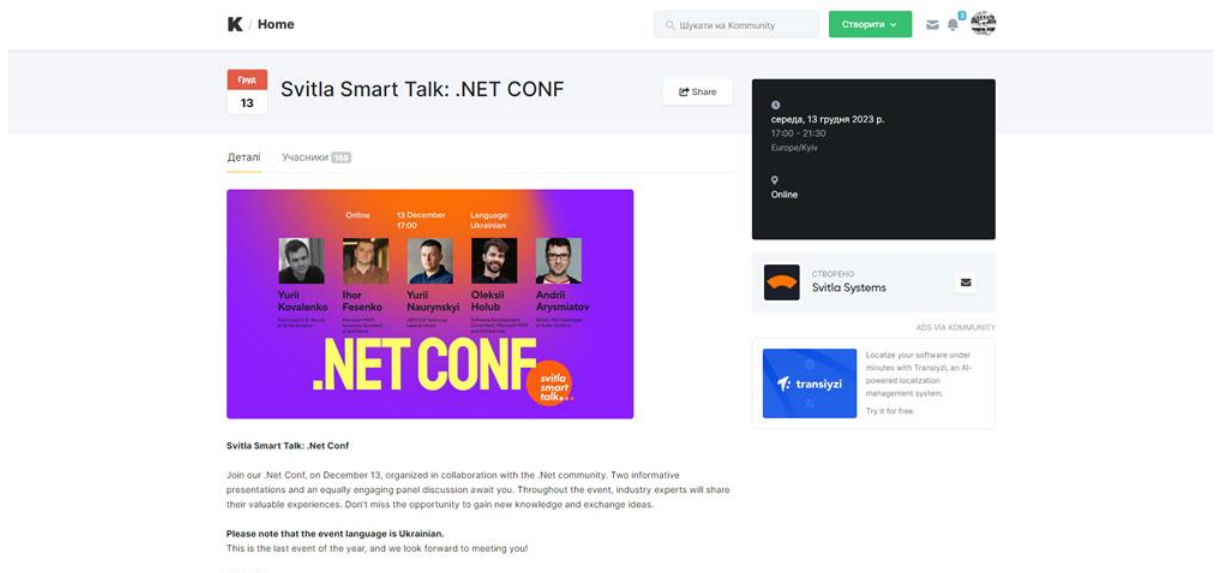


Рис. 1.7 – Сторінка події Community

Переваги:

- гнучкість у налаштуванні спільнот і подій;
- можливість створювати події із платними квитками;
- створення форм для реєстрації та опитувань;
- статистика по запису на події.

Недолік:

- не зручний функціонал пошуку.

Facebook (<https://www.facebook.com>) – найбільша соціальна мережа [5]. Facebook належить корпорації Meta Platforms. За даними рейтингу Alexa Rank,

- складний інтерфейс.

Таблиця 1.1

Порівняльна таблиця

	Календар Dou.ua	Kommunity	Facebook Events	Власна розробка
Створення опитувань	-	+	+	+
Створення форми для вступу у спільноту	-	+	-	+
Спільноти	-	+	+	+
Форматуван ня опису події	+	+	-	+
Спікери та доповіді	-	+	-	+
Комунікація їх відвідувачам и електронною поштою	-	+	-	+
Пошук	-	-	+	+

Отже, у порівнянні з аналогами, додаток, що розробляється, має усі основні функції, необхідні для зручної роботи із системою для управління спільнотами та подіями. Крім того, в деяких аспектах, застосунок навіть перевершує їх.

1.3. Вибір архітектури системи управління спільнотами та подіями

Архітектура – це термін, який багато людей намагаються визначити, але немає згоди у цьому. Є два загальні елементи: перший – це розподіл системи на її частини на найвищому рівні; другий – рішення, які важко змінити. Також все більше визнається, що існує не один спосіб визначити архітектуру системи; натомість, у системі є кілька архітектур, і погляд на те, що є архітектурно значущим, може змінюватися протягом життєвого циклу системи [6].

Мета архітектури програмного забезпечення – зменшити трудовитрати на створення та супровід системи. Щоб створити систему, дизайн та архітектура

якої сприяють зменшенню трудовитрат і збільшенню продуктивності, потрібно знати, які елементи архітектури сприяють цьому [7].

Розробка архітектури — це комплексний процес, який починається зі взаємодії із зацікавленими сторонами. Спершу архітектор повинен зібрати від стейкхолдерів всю необхідну інформацію, аби зрозуміти, з яких компонентів складатиметься продукт, і описати, як вони пов'язані.

Далі зібрану інформацію треба проаналізувати та визначити вимоги до побудови архітектури — ASR. Вони включають функціональні вимоги, проєктні обмеження, атрибути якості (такі як безпека, доступність, продуктивність, зручність використання тощо), а також додаткові вимоги від стейкхолдерів.

Після цього архітектор створює дизайн, який підходить елементам продукту і задовольняє всі ASR, та створює архітектурну документацію. Розробляючи дизайн, архітектори користуються вже відомими дизайн-концептами, серед яких еталонні архітектури, дизайн патерни та стилі, архітектурні тактики, девелопмент фреймворки тощо. Серед концептів зазвичай для кожного рішення є альтернативні опції, тому архітектор аналізує альтернативи (trade-off analysis) і обирає найкращу [8].

Проаналізувавши вимоги до системи, що розробляється, було визначено, що найбільш прийнятною є клієнт-серверна архітектура.

Клієнт-серверна архітектура завжди має два види програмних модулів — клієнт та сервер. Під терміном клієнт мається на увазі пристрій користувача, тобто персональний комп'ютер, смартфон тощо. Сервер як правило знаходиться в іншому місці, зазвичай це більш потужна обчислювальна техніка. Комунікація між клієнтом та сервером відбувається через мережу інтернет [9].

Складається із трьох компонентів (рис. 1.10):

- набір серверів, які надають інформацію або інші послуги;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між ними.

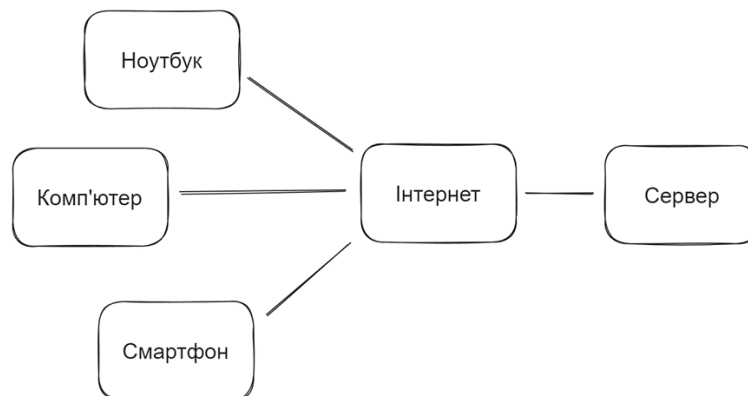


Рис. 1.10 – Клієнт-серверна архітектура

Цей вид архітектури є стандартом для вебзастосунків із використанням бази даних.

З точки зору архітектури застосунку, існує вибір між монолітною та архітектурою мікро сервісів.

Монолітна архітектура – традиційна уніфікована модель побудови програмного забезпечення (рис. 1.11). В даному контексті монолітність означає скомпонованість функціоналу в єдиному місці, масивність, неможливість бути зміненим [10]. Вона являє собою системою з єдиною кодовою базою, яка інкапсулює усю бізнес-логіку. Тому, якщо потрібно внести деякі зміни в додаток, потрібно перебудовувати і деплоїти цілу систему. Проте, моноліти є зручними на ранній стадії життя проєкту.

Переваги монолітної архітектури:

- простота розробки, деплоймента та тестування та відлагодження;
- продуктивність.

Монолітна архітектура



Рис. 1.11 – Монолітна архітектура

Мікросервісна архітектура включає в себе серію сервісів, які можуть бути незалежно задеплойовані (рис. 1.12). Кожен сервіс має свою ціль, кодову базу, бізнес-логіку та сховище даних. Через це можлива незалежна зміна, перебудова та деплоймент окремого сервісу, без впливу на інші. Це додає складності до розробки проєкту.

Переваги мікросервісної архітектури:

- гнучке масштабування;
- незалежний деплоймент;
- гнучкість в розробці.

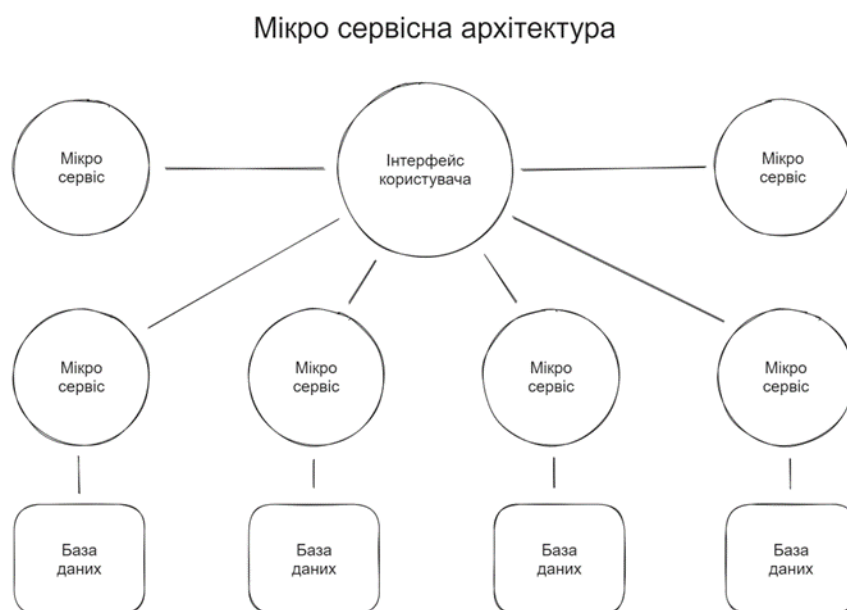


Рис. 1.12 – Мікросервісна архітектура

Гарною практикою в розробці додатку є використання монолітної архітектури за замовчуванням і, при необхідності, виокремлення окремих сервісів задля отримання деяких переваг мікро сервісів. Наприклад, масштабування одного сервісу, якщо його продуктивність у межах моноліту є недостатньою. Раннє впровадження мікросервісної архітектури може призвести до непотрібного ускладнення системи та уповільнення розробки.

Зважаючи на плюси кожної з архітектур, було обрано монолітну для розробки системи.

Найбільш відомими архітектурами для монолітної системи є трьохшарова та чиста архітектури.

В тришаровій архітектурі додаток логічно розділено на три частини:

- інтерфейс користувача або шар презентації. Це шар з яким користувач безпосередньо взаємодіє;
- шар бізнес логіки. Він включає в себе набір компонентів, які відповідальні за обробку даних, отриманих з інтерфейсу користувача;
- шар доступу до даних. Він має моделі даних. Також він включає в себе конкретні класи для роботи із різними технологіями доступу до даних, таких як ORM.

Важливо зазначити те як ці рівні залежать один від одного (рис. 1.13). Шар доступу до даних є незалежним. Шар бізнес логіки залежить від доступу до даних, а користувацького інтерфейсу залежить від бізнес логіки. Отже, дані та те як вони зберігаються є найбільш важливою частиною додатку і зміни в цьому шарі призведуть до необхідності змін в інших частинах додатку.



Рис. 1.13 – Тришарова архітектура

Чиста архітектура це також архітектура шарів. Шар домену (сутності) є в центрі і оточений шаром додатку (варіантів використання). Зовнішній шар складається із портів та адаптерів, які адаптують додаток до зовнішніх систем (веб, бази даних, користувацького інтерфейсу) використовуючи контролери, репозиторії і презентери.

Ця архітектура є доменно центричною. Вона ставить домену модель в центр усього додатку. Доменна модель включає як поведінку, так і дані, але не визначає взаємодію із базою даних. Збереження даних і представлення – це лише деталі, розташовані якомога далі від центру.

Крім поділу шарів потрібно брати до уваги і правила залежностей (рис. 1.14):

- залежності не можна встановити в обхід шарів;
- шари високого рівня не повинні залежати від шарів низького рівня.

Це може бути досягнуто використанням механізму впровадження залежностей (DI).

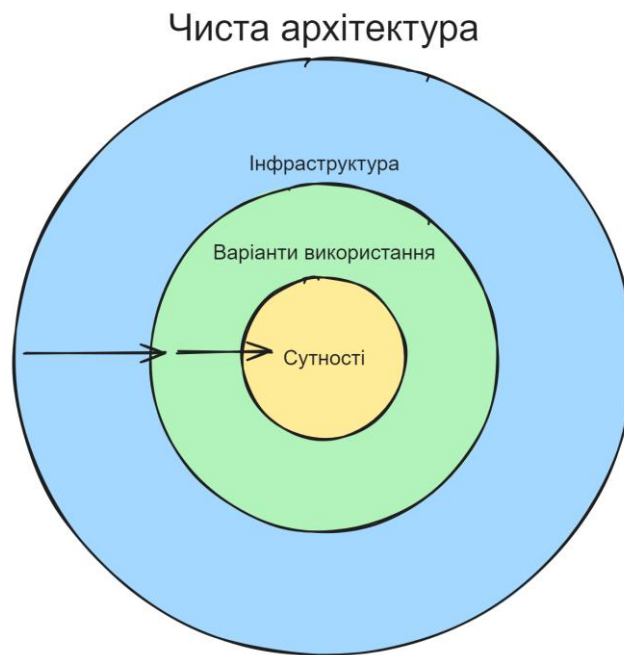


Рис. 1.14 – Чиста архітектура

Отже, очевидно, що тришарову архітектуру краще використовувати для невеликих проєктів. Також її краще використовувати для проєктів, які:

- не включають багато бізнес логіки;
- мають бути розроблені швидко;
- не очікується, що матимуть зміни в технологіях і мають невелику кількість інтеграцій.

Щодо чистої архітектури, її можна використовувати, коли зрозуміло, що проєкт буде складний. Наприклад:

- ентерпрайз рішення з великою кількістю бізнес логіки;
- проєкти, які будуть рости і будуть адаптувати стек технологій під їх розмір [11].

Зважаючи на перелічені фактори, для системи було обрано чисту архітектуру із деякими корективами для її спрощення.

1.4. Обґрунтування вибору засобів реалізації та вимоги до апаратного забезпечення

Вибір засобів реалізації є важливим етапом при розробці програмного забезпечення. Від цього залежатиме якість розробленого додатку, його продуктивність і сама швидкість реалізації запланованого функціоналу.

Так як система має бути вебдодатком, логічно розділити технології для фронтенду та бекенду.

Мовою програмування бекенду було обрано C#. C# (вимовляється як «сі шарп») – сучасна, об’єктно-орієнтована, строго типізована мова програмування. C# дозволяє розробникам створювати багато видів безпечних і надійних програм, які працюють в .NET.

C# – об’єктно-орієнтована, компонентно-орієнтована мова програмування [12]. C# надає мовні конструкції для безпосередньої підтримки цих концепцій, що робить C# природною мовою для створення та використання програмних компонентів. З моменту свого створення, до C# було додано функції для підтримки нових робочих навантажень і новітніх практик проєктування програмного забезпечення. За своєю суттю, C# є об’єктно-орієнтованою мовою. Ви визначаєте типи та їх поведінку.

Для розробки вебдодатків, платформа .NET надає фреймворк ASP.NET Core. ASP.NET Core – це кросплатформна, високопродуктивна структура з відкритим вихідним кодом для створення сучасних хмарних додатків, підключених до Інтернету [13].

Вона дозволяє:

- створювати вебпрограми та служби;
- використовувати інструменти розробки в Windows, macOS і Linux;
- розгорнути в хмарі або локально;
- запустити на .NET.

Предметна область включає в основному дані, які мають чітку структуру і велику кількість зв’язків. Документно-орієнтована база даних не підійде для цієї задачі, тому для збереження даних було обрано реляційну базу даних. Працюючи зі стеком технологій Microsoft, вибір системи управління реляційними базами даних впав на SQL Server.

Microsoft SQL Server – комерційна система керування базами даних, що розповсюджується корпорацією Microsoft. Мова, що використовується для запитів – Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є

реалізацією стандарту ANSI / ISO щодо структурованої мови запитів SQL із розширеннями. Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних. MS SQL використовую реляційну модель СУБД для контролю цілісності даних [14].

Серед переваг MS SQL Server є:

- різноманітність версій;
- end-to-end рішення для бізнес-даних;
- багата документація та велике ком'юніті;
- підтримка хмарних баз даних.

В об'єктно-орієнтованому середовищі для роботи із базою даних зручно використовувати механізм ORM. Цей інструмент дає змогу перетворювати об'єкти класів на записи в базі даних. Таким чином створюється «віртуальна об'єктна база даних», що може бути використана в мові програмування.

Для платформи .NET найбільш популярними ORM є Dapper та Entity Framework Core (EF Core). Dapper не має влаштованого механізму міграцій, CRUD операцій і при зміні структури таблиць, SQL запити потрібно оновлювати вручну. EF Core не має цих мінусів, тому його і було обрано як інструмент розробки.

Entity Framework Core – це легка, розширювана кросплатформна версія популярної технології доступу до даних Entity Framework із відкритим кодом [15].

EF Core може служити об'єктно-реляційним картографом (O/RM), який:

- дозволяє розробникам .NET працювати з базою даних за допомогою об'єктів .NET;
- усуває потребу в більшості коду доступу до даних, який зазвичай потрібно писати.

EF Core дозволяє використовувати інтерфейс LINQ для взаємодії із базою даних, що дає неабияку гнучкість та зручність у використанні.

Для зменшення зв'язності системи, відокремлення запитів від обробників та для логічної імплементації CQRS було обрано для використання бібліотеку MediatR. Для цього вона реалізує патерн посередник.

Валідація вхідних даних у систему це процес необхідний кожному проєкту. Стандартний інструмент в ASP.NET Core на основі атрибутів гарно із цим справляється, проте коли необхідно побудувати щось складніше, створення кастомних атрибутів не завжди дає очікуваний результат, який легко зрозуміти читаючи код. Тому для реалізації валідації обрано бібліотеку FluentValidation. Сама назва говорить, що бібліотека надає fluent інтерфейс для створення правил валідації, що є доволі зручним виходом із ситуації.

Для реалізації пошуку та структурного логування було обрано Elasticsearch – пошуковий сервіс, розроблений на базі Lucene [16]. За його допомогою можна швидко та легко індексувати та виконувати пошук по індексу, реалізувати повнотекстовий пошук, а графічним інтерфейсом для нього слугує Kibana.

Так як для фронтенду безперечним лідером серед мов програмування є JavaScript. Проте у неї є значний мінус – вона має динамічну типізацію. Для вирішення цієї проблеми було створено мову програмування TypeScript.

TypeScript – це безкоштовна високорівнева мова програмування з відкритим вихідним кодом, розроблена корпорацією Майкрософт, яка додає до JavaScript статичну типізацію із не обов'язковими анотаціями типу. Він розроблений для розробки великих додатків і транспілюється на JavaScript. Оскільки TypeScript є надмножиною JavaScript, усі програми JavaScript є синтаксично дійсними TypeScript, але вони можуть не пройти перевірку типів з міркувань безпеки [17].

Для розробки було визначено, що архітектурно, фронтенд додаток має бути SPA. Серед SPA фреймворків та бібліотек, найбільш популярним є React. Він надає змогу будувати інтерфейс користувача використовуючи компоненти, написані синтаксисом JSX. Його можна використовувати не тільки для SPA, а і для побудови мобільних та SSR додатків. React підтримує як JavaScript так і

TypeScript. Так як він є найбільш розповсюдженою бібліотекою для розробки SPA додатків, він має велике активне ком'юніті.

Система, що розробляється, включає велику кількість форм. Використання стандартних інструментів React приведе до роботи із елементами форми на низькому рівні та дублювання коду. Тому вирішено використати спеціальну бібліотеку для цього. Наразі є декілька відомих бібліотек які справляються із цим завданням. Вибір впав на react-hook-form. Це легка та продуктивна та легка для розширення бібліотека з інтуїтивним програмним інтерфейсом, яка дозволяє будувати гнучкі форми із складними механізмами валідації.

Для асинхронної взаємодії із сервером та менеджменту стану обрано бібліотеку TanStack Query. Вона також надає механізм кешування, синхронізації та оновлення стану серверу в вебдодатку «прямо з коробки».

Система складатиметься із кількох підсистем: вебсервер, клієнтська частина, сервери Elasticsearch та Kibana, було обрано інструмент контейнеризації Docker.

Для збереження медіа-файлів було обрано хмарний сервіс провайдера Azure – Azure Storage Account, зокрема Blob Storage. Він має зручний програмний інтерфейс і широкий функціонал для роботи із файлами.

Висновки до першого розділу

Було поставлено задачу на кваліфікаційну роботу бакалавра. Визначено основні модулі необхідні для імплементації для забезпечення зручності використання вебдодатком.

Виконано дослідження існуючих аналогів, виявлено їх плюси та мінуси. Зроблено зведену таблицю із їх порівнянням та порівнянням власної розробки на кваліфікаційну роботу бакалавра.

Було порівняно можливі архітектурні рішення для додатку, виявлено їх переваги та недоліки для різних видів програмних систем. Обрано архітектуру для відповідного застосунку.

Обрано технології для реалізації бекенду та фронтенду у відповідності до вимог.

					ІПЗ.КР.Б-121-24-ПЗ	
						25

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ

2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи

Система управління спільнотами та подіями є веборієнтованою, тому взаємодія із нею проходитиме через браузер. Використовуючи вебпортал, користувачі зможуть входити у систему. Тому, проаналізувавши предметну область, було виділено три види акторів: учасники, які можуть відвідувати події, організатори, які можуть створювати спільноти та події і водночас використовувати функціонал учасників, та адміністратор, який має доступ до інформації про всіх користувачів. Взаємодія зі спільнотами і подіями на рівні учасника включає в себе пошук, перегляд детальної інформації та можливість стати учасником. На рівні організатора включає менеджмент спільнот і подій, їх учасників, конструювання реєстраційної форми, комунікація з учасниками. Далі представлено наочне відображення цього у діаграмі варіантів використання (рис. 2.1) та її опис (табл. 2.1 – 2.7).

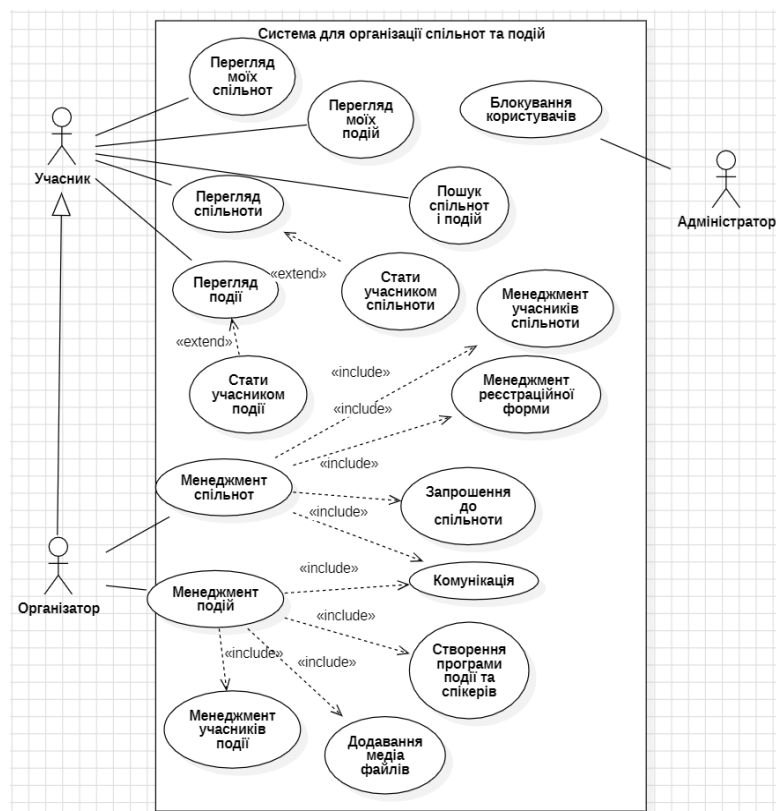


Рис. 2.1 – Діаграма варіантів використання

Детальний опис діаграми використання наведено у табл. 2.1 - 2.7.

Таблиця 2.1

Опис прецеденту «Перегляд моїх спільнот»

Назва варіанту використання	Перегляд моїх спільнот
Основні актори	Учасник
Короткий опис	Учасник може переглядати список спільнот, в яких він є учасником

Таблиця 2.2

Опис прецеденту «Перегляд моїх подій»

Назва варіанту використання	Перегляд моїх подій
Основні актори	Учасник
Короткий опис	Учасник може переглядати список подій, в які він відвідує

Таблиця 2.3

Опис прецеденту «Пошук спільнот і подій»

Назва варіанту використання	Пошук спільнот і подій
Основні актори	Учасник
Короткий опис	Учасник може виконувати пошук спільнот і подій за заданими параметрами

Таблиця 2.4

Опис прецеденту «Перегляд спільноти»

Назва варіанту використання	Перегляд спільноти
Основні актори	Учасник
Короткий опис	Учасник може переглядати детальну інформацію про спільноту, її події та стати її учасником

Таблиця 2.5

Опис прецеденту «Перегляд події»

Назва варіанту використання	Перегляд події
Основні актори	Учасник
Короткий опис	Учасник може переглядати детальну інформацію про подію, місце проведення, спікерів, програму події та медіа файли

Таблиця 2.6

Опис прецеденту «Перегляд події»

Назва варіанту використання	Перегляд події
Основні актори	Учасник
Короткий опис	Учасник може переглядати детальну інформацію про подію, місце проведення, спікерів, програму події та медіа файли, подавати заявку на відвідування події

Таблиця 2.7

Опис прецеденту «Менеджмент спільнот»

Назва варіанту використання	Менеджмент спільнот
Основні актори	Організатор
Короткий опис	Організатор може займатися менеджментом спільнот, їх учасників, реєстраційної форми. Також може запрошувати користувачів до спільноти та комунікувати із учасниками спільноти шляхом надсиланням листів на електронну пошту

Таблиця 2.8

Опис прецеденту «Менеджмент подій»

Назва варіанту використання	Менеджмент подій
Основні актори	Організатор
Короткий опис	Організатор може займатися менеджментом подій, їх учасників, підтверджувати участь у події. Також може додавати спікерів до події, створювати її програму.

Таблиця 2.9

Опис прецеденту «Блокування користувачів»

Назва варіанту використання	Блокування користувачів
Основні актори	Адміністратор
Короткий опис	Адміністратор може блокувати користувачів. Таким чином, вони будуть не в змозі аутентифікуватися у системі

Зібравши варіанти використання для системи організації спільноти та подій, було виділено основні напрямки, у яких потрібно рухатися, для імплементації додатку. За цим, було складено вимоги до застосунку, що розробляється.

Вимоги учасників:

1. Перегляд спільнот та подій.
2. Можливість стати учасником спільноти.
3. Реєстрація на подію.
4. Редагування профілю.

Вимоги організаторів:

1. Створення спільнот та подій.
2. Менеджмент спікерів та програм події.

3. Створення форми реєстрації до спільноти та перегляд відповідей на неї.
4. Можливість підтвердження реєстрації відвідувача на подію.
5. Перегляд зареєстрованих на подію відвідувачів та учасників спільноти.
6. Відміна події.
7. Комунікація з учасниками спільноти та відвідувачами події шляхом надсиланням листів на електронну пошту.
8. Запрошення користувачів платформи для вступу у спільноту.
9. Додавання медіа файлів до події.

Вимоги адміністраторів:

1. Блокування користувачів.

Функціональні вимоги:

1. В системі має бути представлена можливість аутентифікації та авторизації користувачів. Повинна бути реалізована можливість реєстрації користувачів.
2. Забезпечення організаторам переглядати статистику по кількості зареєстрованих відвідувачів.
3. Можливість організаторам виконувати менеджмент спільнот та подій.
4. Забезпечення учасникам можливість перегляду спільнот та подій і вступу до них.

Не функціональні вимоги:

1. Сприйняття
 - час, потрібний для освоєння веборієнтованого додатку – 30-50 хвилин;
 - час відповіді системи для звичайних запитів не повинен перевищувати 10 секунд, а для більш складних запитів – 15 секунд;
 - інтерфейс представлення вебдодатку повинен бути інтуїтивно зручним для користувача та не вимагати від нього додаткової підготовки.

2. Надійність

- доступність;
- середній час безперервної роботи – 30 робочих днів;
- максимальна норма помилок та дефектів в роботі системи – 1 помилка на 100 запитів користувача.

3. Продуктивність

- система повинна підтримувати мінімум 20 одночасно працюючих користувачів, пов'язаних спільною БД.

Більшість програмних систем розробляються за допомогою засобів об'єктно-орієнтованого програмування. Це стало стандартом де-факто, тому слід створити об'єктно-орієнтовану модель системи. Для цього найкраще підходить діаграма класів. Проаналізувавши функціональні вимоги до додатку, було виділено основні доменні сутності (рис. 2.2).

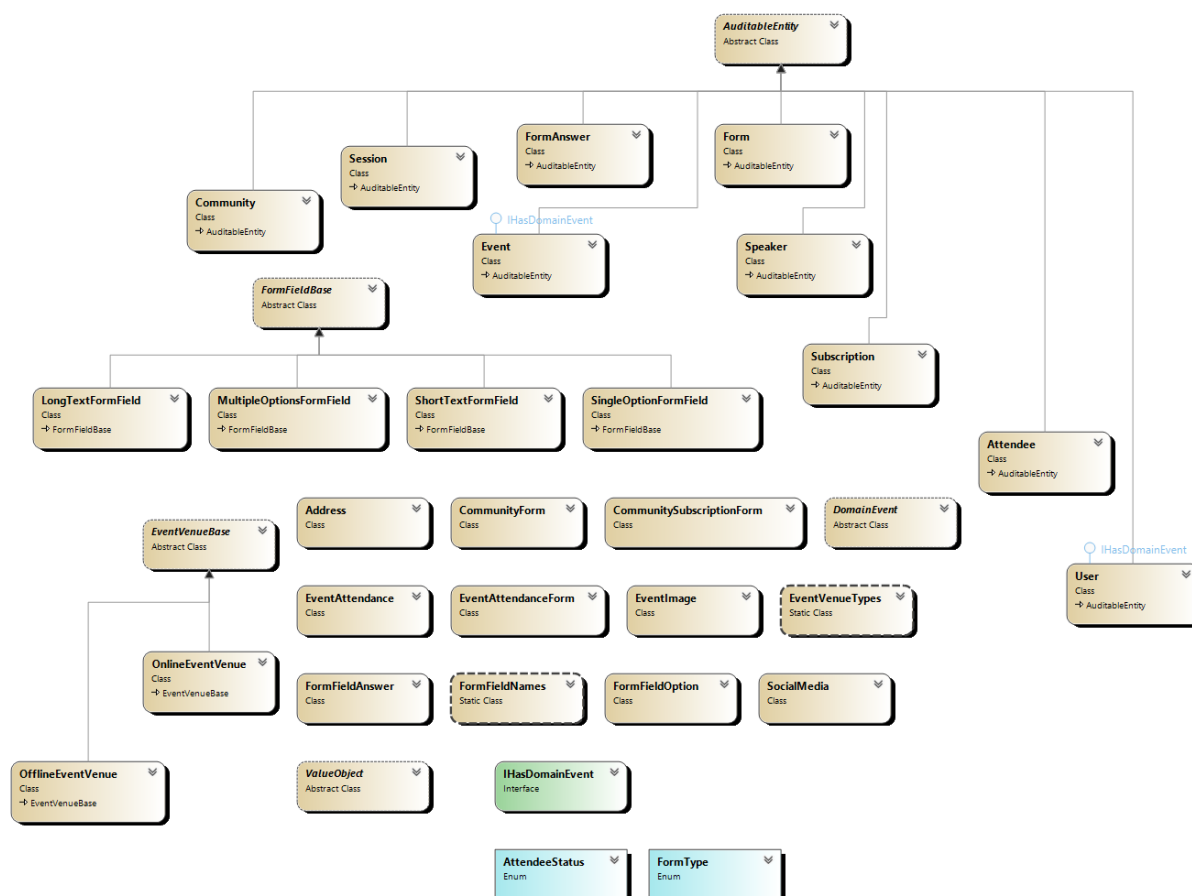


Рис. 2.2 – Діаграма класів

- AuditableEntity – базовий абстрактний клас, який має властивості для аудиту;

- Community – клас, який є сутністю спільноти;
- Session – клас, який є сутністю доповіді на події;
- FormAnswer – клас, який є сутністю заповненої учасником форми;
- Event – клас, який є сутністю події;
- Form – клас, який є сутністю форми;
- Speaker – клас, який є сутністю спікера на події;
- Subscription – клас, який є сутністю підписки учасника на спільноту;
- Attendee – клас, який є сутністю відвідувача події;
- User – клас, який є сутністю користувача системи;
- FormFieldBase – базовий абстрактний клас для поля у формі;
- LongTextFormField – клас, який відповідає за багаторядкове текстове поле у формі;
- MultipleOptionsFormField – клас, який відповідає за поле із декількома можливими обраними відповідями у формі;
- ShortTextFormField – клас, який відповідає за текстове поле у формі;
- SingleOptionFormField – клас, який відповідає за поле із однією можливою обраною відповіддю у формі;
- EventVenueBase – базовий абстрактний клас, який відповідає за місце проведення події;
- OfflineEventVenue – клас, який відповідає за офлайн місце проведення події;
- OnlineEventVenue – клас, який відповідає за онлайн місце проведення події;
- Address – клас, який відповідає за адресу місця проведення події;
- EventAttendance – клас, який відповідає за відвідування користувачем події;
- FormFieldAnswer – клас, який відповідає за заповнене користувачем поле у формі;
- CommunitySubscriptionForm – клас, який відповідає за форму для подачі заявки на участь у спільноті;

- EventImage – клас, який відповідає картинку завантажену до деталей події;
- FormFieldOption – клас, який відповідає за один можливий варіант відповіді у полі форми;
- SocialMedia – клас, який відповідає за набір посилань спільноти до її соціальних мереж.

При проєктуванні систем часто зустрічаються стандартні проблеми, на які вже є рішення. Система управління спільнотами та подіями не є виключенням, тому було застосовано кілька патернів проєктування, які і вирішують дані проблеми.

Було виокремлено патерн Unit of work в спрощеному вигляді (рис. 2.3). Для доступу до даних використовуються властивості Attendees, Communities, CommunityForms та інші, а для забезпечення транзакції виділено метод SaveChangesAsync.

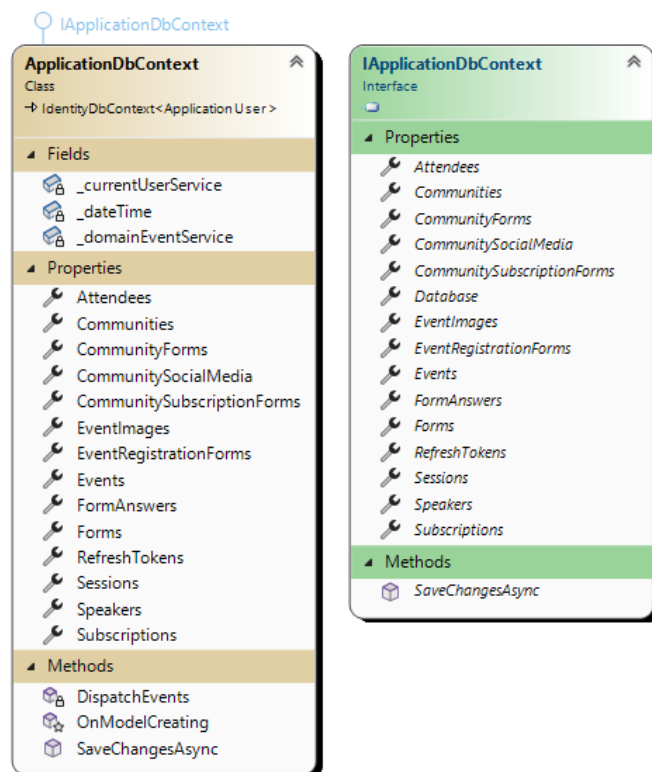


Рис. 2.3 – Патерн Unit of Work

Для роботи із файлом, отриманим через форму шляхом REST API, для того щоб не порушувати чисту архітектуру, потрібно використати проксі клас (рис.

2.4). Клас FormFileProху є обгорткою навколо інтерфейсу IFormFile і реалізує інтерфейс IFile, який знаходиться в шарі доменної логіки.

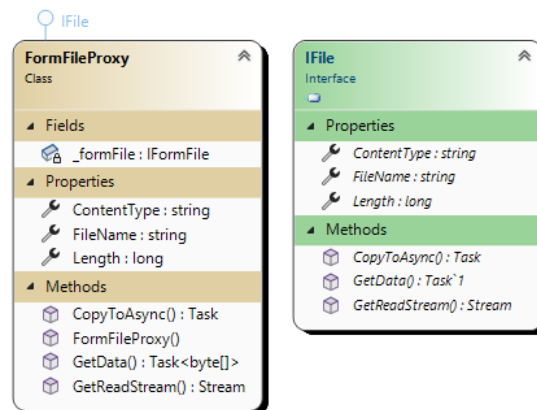


Рис. 2.4 – Патерн Проху

2.2. Розробка бази даних системи

Провівши аналіз, було виявлено, що для збереження даних найбільше підходить реляційна система управління базами даних Microsoft SQL Server.

База даних системи для управління спільнотами та подіями складається із наступних таблиць, наведених на діаграмі (рис. 2.5).

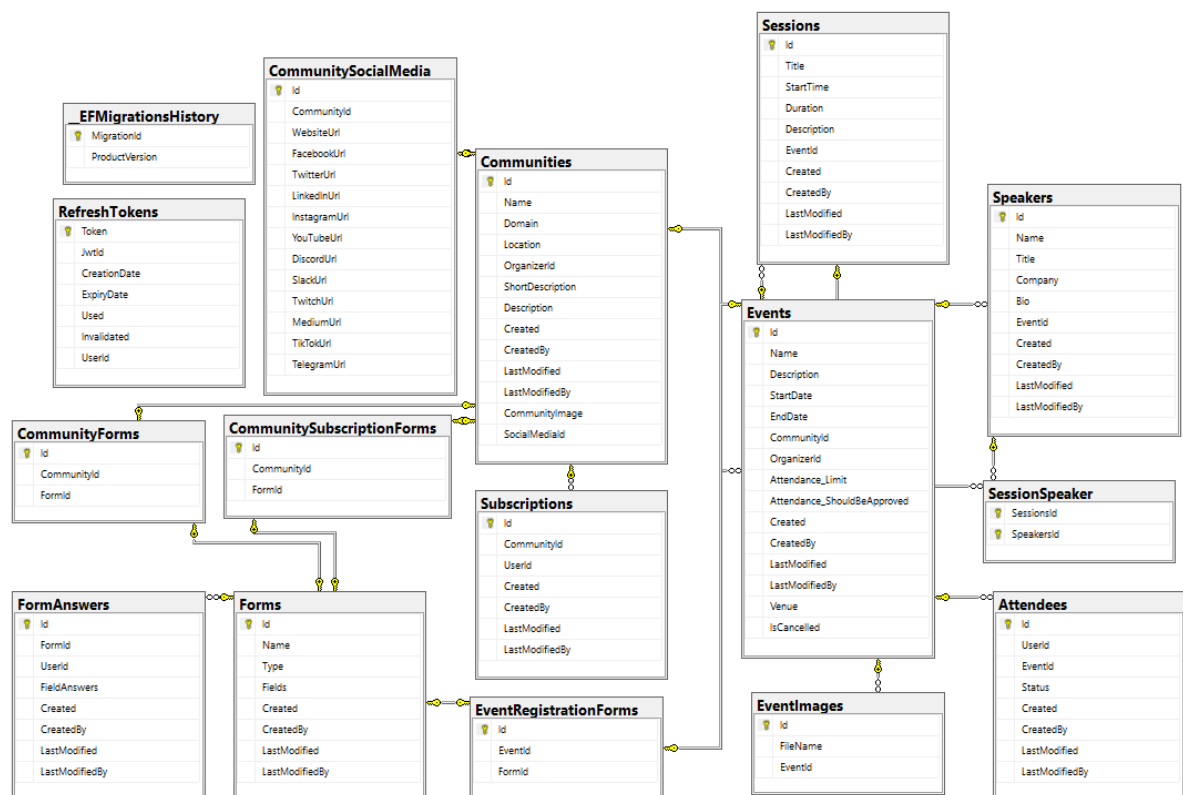


Рис. 2.5 – Діаграма бази даних

- `__EFMigrationsHistory` – таблиця, створена Entity Framework Core для трекінгу міграцій;
- `Attendees` – таблиця відвідувачів події;
- `Communities` – таблиця спільнот;
- `CommunitySocialMedia` – таблиця посилань на соціальні мережі події;
- `CommunitySubscriptionForms` – таблиця форм для заявки вступу у спільноту;
- `EventImages` – таблиця картинок події;
- `Events` – таблиця подій;
- `FormAnswers` – таблиця відповідей форми;
- `Forms` – таблиця форм;
- `RefreshTokens` – таблиця токенів оновлення;
- `Sessions` – таблиця доповідей події;
- `SessionSpeaker` – таблиця для зв'язку доповіді і спікера;
- `Speakers` – таблиця спікерів;
- `Subscriptions` – таблиця учасників спільноти.

Далі наведено детальний опис полів кожної таблиці (рис. 2.10 – 2.22).

Таблиця 2.10

Опис полів таблиці «`__EFMigrationsHistory`»

Назва	Тип даних	PK	FK	Опис
MigrationId	nvarchar(150) not null	+	-	Ідентифікаційний номер міграції
ProductVersion	nvarchar(32), not null	-	-	Версія Entity Framework Core

Таблиця 2.11

Опис полів таблиці «`Attendees`»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер відвідувача події

Продовження таблиці 2.11

UserId	nvarchar(450), not null	-	-	Ідентифікаційний номер користувача
EventId	int, not null	-	Events.Id	Ідентифікаційний номер події
Status	int, not null	-	-	Статус відвідувача (0 – в очікуванні, 1 – підтверджено, 2 – відмінено)
Created	datetime2(7), not null	-	-	Дата, коли запис було додано до таблиці
CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис

Таблиця 2.12

Опис полів таблиці «Communities»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер спільноти
Name	nvarchar(200), not null	-	-	Назва
Domain	nvarchar(200), not null	-	-	Домен
Location	nvarchar(200), not null	-	-	Локація
OrganizerId	nvarchar(450), not null	-	-	Ідентифікаційний номер користувача, який організував спільноту
ShortDescription	nvarchar(450), null	-	-	Короткий опис
Created	datetime2(7), not null	-	-	Дата, коли запис було додано до таблиці

Продовження таблиці 2.12

CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис
CommunityImage	nvarchar(200), null	-	-	Назва файлу картинки спільноти
SocialMediaId	int, not null	-	SocialMedia.Id	Ідентифікаційний номер запису про посилання на соціальні мережі спільноти

Таблиця 2.13

Опис полів таблиці «CommunitySocialMedia»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер запису посилань на соціальні мережі спільноти
CommunityId	int not null	-	Communities.I d	Ідентифікаційний номер спільноти
WebsiteUrl	nvarchar(200), null	-	-	URL вебсайту
FacebookUrl	nvarchar(200), null	-	-	URL Facebook акаунту
TwitterUrl	nvarchar(200), null	-	-	URL Twitter акаунту
LinkedInUrl	nvarchar(200), null	-	-	URL LinkedIn акаунту
InstagramUrl	nvarchar(200), null	-	-	URL Instagram акаунту
YouTubeUrl	nvarchar(200), null	-	-	URL YouTube акаунту

Продовження таблиці 2.13

DiscordUrl	nvarchar(200), null	-	-	URL Discord акаунту
SlackUrl	nvarchar(200), null	-	-	URL Slack акаунту
TwitchUrl	nvarchar(200), null	-	-	URL Twitch акаунту
MediumUrl	nvarchar(200), null	-	-	URL Medium акаунту
TikTokUrl	nvarchar(200), null	-	-	URL TikTok акаунту
TelegramUrl	nvarchar(200), null	-	-	URL Telegram акаунту

Таблиця 2.14

Опис полів таблиці «CommunitySubscriptionForms»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер форми для заявки на участь у спільноті
CommunityId	int not null	-	Communities.Id	Ідентифікаційний номер спільноти
FormId	int not null	-	Forms.Id	Ідентифікаційний номер форми

Таблиця 2.15

Опис полів таблиці «EventImages»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер картинки доданої до події
FileName	nvarchar(200) not null	-	-	Назва файлу
EventId	int not null	-	Event.Id	Ідентифікаційний номер події

Опис полів таблиці «Events»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер спільноти
Name	nvarchar(200), not null	-	-	Назва
Description	nvarchar(2000), not null	-	-	Опис
StartDate	datetime2(7), not null	-	-	Дата початку
EndDate	datetime2(7), not null	-	-	Дата кінця
CommunityId	int not null	-	Communities.Id	Ідентифікаційний номер спільноти
OrganizerId	nvarchar(450), not null	-	-	Ідентифікаційний номер користувача
Attendance_Limit	int, null	-	-	Ліміт відвідувачів
Attendance_Should BeApproved	bit, not null	-	-	Значення, що означає, чи повинні відвідувачі події бути підтвердженні
Created	datetime2(7), not null	-	-	Дата, коли запис було додано до таблиці
CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис
Venue	nvarchar(max), not null	-	-	Місце проведення події (JSON)
IsCancelled	bit, not null	-	-	Значення, що означає, чи подія є відмінена

Таблиця 2.17

Опис полів таблиці «FormAnswers»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер відповідей на форму
FormId	int not null	-	Forms.Id	Ідентифікаційний номер форми
UserId	nvarchar(450), not null	-	-	Ідентифікаційний номер користувача
FieldAnswers	nvarchar(max), not null	-	-	Поля форми і відповіді на них (JSON)
Created	datetime2(7), not null	-	-	Дата, коли запис було додано до таблиці
CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис

Таблиця 2.18

Опис полів таблиці «Forms»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер форми
Name	nvarchar(200), not null	-	Forms.Id	Назва форми
Type	int, not null	-	-	Тип форми
Fields	nvarchar(max), not null	-	-	Поля форми (JSON)
Created	datetime2(7), not null	-	-	Дата, додавання запису до таблиці

Продовження таблиці 2.18

CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис

Таблиця 2.19

Опис полів таблиці «Sessions»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер доповіді
Title	nvarchar(200), not null	-	-	Назва доповіді
StartTime	datetime2(7), not null	-	-	Дата початку
Duration	time(7), not null	-	-	Тривалість, хв.
Description	nvarchar(1000), not null	-	-	Опис
EventId	int not null	-	Events.Id	Ідентифікаційний номер події
Created	datetime2(7), not null	-	-	Дата, коли запис було додано до таблиці
CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис

Таблиця 2.20

Опис полів таблиці «SessionSpeaker»

Назва	Тип даних	PK	FK	Опис
SessionsId	int not null	+	Sesions.Id	Ідентифікаційний номер доповіді
SpeakersId	int not null	+	Speakers.Id	Ідентифікаційний номер спікера

Таблиця 2.21

Опис полів таблиці «Speakers»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер спікера
Name	nvarchar(200), not null	-	-	Ім'я спікера
Title	nvarchar(200), not null	-	-	Посада
Company	nvarchar(200), not null	-	-	Компанія
Bio	nvarchar(1000), not null	-	-	Опис
EventId	int not null	-	Events.Id	Ідентифікаційний номер події
Created	datetime2(7), not null	-	-	Дата, коли запис було додано до таблиці
CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис

Опис полів таблиці «Subscriptions»

Назва	Тип даних	PK	FK	Опис
Id	int not null	+	-	Ідентифікаційний номер спікера
CommunityId	int not null	-	Communities.Id	Ідентифікаційний номер спільноти
UserId	nvarchar(450), not null	-	-	Ідентифікаційний номер користувача
Created	datetime2(7), not null	-	-	Дата, коли запис було додано до таблиці
CreatedBy	nvarchar(450), null	-	-	Ім'я користувача, який додав запис
LastModified	datetime2(7), null	-	-	Дата, коли запис було модифіковано
LastModifiedBy	nvarchar(450), null	-	-	Ім'я користувача, який модифікував запис

2.3 Проєктування та реалізація алгоритмів роботи системи

Для опису алгоритмів роботи та станів системи слід використати наочні засоби, такі як діаграми або схеми. Тому було складено декілька діаграм для можливих сценаріїв роботи системи.

Було складено діаграму активності реєстрації на подію (рис. 2.6). Вона складається із доріжок учасника і організатора. Спочатку учасник має здійснити пошук відповідної події для реєстрації. Якщо подія відмінена, або вже пройшла, або досягнуто ліміт учасників, або учасник є організатором цієї події, то подача заявки на участь у події є неможливою. В іншому випадку, учасник може це зробити. В залежності від налаштувань події, можливе додаткове підтвердження від організатора на участь у події.

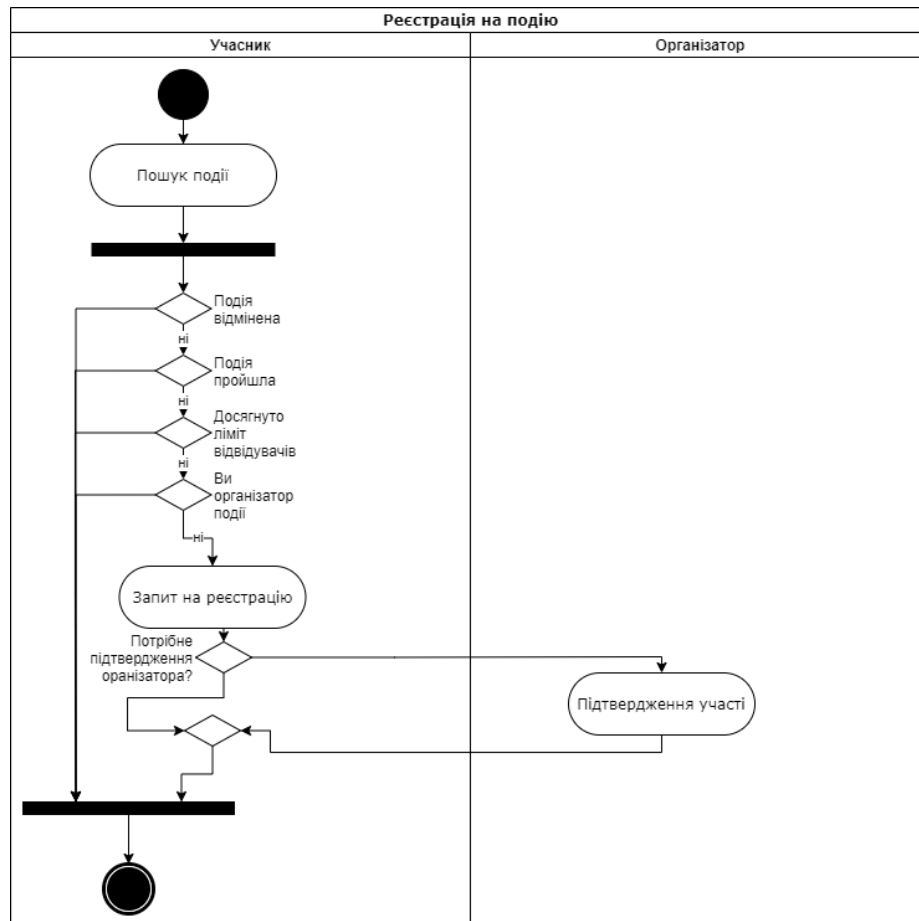


Рис. 2.6 – Діаграма активності реєстрації на подію

Для зображення алгоритму створення події було складено діаграму послідовності (рис. 2.7). При HTTP запиті на створення події, першим його обробляє контролер (життєва лінія `EventsController`). Він викликає метод `Send` у медіатора (життєва лінія `Mediator`) і передає йому аргументом об'єкт команди `CreateEventCommand`. В пайплайні медіатора викликається метод `Next`, який передає об'єкт команди валідатору (життєва лінія `CreateEventValidator`). Він в свою чергу валідує поля команди. Після цього, команда передається обробнику команди (`CreateEventCommandHandler`). Використовуючи дані про поточного користувача з аксесора (життєва лінія `IUserAccessor`) додає запис до бази даних, використовуючи методи `AddAsync` та `SaveChangesAsync` контекста бази даних (`IApplicationDbContext`). Після цього подія індексується для пошуку відповідним сервісом (життєва лінія `IEventSearchService`). Як результат, дію передано назад контролеру, який відправляє відповідний HTTP статус код.

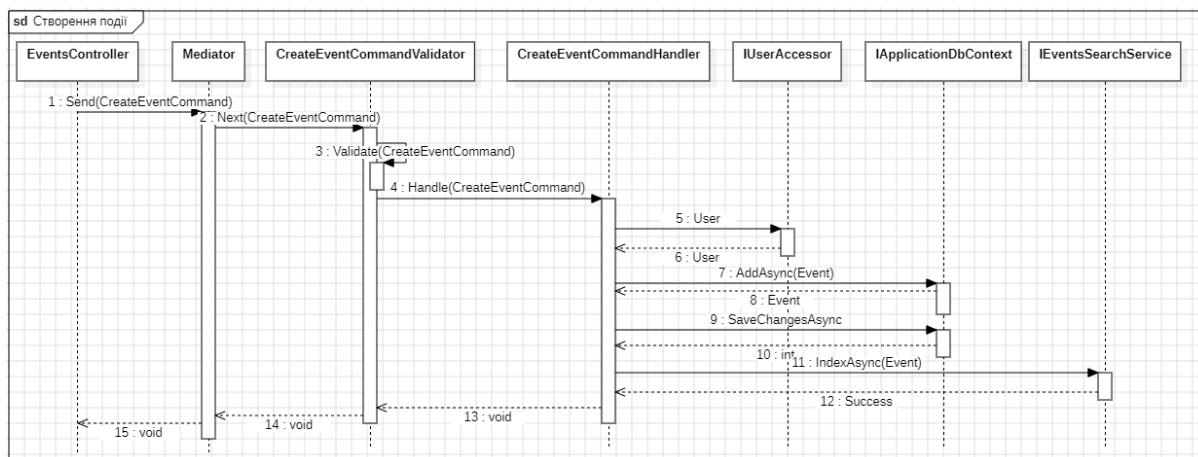


Рис. 2.7 – Діаграма послідовності додавання події

2.4. Реалізація функціоналу системи організації спільнот та подій

Розгляньмо створений функціонал для додавання нової спільноти. Наведено відповідний лістинг методу класу контролера (рис. 2.8). Він є обробником POST HTTP запиту на стороні серверу.

```
[HttpPost]
public async Task<CreateCommunityResultDto> CreateCommunityAsync(CreateCommunityCommand command)
{
    return await Mediator.Send(command);
}
```

Рис. 2.8 – Метод додавання спільноти класу CommunitiesController

Команда CreateCommunityCommand це рекорд, який включає в собі властивості, необхідні для створення спільноти (рис. 2.9). Він реалізує інтерфейс IRequest<CreateCommunityCommandResultDto>, який маркує цей рекорд як команду і описує тип результату виконання команди. Кастомний атрибут Authorize означає, що користувач має бути аутентифікованим у системі, щоб виконувати цю команду.

```
[Authorize]
public sealed record CreateCommunityCommand(
    string Name,
    string Location,
    string Domain) : IRequest<CreateCommunityResultDto>;
```

Рис. 2.9 – Рекорд команди створення спільноти

Використовуючи бібліотеку MediatR, команду створення передано зконфігурованим пайплайнам, класам, які реалізують інтерфейс IPipelineBehavior, в тому числі тому, який викликає валідатора. Нижче наведено клас валідатор команди створення спільноти (рис. 2.10).

```

public sealed class CreateCommunityCommandValidator
    : AbstractValidator<CreateCommunityCommand>
{
    public CreateCommunityCommandValidator()
    {
        RuleFor(x => x.Name)
            .NotEmpty().WithMessage("Назва обов'язкова.")
            .MaxLength(100).WithMessage("Назва не може бути більше 100 символів.");

        RuleFor(x => x.Domain)
            .NotEmpty().WithMessage("Домен обов'язковий.")
            .MaxLength(100).WithMessage("Домен не може бути більше 100 символів.");

        RuleFor(x => x.Location)
            .NotEmpty().WithMessage("Локація обов'язкова.")
            .MaxLength(100).WithMessage("Локація не може бути більше 100 символів.");
    }
}

```

Рис. 2.10 – Валідатор команди додавання спільноти

Для реалізації валідатора використано бібліотеку FluentValidator. З її допомогою описано правила валідації кожної властивості команди.

Далі команда потрапляє в обробник цієї команди CreateCommunityCommandHandler (рис. 2.11)

```

internal sealed class CreateCommunityCommandHandler(
    IApplicationDbContext context,
    ICurrentUserAccessor currentUserAccessor,
    ICommunitiesSearchService searchService)
    : IRequestHandler<CreateCommunityCommand, CreateCommunityResultDto>
{
    private readonly IApplicationDbContext _context = context;
    private readonly ICurrentUserAccessor _currentUserAccessor = currentUserAccessor;
    private readonly ICommunitiesSearchService _searchService = searchService;

    public async Task<CreateCommunityResultDto> Handle(
        CreateCommunityCommand request,
        CancellationToken cancellationToken)
    {
        var community = await SaveCommunityAsync(request, cancellationToken);
        var id = community.Id;

        await AddOrganizerSubscriptionAsync(id, cancellationToken);
        await AddCommunitySubscriptionFormAsync(id, cancellationToken);
        await AddEmptySocialMediaAsync(id, cancellationToken);
        await _context.SaveChangesAsync(cancellationToken);

        await IndexCommunityAsync(community, cancellationToken);

        return new CreateCommunityResultDto(id);
    }
}

```

Рис. 2.11 – Обробник команди додавання спільноти

Використовуючи вбудований в ASP.NET Core контейнер впровадження залежностей, в обробник було впроваджено сервіси контексту бази даних (інтерфейс `IApplicationDbContext`), отримання поточного аутентифікованого користувача (інтерфейс `ICurrentUserAccessor`) та сервіс пов'язаний із пошуком (інтерфейс `ICommunitiesSearchService`).

В обробнику для збереження інформації використано патерн Unit of work. Це можна визначити по використанню методу `SaveChangesAsync` сервісу контексту бази даних.

Слід детальніше зупинитися на процесі індексації спільноти. Для комунікації із Elasticsearch використано бібліотеку `Elastic.Clients.Elasticsearch` (рис. 2.12).

```
public async Task<bool> IndexAsync(CommunityIndexDocument community, CancellationToken cancellationToken = default)
{
    var document = community.ToDocument();

    await CreateCommunityIndexIfNotExistsAsync(cancellationToken);
    var result = await _client.IndexAsync(document, (IndexName)_options.Indices.Community, cancellationToken);
    return result.IsSuccess();
}
```

Рис. 2.12 – Метод індексації спільноти

На основі об'єкту `CommunityIndexDocument`, з усіма необхідними для індексації властивостями, створено об'єкт рекорду `CommunityDocument` (рис. 2.13). Він має структуру документа, який вже індексовано Elasticsearch.

```
public sealed record CommunityDocument(
    int Id,
    string Name,
    string NameSuggestion,
    string? Description,
    string Location,
    string Domain,
    int SubscribersCount);
```

Рис. 2.13 – Документ спільноти для індексації

Якщо індекс не було створено раніше, він створюється в методі індексації (рис. 2.14). Для створення індексу використовується явний мапінг, тобто типи усіх полів описано явно в конфігурації.

```

private async Task<bool> CreateCommunityIndexIfNotExistsAsync(CancellationToken cancellationToken = default)
{
    var indexExistsResponse = await _client.Indices.ExistsAsync(_options.Indices.Community, cancellationToken);
    if (indexExistsResponse.Exists)
        return true;

    var createIndexResponse = await _client.Indices.CreateAsync(_options.Indices.Community, config =>
    {
        config.Mappings(m => m
            .Properties<CommunityDocument>(p => p
                .SearchAsYouType(d => d.NameSuggestion)
                .Keyword(d => d.Name)
                .Keyword(d => d.Domain)
                .Text(d => d.Description)
                .Keyword(d => d.Location)));
    }, cancellationToken);

    return createIndexResponse.IsSuccess();
}

```

Рис. 2.14 – Створення індексу спільнот

Розгляньмо функціонал відміни події. Наведено обробник цієї команди (рис. 2.15). Перевіривши, чи поточний користувач є організатором цієї події, чи подія вже почалася або пройшла і чи вона вже є відміненою, значення про відміну події збережено до бази даних. Також використано механізм доменної події (domain event) `EventCancelledEvent`, щоб повідомити іншим модулям, що подію було відмінено.

```

[Authorize]
public sealed record CancelEventCommand(int EventId) : IRequest;

internal sealed class CancelEventCommandHandler(
    IApplicationDbContext context,
    ICurrentUserAccessor currentUserAccessor) : IRequestHandler<CancelEventCommand>
{
    private readonly IApplicationDbContext _context = context;
    private readonly ICurrentUserAccessor _currentUserAccessor = currentUserAccessor;

    public async Task Handle(CancelEventCommand request, CancellationToken cancellationToken)
    {
        var @event = await _context.Events
            .FirstOrDefaultAsync(e => e.Id == request.EventId
                && e.OrganizerId == _currentUserAccessor.UserId, cancellationToken)
            ?? throw new NotFoundException(nameof(Event), request.EventId);

        if (@event.StartDate < DateTime.UtcNow)
            throw new InvalidRequestException(nameof(request.EventId), "Cannot cancel event that has already started");

        if (@event.IsCancelled)
            throw new InvalidRequestException(nameof(request.EventId), "Event is already canceled");

        @event.IsCancelled = true;
        @event.DomainEvents.Add(new EventCancelledEvent(@event.Id));
        await _context.SaveChangesAsync(cancellationToken);
    }
}

```

Рис. 2.15 – Створення індексу спільнот

В обробнику події `EventCancelledEvent` користувачам, які є відвідувачами події, або присутні в списку очікування, надсилається лист-повідомлення про те, що подію було скасовано (рис. 2.16).

```

internal sealed class EventCancelledEventHandler(
    IApplicationDbContext applicationDbContext,
    IUserService userService,
    IMailService mailService)
    : INotificationHandler<DomainEventNotification<EventCancelledEvent>>
{
    private readonly IApplicationDbContext _applicationDbContext = applicationDbContext;
    private readonly IUserService _userService = userService;
    private readonly IMailService _mailService = mailService;

    public async Task Handle(
        DomainEventNotification<EventCancelledEvent> notification,
        CancellationTokens cancellationTokens)
    {
        var eventId = notification.DomainEvent.EventId;
        var attendees = await _applicationDbContext.Attendees
            .Where(a => a.EventId == eventId && (
                a.Status == AttendeeStatus.Pending
                || a.Status == AttendeeStatus.Confirmed))
            .ToListAsync(cancellationTokens);

        var @event = await _applicationDbContext.Events
            .Include(e => e.Community)
            .SingleOrDefault(e => e.Id == eventId, cancellationTokens);

        var attendeesIds = attendees.Select(a => a.UserId).ToList();
        var users = await _userService.GetUsersByIdListAsync(attendeesIds, cancellationTokens);

        var tasks = users.Select(u => _mailService.SendEventCancelledMailAsync(
            u.Email,
            new EventCancelledMailTemplateModel(@event.Community.Name, @event.Name),
            cancellationTokens)).ToList();

        await Task.WhenAll(tasks);
    }
}

```

Рис. 2.16 – Обробник події скасування події

Розгляньмо функціонал редагування форми для подачі заявки на вступ у спільноту. Для реалізації додавання полів різних типів (короткий текст, довгий текст, одна обрана відповідь, декілька обраних відповідей) було створено ієрархію класів на рівні доменних сутностей. Так як для комунікації із клієнтським додатком використовується REST API, тип даних для відправки даних формату JSON. JSON не має механізму ієрархії об'єктів і розрізнення дочірніх, а ASP.NET Core контролери приймають строго типізовані об'єкти. Для досягнення такої мети у форматі JSON може слугувати тип даних tagged union (discriminated union, sum type), проте він не підтримується мовою C#. Тому було прийнято рішення створити кастомний JSON парсер для поля форми, отримуючи JSON формату tagged union і перетворюючи його на об'єкт одного з дочірніх класів поля форми (рис. 2.17).

```

public class CommunitySubscriptionFormFieldDtoJsonConverter : JsonConverter<CommunitySubscriptionFormFieldDto>
{
    public override CommunitySubscriptionFormFieldDto? Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)
    {
        Utf8JsonReader readerClone = reader;

        if (readerClone.TokenType != JsonTokenType.StartObject)
        {
            throw new JsonException();
        }

        while (readerClone.Read())
        {
            if (readerClone.TokenType == JsonTokenType.EndObject)
            {
                throw new JsonException();
            }

            if (readerClone.TokenType == JsonTokenType.PropertyName)
            {
                var propertyName = readerClone.GetString() ?? throw new JsonException();
                if (propertyName.Equals(nameof(CommunitySubscriptionFormFieldDto.Type), StringComparison.OrdinalIgnoreCase))
                {
                    readerClone.Read();
                    var type = readerClone.GetString();
                    return type switch
                    {
                        FormFieldNames.ShortText => JsonSerializer.Deserialize<ShortTextFormFieldDto>(ref reader, options),
                        FormFieldNames.LongText => JsonSerializer.Deserialize<LongTextFormFieldDto>(ref reader, options),
                        FormFieldNames.SingleOption => JsonSerializer.Deserialize<SingleOptionFormFieldDto>(ref reader, options),
                        FormFieldNames.MultipleOptions => JsonSerializer.Deserialize<MultipleOptionsFormFieldDto>(ref reader, options),
                        _ => throw new NotSupportedException($"Field type '{type}' is not supported."),
                    };
                }
            }
        }

        throw new JsonException();
    }
}

```

Рис. 2.17 – Кастомний JSON парсер для поля форми

Всі поля форми зберігаються в одному полі таблиці бази даних у форматі JSON, Entity Framework Core підтримує це. Проте постало схоже питання як зберігати об'єкти дочірніх класів у форматі JSON. Entity Framework має функціонал tagged union, тому було обрано його для вирішення проблеми. До базового класу поля додано атрибуту JsonDerivedType для позначення дочірніх класів та значення поля, яке відповідатиме за клас, в який має десеріалізуватися об'єкт (рис. 2.18).

```

[JsonDerivedType(typeof(ShortTextFormField), typeDiscriminator: FormFieldNames.ShortText)]
[JsonDerivedType(typeof(LongTextFormField), typeDiscriminator: FormFieldNames.LongText)]
[JsonDerivedType(typeof(SingleOptionFormField), typeDiscriminator: FormFieldNames.SingleOption)]
[JsonDerivedType(typeof(MultipleOptionsFormField), typeDiscriminator: FormFieldNames.MultipleOptions)]
public abstract class FormFieldBase
{
    public abstract string Type { get; }

    public string Name { get; set; } = null!;

    public string? Description { get; set; }

    public bool IsRequired { get; set; }

    public int Order { get; set; }

    public abstract bool Validate(object value);
}

```

Рис. 2.18 – Базовий клас поля форми

Також для того щоб конфігурувати поле таблиці як текстове у форматі JSON, було використано метод HasConversion у вбудованому в Entity Framework Core класі EntityTypeBuilder (рис. 2.19).

```
var fieldSerializerOptions = new JsonSerializerOptions();
builder.Property(e => e.Fields)
    .HasConversion(
        v => JsonSerializer.Serialize(v, fieldSerializerOptions),
        v => JsonSerializer.Deserialize<List<FormFieldBase>>(v, fieldSerializerOptions)!);
```

Рис. 2.19 – Конфігурація поля таблиці для збереження полів форми

Логування в додатку виконано за допомогою вбудованого логера в ASP.NET Core та бібліотеки Serilog з модулями Serilog.AspNetCore, Serilog.Enrichers.Environments, Serilog.Exceptions, Serilog.Sinks.Elasticsearch. За її допомогою налаштовано логування в консоль та структуроване логування в Elasticsearch (рис. 2.20).

```
builder.Host.UseSerilog((context, configuration)
=> configuration.Enrich.FromLogContext()
    .Enrich.WithExceptionDetails()
    .Enrich.WithMachineName()
    .WriteTo.Console()
    .WriteTo.Elasticsearch(new ElasticsearchSinkOptions(new Uri(context.Configuration["Elastic:Uri"]!))
    {
        IndexFormat = $"{context.Configuration["ApplicationName"]}-logs-{context.HostingEnvironment.EnvironmentName?.ToLower().Replace(".", "-")}-{DateTime.UtcNow:yyyy-MM}",
        AutoRegisterTemplate = true,
        NumberOfShards = 2,
        NumberOfReplicas = 1,
    })
    .Enrich.WithProperty("Environment", context.HostingEnvironment.EnvironmentName!)
    .ReadFrom.Configuration(context.Configuration));
```

Рис. 2.20 – Конфігурація логування

Лог можна переглянути за допомогою інструмента Kibana (рис. 2.21).

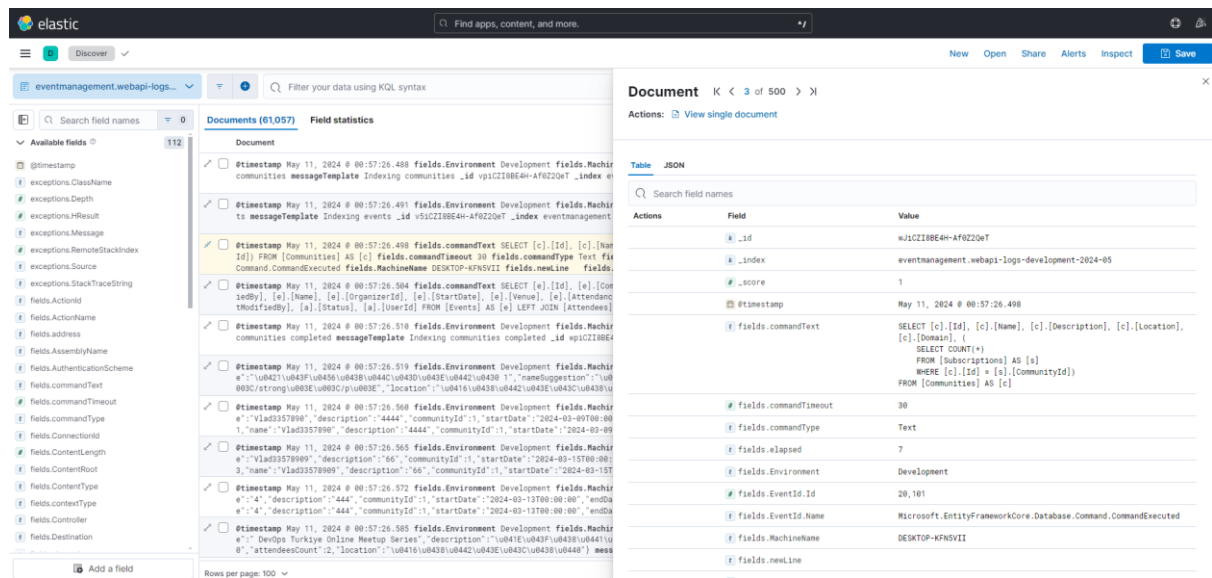


Рис. 2.21 – Інтерфейс Kibana із відкритим логом

Висновки до другого розділу

В результаті проєктування системи для управління спільнотами та подіями було визначено та описано варіанти використання, вимоги до користувачів і системи. Також було створено об'єктно-орієнтовану структуру системи, складено діаграму класів доменних сутностей, виокремлено декілька патернів проєктування.

Було розроблено базу даних додатку на основі реляційної системи управління базами даних відповідно до описаних вимог.

Було спроєктовано алгоритми системи, складено декілька діаграм для їх наочного зображення.

Як результат, було реалізовано функціонал додатку, описано декілька модулів системи, наведено лістинги окремих методів та класів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ ІЗ СИСТЕМОЮ ДЛЯ УПРАВЛІННЯ СПІЛЬНОТАМИ ТА ПОДІЯМИ

3.1. Порядок встановлення та налаштування параметрів платформи

Встановлення та налаштування параметрів системи вимагає встановлення серверного середовища, розгортання серверної та клієнтської частини, а також налаштування зв'язків між компонентами системи.

Для опису процесу розгортання було використано діаграму розгортання (рис. 3.1).

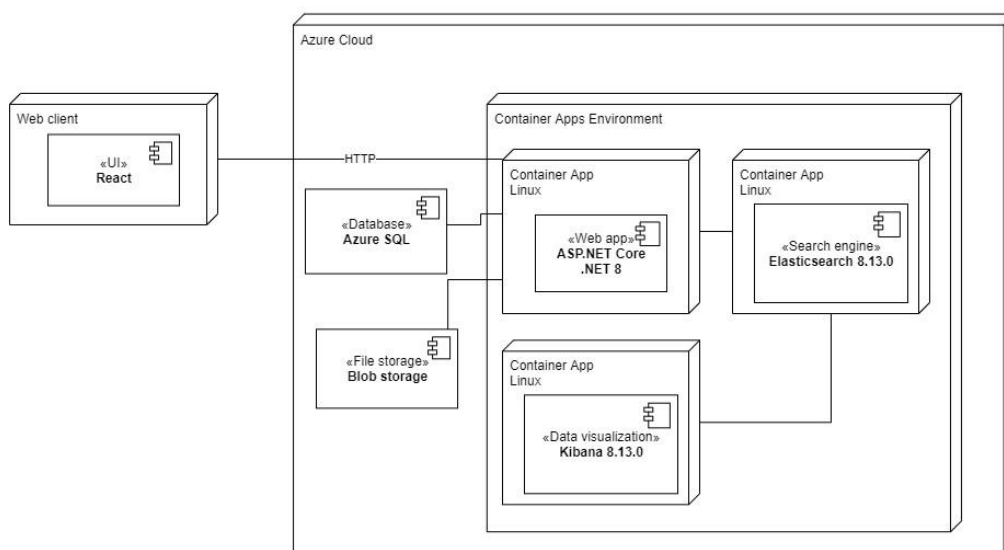


Рис. 3.1 – Діаграма розгортання

Для розгортання використовується браузер, та провайдер хмарних послуг Azure:

- в браузері виконується клієнтський код, написаний із використанням бібліотеки React;
- в хмарі Azure використано сервіси для розгортання бази даних Azure SQL та файлів Blob storage. Docker контейнери з вебдодатком, пошуковим двигуном Elasticsearch та панеллю для візуалізації даних Kibana розгорнуто в окремих Container apps в межах єдиної мережі Container Apps Environment.

Розгортання контейнерів у середовищі Azure Container Apps Environment забезпечує ізолюваність, безпеку та простоту управління різними компонентами додатку.

Цей підхід дозволяє легко масштабувати окремі компоненти в залежності від навантаження, забезпечуючи при цьому високу доступність та продуктивність системи. Крім того, використання Docker контейнерів спрощує процес оновлення та розгортання нових версій сервісів, що підвищує гнучкість та ефективність управління додатком.

Таким чином, поєднання цих технологій та сервісів дозволяє створити надійну, масштабовану та керовану інфраструктуру для системи для управління спільнотами та подіями.

3.2. Структура інтерфейсу системи для управління спільнотами та подіями

Робота із додатком відбувається через вебінтерфейс. Зайшовши на вебсайт, користувача зустрічає сторінка входу до облікового запису (рис. 3.2).

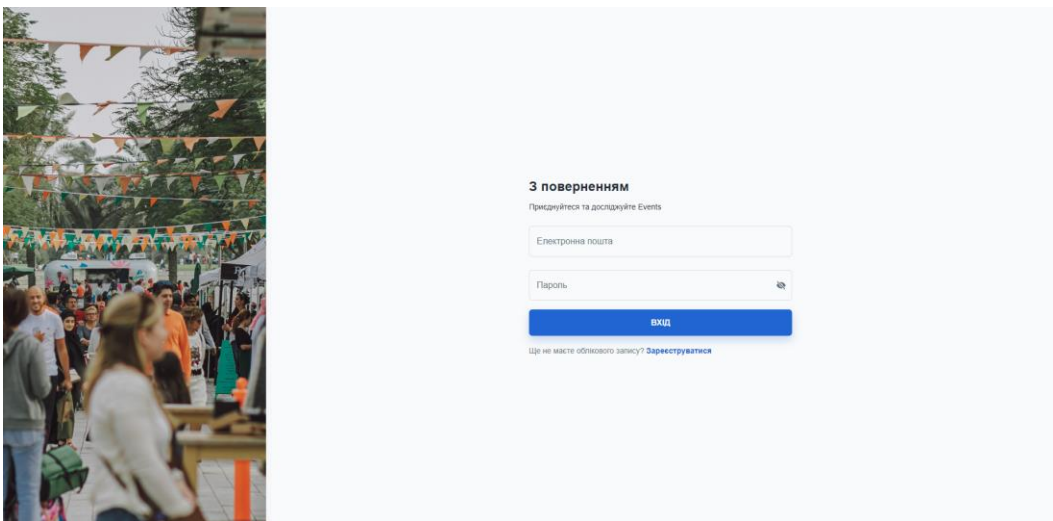


Рис. 3.2 – Сторінка входу до облікового запису

Якщо у користувача немає облікового запису, він може перейти на сторінку реєстрації (рис. 3.3) по відповідному посиланню.

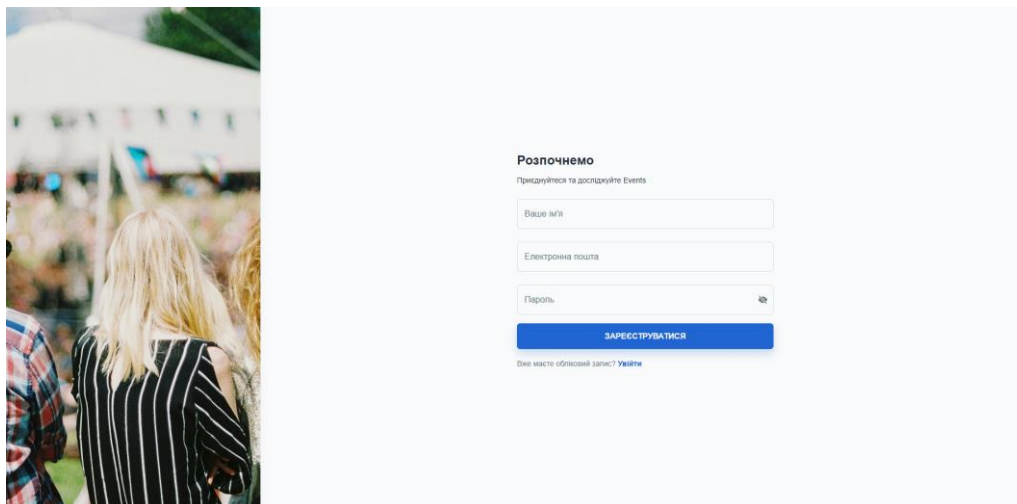


Рис. 3.3 – Сторінка реєстрації

Ввійшовши до свого облікового запису, відбувається автоматичний перехід на сторінку «Мої спільноти» (рис. 3.4). На ній представлено список спільнот, у яких він є учасником або організатором.

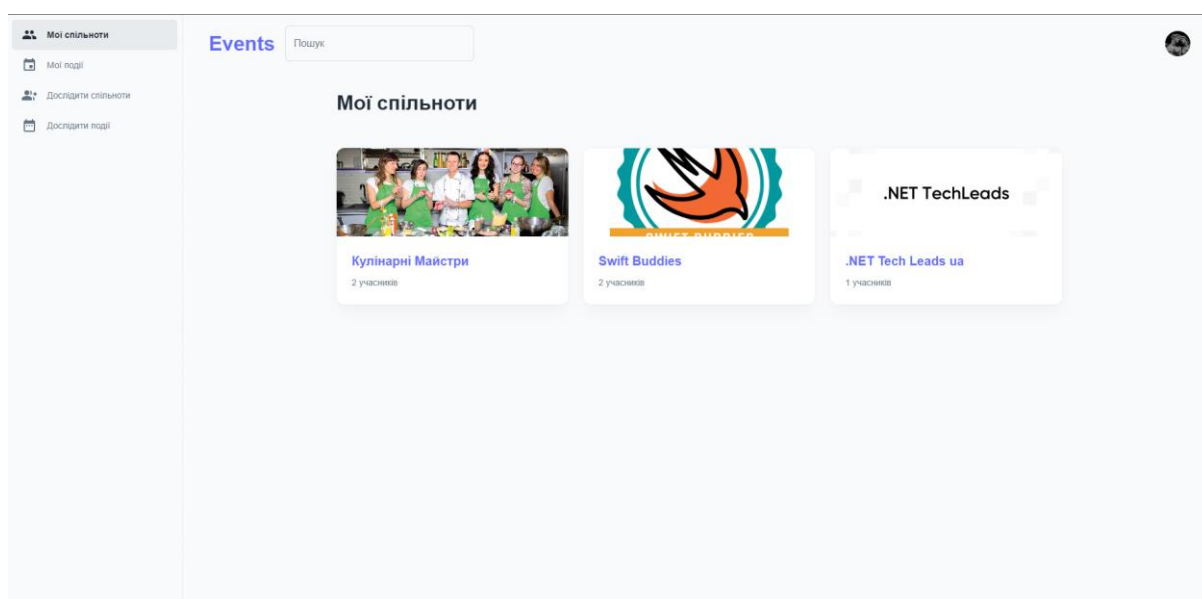


Рис. 3.4 – Сторінка реєстрації

Бокове меню складається із пунктів: «Мої спільноти», «Мої події», «Дослідити спільноти», «Дослідити події».

У правому верхньому куті знаходиться картинка профілю користувача. Натиснувши на неї, з'явиться випадаючий список із додатковими даними про користувача, посиланнями та кнопкою виходу із облікового запису (рис. 3.5).

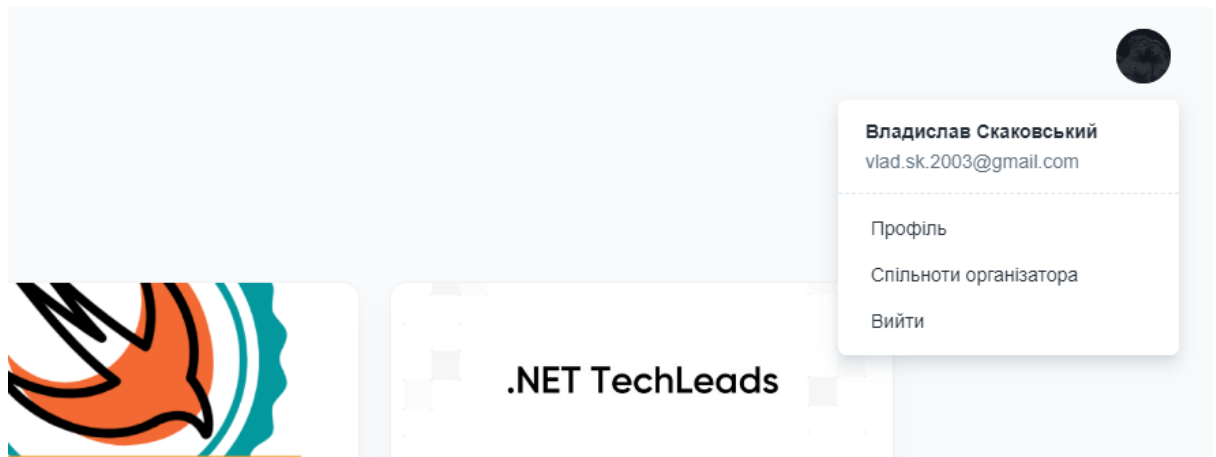


Рис. 3.5 – Випадаючий список

Натиснувши на елемент «Профіль», відбувається перехід на сторінку його профіля (рис. 3.6). Тут представлено інформацію, надану про себе, користувачем. Натиснувши на кнопку «Редагувати профіль», відбудеться перехід на відповідну сторінку (рис. 3.7).

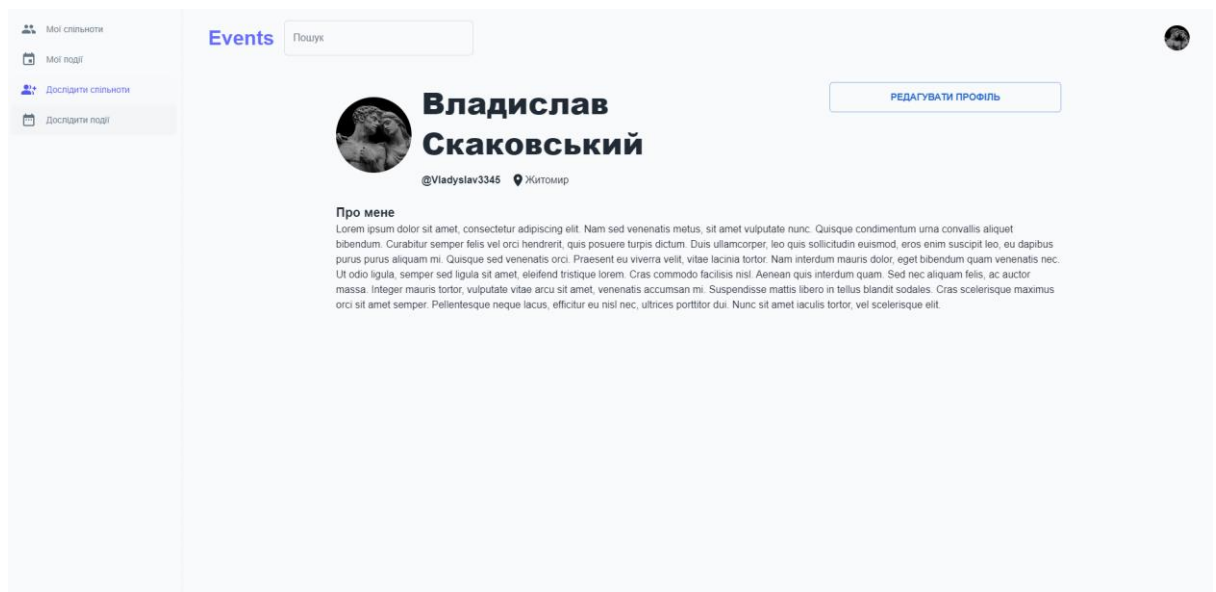


Рис. 3.6 – Профіль користувача

Інформація

Events

Пошук

Інформація

ЗАВАНТАЖИТИ КАРТИНКУ

Електронна пошта

vlad.sk.2003@gmail.com

Ім'я *

Владислав Скаковський

Пов'яз *

Vladyslav3345

Ваша локація

Житомир

Інформація

>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam sed venenatis metus, sit amet vulputate nunc. Quisque condimentum urna convallis aliquet bibendum. Curabitur semper felis vel orci hendrerit, quis posuere turpis dictum. Duis ullamcorper, leo quis sollicitudin eusmod, eros enim suscipit leo, eu dapibus purus purus aliquam mi. Quisque sed venenatis orci. Praesent eu viverra velit, vitae lacinia tortor. Nam interdum mauris dolor, eget bibendum quam venenatis nec. Ut odio ligula, semper sed ligula sit amet, eleifend tristique lorem. Cras commodo facilisis nisl. Aenean quis interdum quam. Sed nec aliquam felis, ac auctor massa. Integer mauris tortor, vulputate vitae arcu sit amet, venenatis accumsan mi. Suspendisse mattis libero in tellus blandit sodales. Cras

ЗБЕРЕГТИ ЗМІНИ

Рис. 3.7 – Редагування профіля користувача

На сторінці «Мої події» відображено список подій, у яких користувач є учасником, включаючи ті, де він є організатором. Список поділено на майбутні (рис. 3.8) та минулі події (рис. 3.9).

Мої спільноти

Мої події

Дослідити спільноти

Дослідити події

Events

Пошук

Мої події

МАЙБУТНІ

МИНУЛІ

30 травня 2024 р.

Event Test

00:05 Від Кулікарні Майстри

Онлайн

1 учасників йде

SwiftBuddies Nisan 2024

12:00 Від Swift Buddies

Онлайн

1 учасників йде

3 червня 2024 р.

SwiftBuddies Ocak 2024

12:00 Від Swift Buddies

Онлайн

Рис. 3.8 – Мої події - майбутні події

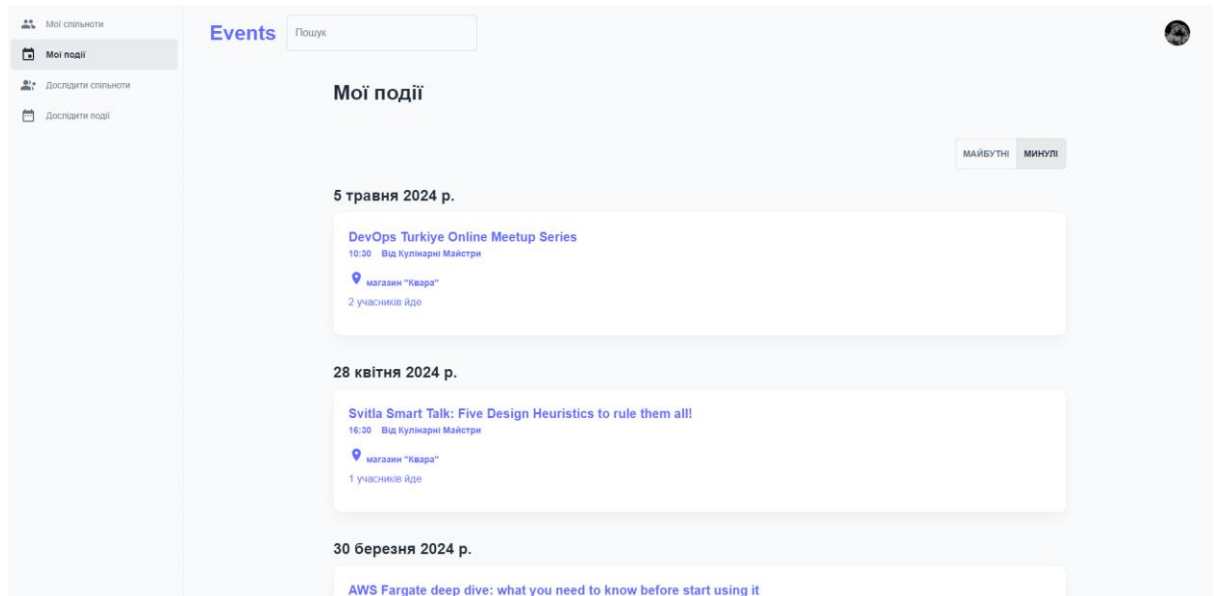


Рис. 3.9 – Мої події - минулі події

На сторінці «Дослідити спільноти» у користувача є можливість виконувати фільтрацію по усіх спільнотах у системі (рис. 3.10). Для цього використовуються фільтри по локації, домену та сортування по замовчуванню, назвою та кількістю учасників.

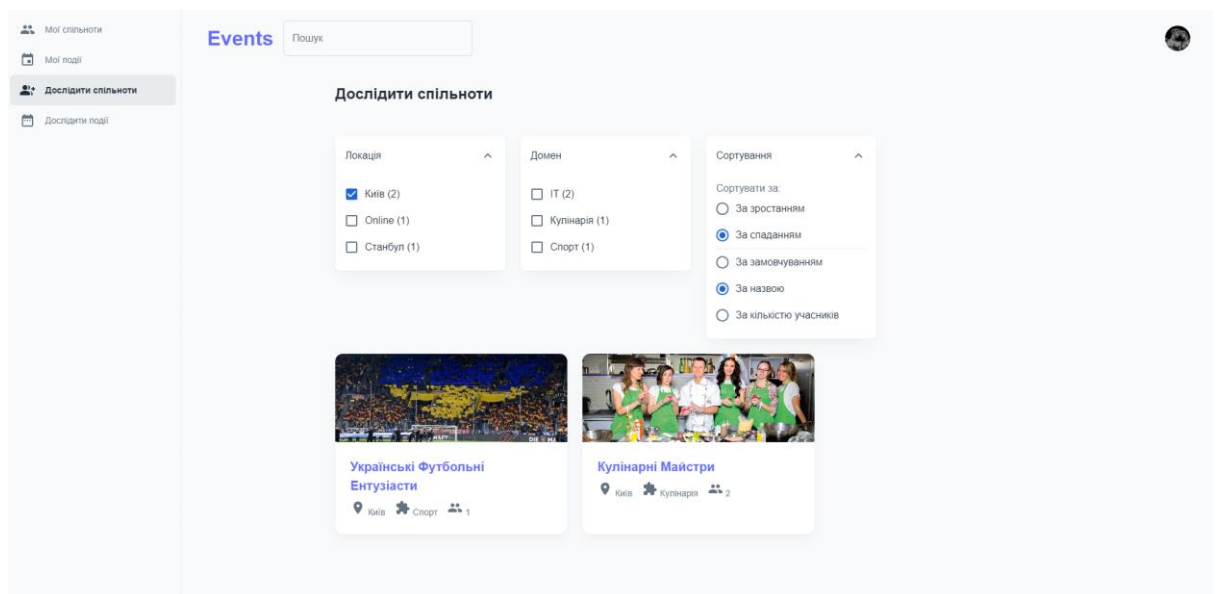


Рис. 3.10 – Дослідити спільноти

На сторінці «Дослідити події», у користувача є можливість виконати фільтрацію усіх подій у системі (рис. 3.11). Для цього є фільтри по локації та даті початку та сортування за датою початку і кількістю учасників.

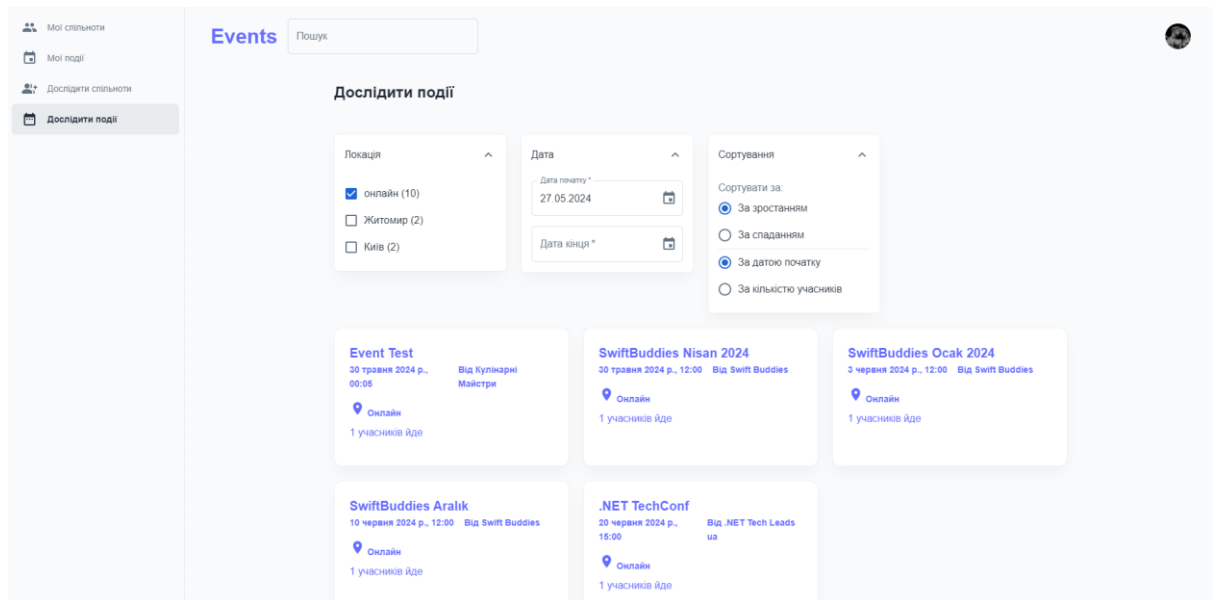


Рис. 3.11 – Дослідити події

Також для виконання пошуку спільнот та подій, в хедері сайту є текстове поле (рис. 3.12).

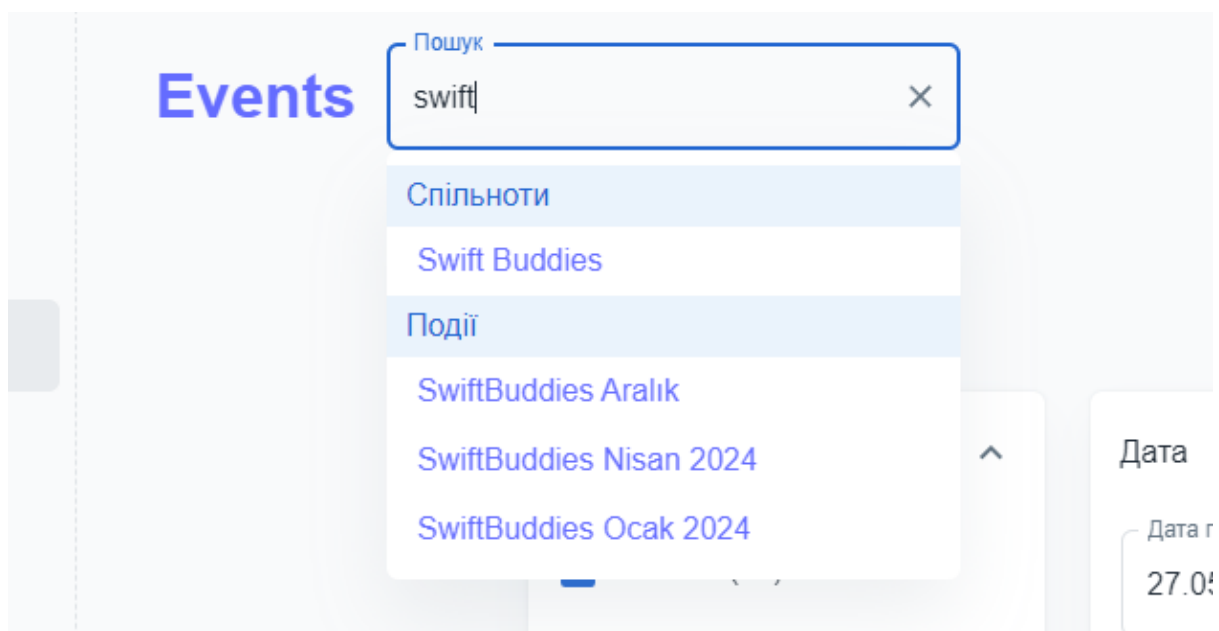


Рис. 3.12 – Поле для пошуку

У кожної спільноти є власна сторінка (рис. 3.13 та рис. 3.14). На ній розміщено інформацію про спільноту, посилання на її ресурси, опис та події - майбутні та минулі. Для користувача, який не є організатором спільноти є можливість стати її учасником.

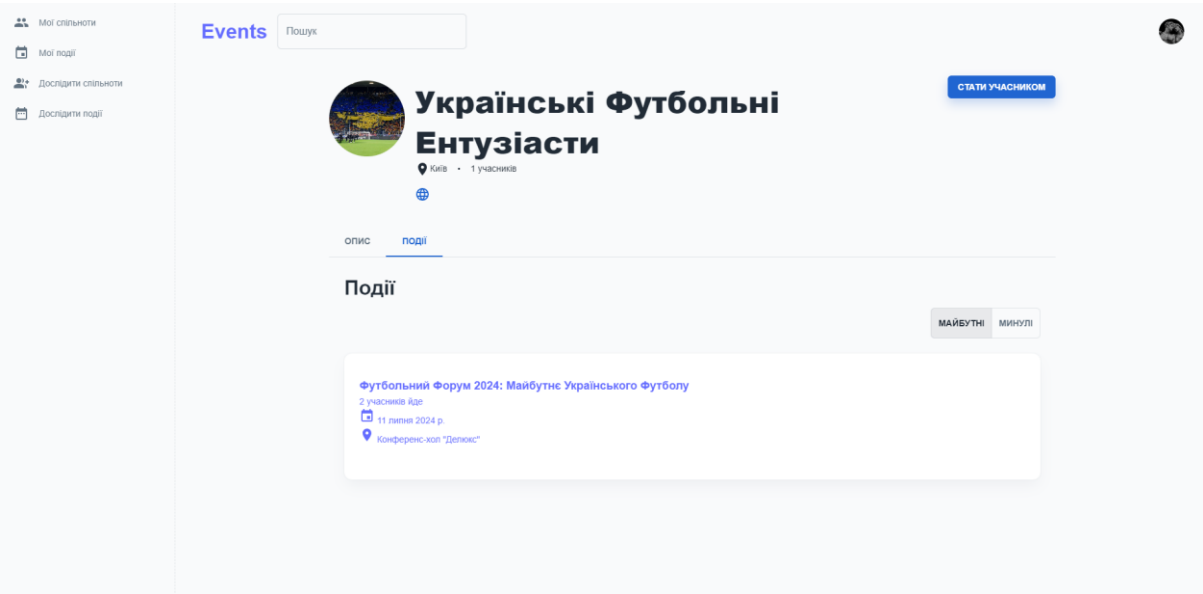


Рис. 3.13 – Спільнота та її події

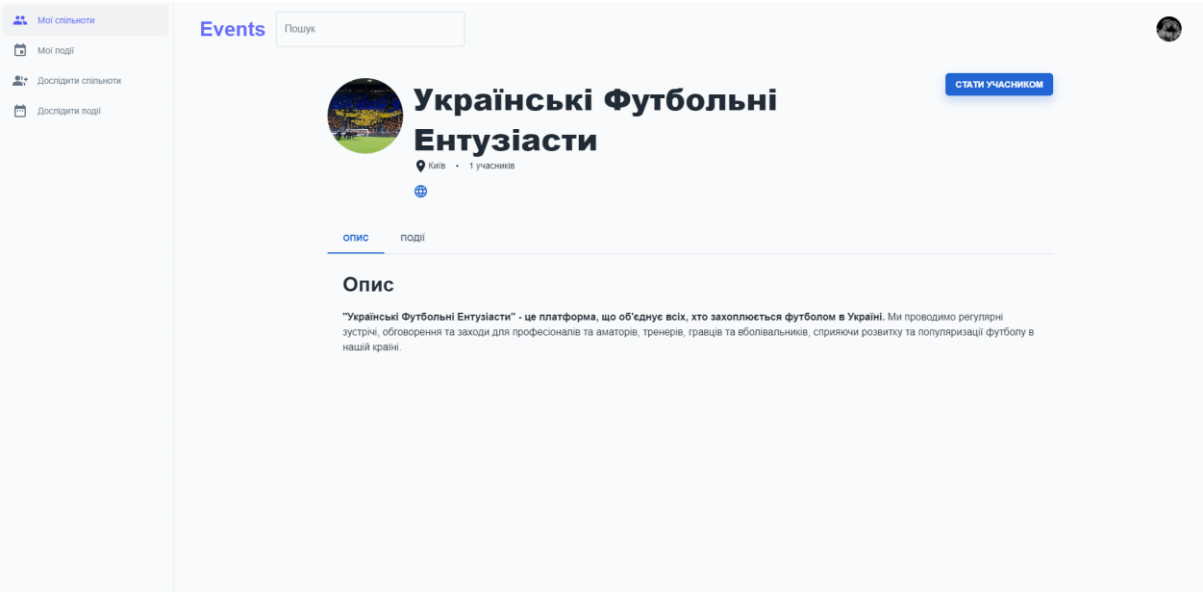


Рис. 3.14 – Спільнота та її опис

Також для кожної події є її сторінка. На представлено детальну інформацію про неї, місце та дату, програму, картинки та спікерів. Для користувачів, які не є організаторами події, є можливість стати її учасником (рис. 3.15 – 3.17).

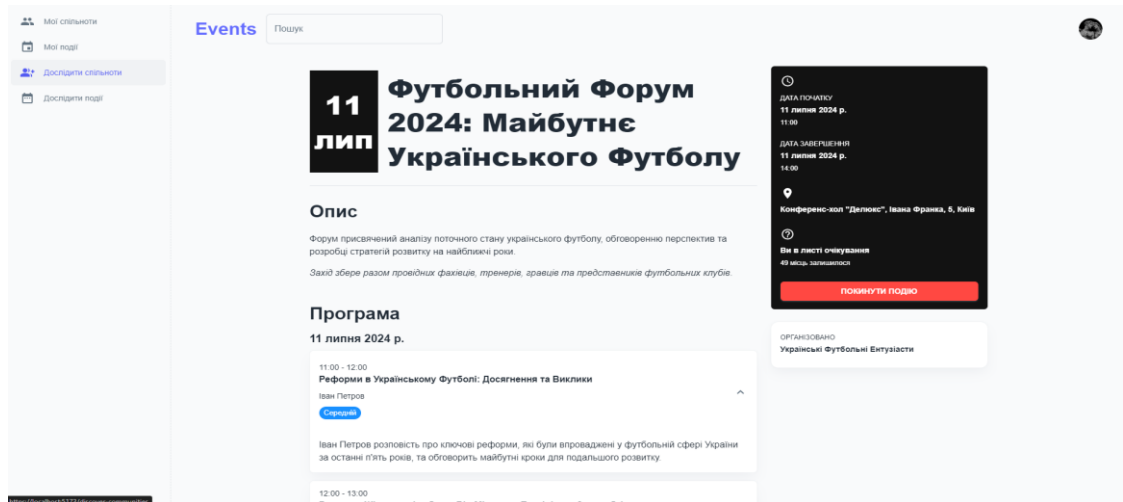


Рис. 3.15 – Сторінка події - опис та програма

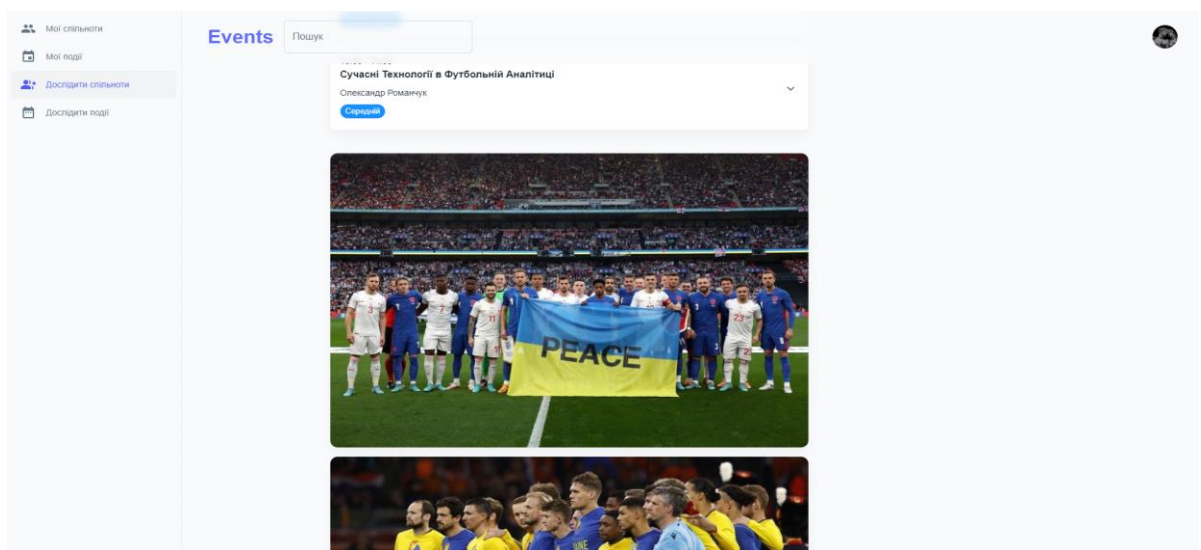


Рис. 3.16 – Сторінка події – картинки

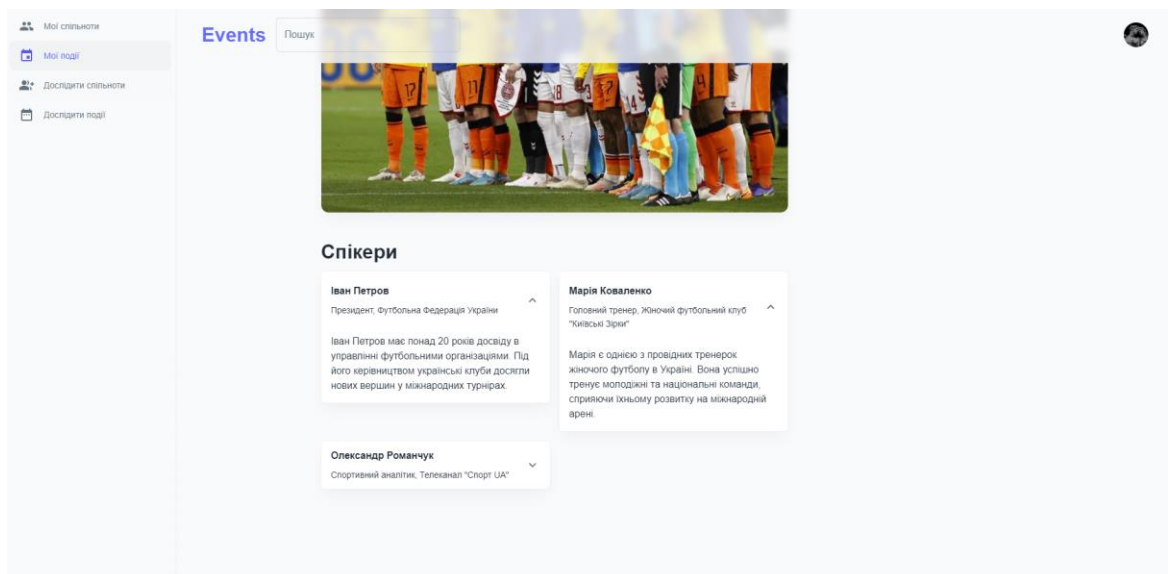


Рис. 3.17 – Сторінка події – спікери

Розглянемо функціонал панелі організатора. Щоб до нього перейти потрібно обрати «Спільноти організатора» у випадаючому списку облікового запису. Натиснувши на елемент меню, відбувається перехід на сторінку вибору спільноти (рис. 3.18). Тут перелічені всі спільноти, в яких користувач є організатором. Також є можливість створення нової спільноти.

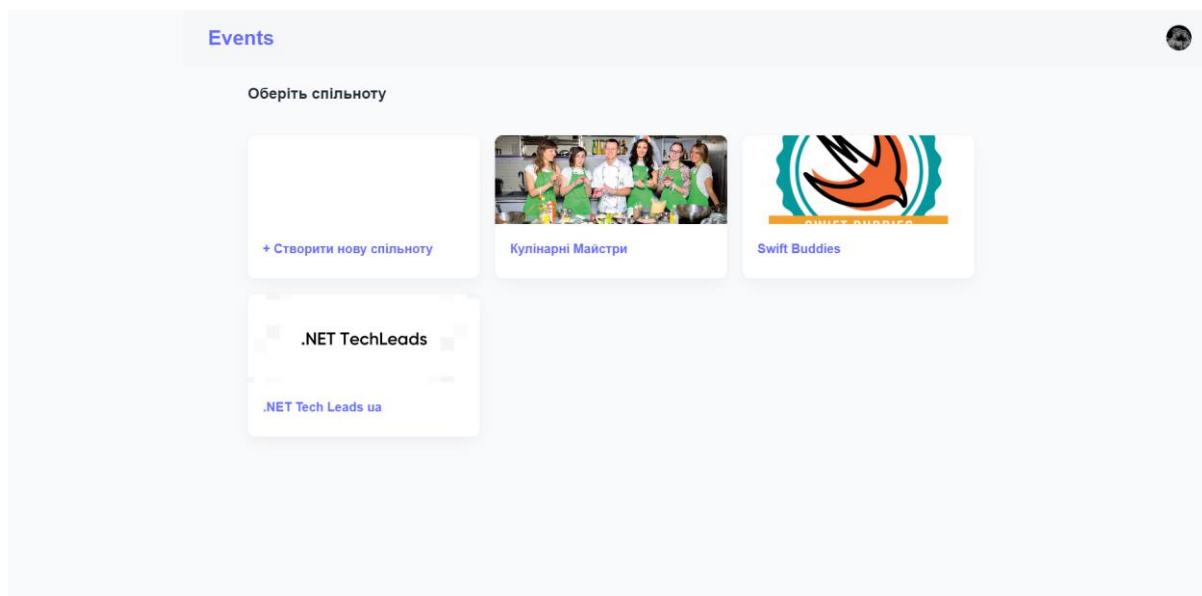


Рис. 3.18 – Сторінка вибору спільноти організатора

Натиснувши на блок створення спільноти, користувач попадає на відповідну форму (рис. 3.19). Заповнивши поля (рис. 3.20) та відправивши форму, відбувається перехід на сторінку дашборду (рис. 3.21).

Рис. 3.19 – Створення спільноти

Створити спільноту

Назва

ТехноПізнавачі України

Локація

Житомир

Домен

Tech

СТВОРИТИ

Рис. 3.20 – Заповнена форма створення спільноти

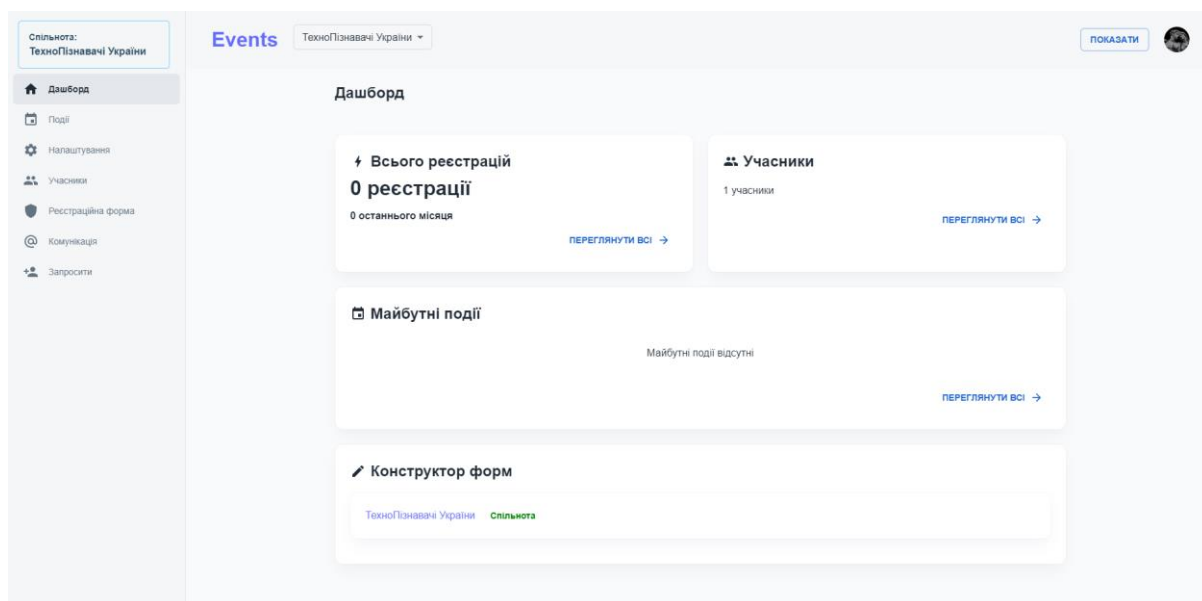


Рис. 3.21 – Дашбордин

На сторінці дашборду показано наступні віджети:

- віджет реєстрації, що містить кількість реєстрацій на події спільноти за останні тридцять днів. Посилання веде на сторінку подій спільноти;
- віджет учасників, що відображає кількість учасників спільноти. Посилання веде на сторінку учасників;
- віджет майбутніх подій, в якому представлено п'ять найближчих подій. Посилання веде на сторінку подій спільноти;
- віджет конструктора форм зі список форм опитування спільноти. Наявна лише одна форма - реєстраційна форма спільноти.

На сторінці «Налаштування», користувач може додати більше інформації про спільноту, додати посилання та видалити спільноту (рис. 3.22 – 3.24). Заповнимо цю інформацію.

Спільнота:
ТехноПознавчі Україна

EventsТехноПознавчі Україна

ПОКАЗАТИ

Налаштування

ІНФОРМАЦІЯПОСИЛАННЯ ТА СОЦІАЛЬНІ МЕРЕЖІВИДАЛИТИ СПІЛЬНОТУ

TECH

ЗАВАНТАЖИТИ КАРТИНКУ СПІЛЬНОТІ

Назва спільноти*
ТехноПознавчі Україна

Локація*
Житомир

Напрямки спільноти*
Tech

Коротка історія спільноти
"ТехноПознавчі Україна" - це спільнота для всіх, хто цікавиться технологіями, інноваціями та сучасними науковими досягненнями.

Опис спільноти
"ТехноПознавчі Україна" - це спільнота для всіх, хто цікавиться технологіями, інноваціями та сучасними науковими досягненнями. Ми об'єднуємо розробників, інженерів, науковців, студентів та просто ентузіастів, які прагнуть йти в ногу з часом і дізнатися про новітні технологічні тренди. Наша місія - сприяти обміну знаннями та досвідом, підтримувати інновації та надихати на нові технічні досягнення. Наші учасники мають можливість брати участь у розмовітих заходах, таких як хакатони, семінари, майстер-класи, конференції та науково-дослідні проекти. Ми створюємо простір, де кожен може поділитися своїми ідеями та знаннями, а також знайти нових друзів та колеги.

Рис. 3.22 – Налаштування спільноти

Спільнота:
ТехноПознавчі Україна

EventsТехноПознавчі Україна

ПОКАЗАТИ

Налаштування

ІНФОРМАЦІЯПОСИЛАННЯ ТА СОЦІАЛЬНІ МЕРЕЖІВИДАЛИТИ СПІЛЬНОТУ

Website URL
https://businessday.ng/companies/article/the-tech-creatives-empower-ma

Facebook URL

Twitter URL

LinkedIn URL

Instagram URL
https://www.instagram.com/mykhailo_ddunov/

YouTube URL

Discord URL

Slack URL

Twitch URL

Medium URL

TikTok URL

Telegram URL
https://t.me/lozzi_tech

ЗБЕРЕГТИ ЗМІНИ

Рис. 3.23 – Налаштування посилань спільноти

Спільнота:
ТехноПознавчі Україна

EventsТехноПознавчі Україна

ПОКАЗАТИ

Налаштування

ІНФОРМАЦІЯПОСИЛАННЯ ТА СОЦІАЛЬНІ МЕРЕЖІВИДАЛИТИ СПІЛЬНОТУ

Видалити спільноту
Видалення спільноти незворотний процес. Після видалення ви вже не зможете відновити спільноту. Все інформація про спільноту буде видалено.

ВИДАЛИТИ СПІЛЬНОТУ

Рис. 3.24 – Сторінка видалення спільноти

На сторінці «Реєстраційна форма» організатор може створити форму, яку повинен заповнити користувач, щоб вступити у спільноту (рис. 3.25). Також в іншій вкладці є можливість перегляду заповнених форм.

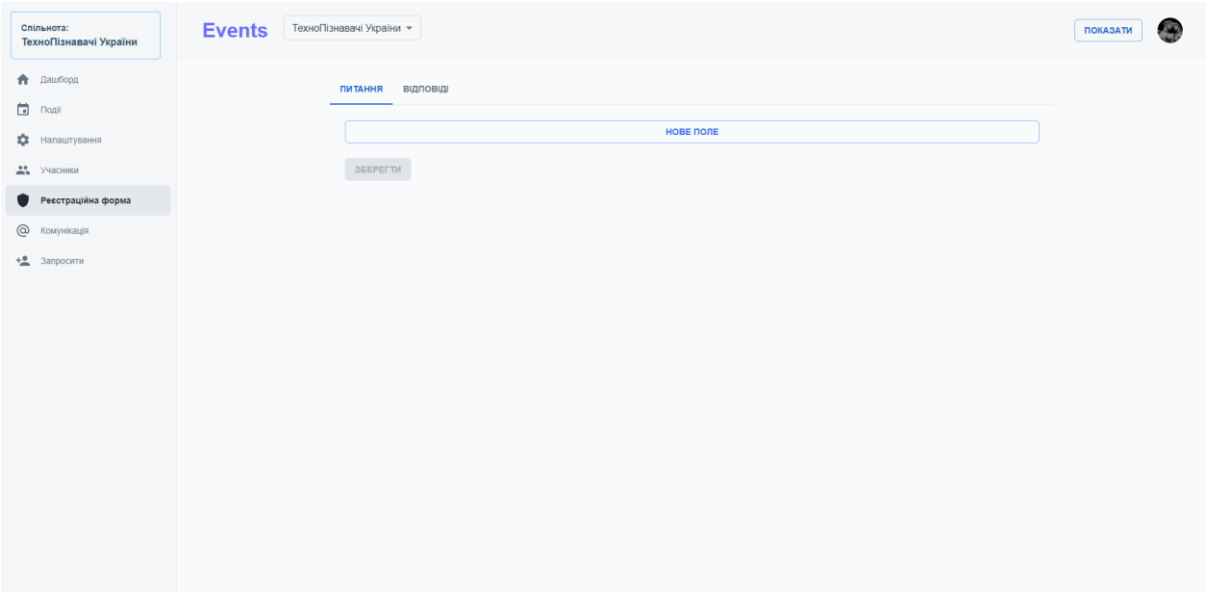


Рис. 3.25 – Сторінка створення реєстраційної форми

Поле форми може бути: коротким текстом, довгим текстом, один варіант відповіді або декілька варіантів відповіді (рис. 3.26). Також є можливість налаштування обов’язковості поля. Додамо поля до форми (рис. 3.27).

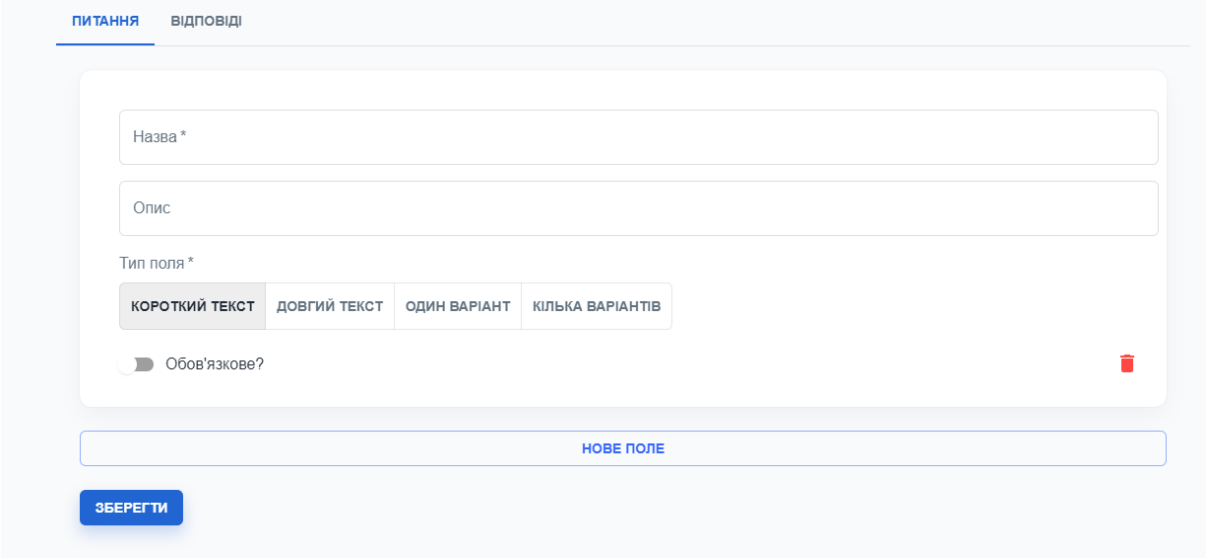


Рис. 3.26 – Поле реєстраційної форми

Назва *

Місто проживання

Опис

Вкажіть місто, в якому ви проживаєте.

Тип поля *

КОРОТКИЙ ТЕКСТ

ДОВГИЙ ТЕКСТ

ОДИН ВАРІАНТ

КІЛЬКА ВАРІАНТІВ

Обов'язкове?

Назва *

Ваш професійний рівень

Опис

Виберіть ваш професійний рівень у сфері технологій.

Тип поля *

КОРОТКИЙ ТЕКСТ

ДОВГИЙ ТЕКСТ

ОДИН ВАРІАНТ

КІЛЬКА ВАРІАНТІВ

Варіант *

Початківець

Варіант *

Фахівець

Рис. 3.27 – Заповнена форма

В результаті налаштування, було отримано заповнену сторінку спільноти (рис. 3.28).

Моя спільнота

Моя подія

Дослідити спільноту

Дослідити події

Events

Пошук

TECH

ТехноПізнавачі України

Житомир • 1 учасників

СТАТИ УЧАСНИКОМ

опис

події

Опис

"ТехноПізнавачі України" - це спільнота для всіх, хто цікавиться технологіями, інноваціями та сучасними науковими досягненнями. Ми об'єднуємо розробників, інженерів, науковців, студентів та просто ентузіастів, які прагнуть йти в ногу з часом і дізнаватися про новітні технологічні тренди. Наша місія - сприяти обміну знаннями та досвідом, підтримувати інновації та надихати на нові технічні досягнення.

Наші учасники мають можливість брати участь у різноманітних заходах, таких як хакатони, семінари, майстер-класи, конференції та науково-популярні лекції. Ми створюємо платформу для нетворкінгу, спільних проєктів та розвитку професійних навичок.

Наші цілі:

- Популяризація технологій** - Просування знань про сучасні технології та інновації серед широкої аудиторії.
- Освіта та навчання** - Проведення освітніх заходів для підвищення кваліфікації та розвитку нових навичок у сфері технологій.
- Інновації та розробки** - Сприяння створенню та реалізації інноваційних проєктів і стартапів.
- Нетворкінг** - Створення платформи для взаємодії між професіоналами та аматорами, обміну ідеями та досвідом.
- Інформаційна підтримка** - Надання актуальної інформації про технологічні тренди, нові розробки та дослідження.

Наші заходи:

- Хакатони** - Інтенсивні змагання з розробки програмного забезпечення, що об'єднують команди для вирішення технологічних завдань.
- Семінари та майстер-класи** - Освітні заходи, де учасники можуть поглибити свої знання у різних сферах технологій.

Рис. 3.28 – Створена сторінка спільноти

Натиснувши на кнопку «Стати учасником», з'являється вікно із реєстраційною формою учасника (рис. 3.29).

					ІПЗ.КР.Б-121-24-ПЗ	66

Форма учасника

Для того щоб стати учасником спільноти, заповніть форму.

ПІБ *

Введіть ваше повне ім'я.

Електронна пошта *

Введіть вашу електронну пошту для зв'язку.

Номер телефону *

Введіть ваш контактний номер телефону.

Місто проживання *

Вкажіть місто, в якому ви проживаєте.

Ваш професійний рівень *

☐ Початківець

☐ Фахівець

☐ Професіонал

☐ Дослідник

Інтереси у технологіях *

☐ Штучний інтелект

☐ Робототехніка

☐ Нанотехнології

☐ Інше

Рис. 3.29 – Реєстраційна форма учасника створеної спільноти

Заповнивши форму, користувач став учасником спільноти (рис. 3.30).

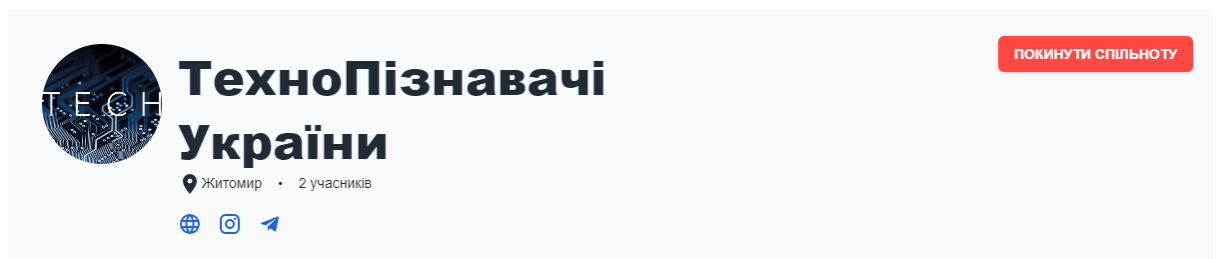


Рис. 3.30 – Результат заповнення реєстраційної форми

Перейшовши назад на обліковий запис організатора, на сторінці «Учасники» з'явився запис про зареєстрованого користувача (рис. 3.31). На цій же сторінці є можливість видалити користувача із учасників спільноти.

Учасники

Ім'я	Логін	Дата приєднання	Дія
Владислав Скаковський	Vladyslav3345	27 травня 2024 р.	
Maksym	maksym_maksym_com	27 травня 2024 р.	⋮

🗑️ Видалити учасника

Рядків на сторінці: 10 1–2 з 2 < >

Рис. 3.31 – Учасники спільноти

На сторінці відповідей на реєстраційну форму також з’явився новий запис про користувача (рис. 3.32).

ПИТАННЯ

ВІДПОВІДІ

Ім'я	Логін	Дата відповіді	
Maksym	maksym_maksym_com	27 травня 2024 р. о 23:21	ВІДПОВІДІ

Рядків на сторінці: 10 1–1 з 1 < >

Рис. 3.32 – Запис про заповнення реєстраційної форми

Натиснувши на кнопку «Відповіді», відбувається перехід на деталі заповненої форми (рис. 3.33).

Maksymmaksym_maksym_com27 травня 2024 р. о 23:21

ПІБ *

Максименко Максим Максимович

Введіть ваше повне ім'я.

Електронна пошта *

maksym@maksym.com

Введіть вашу електронну пошту для зв'язку.

Номер телефону *

0987135000

Введіть ваш контактний номер телефону.

Місто проживання *

Житомир

Вкажіть місто, в якому ви проживаєте.

Ваш професійний рівень *

☐ Початківець

☒ Фахівець

☐ Професіонал

☐ Дослідник

Інтереси у технологіях *

☒ Штучний інтелект

☒ Робототехніка

Рис. 3.33 – Заповнена учасником форма

На сторінці «Комунікація» є можливість відправляти листи на електронну пошту учасникам спільноти (рис. 3.34).

Комунікація

ЛИСТ УЧАСНИКАМ СПІЛЬНОТИ

Тема *

ТехноПізнавачі - новина

Сповідання *

↶↷Paragraph

B*I*







Новина для учасників спільноти - ми розширюємось!
Усім дякую за участь!

НАДІСЛАТИ

Рис. 3.34 – Сторінка комунікації із учасниками спільноти

Надіславши листа із заповненою темою та змістом листа, було отримано повідомлення на електронну пошту (рис. 3.35).

					ІПЗ.КР.Б-121-24-ПЗ	
						69

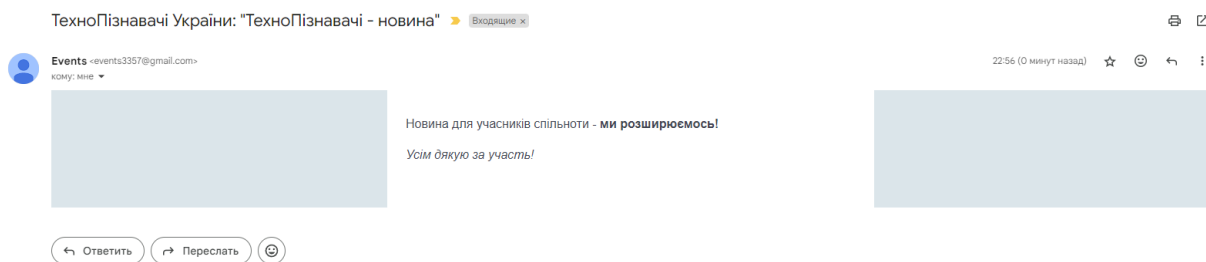


Рис. 3.35 – Отриманий лист

На сторінці «Запросити» організатор може надіслати електронний лист із запрошення вступити до спільноти (рис. 3.36). Якщо на введених електронних адресах є створені облікові записи, то їм надійде лист із посиланням.

Рис. 3.36 – Сторінка запрошення учасників до спільноти

Як результат, було отримано лист із посиланням на спільноту (рис. 3.37).

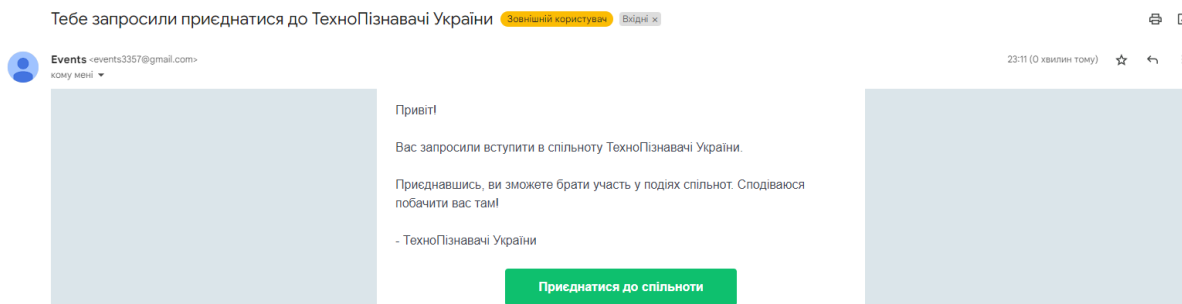


Рис. 3.37 – Лист із запрошенням до спільноти

Перейшовши на сторінку «Події», вибравши її в боковому меню, користувач потрапляє на сторінку зі списком подій спільноти: минулих і майбутніх (рис. 3.38). Наразі вона порожня.

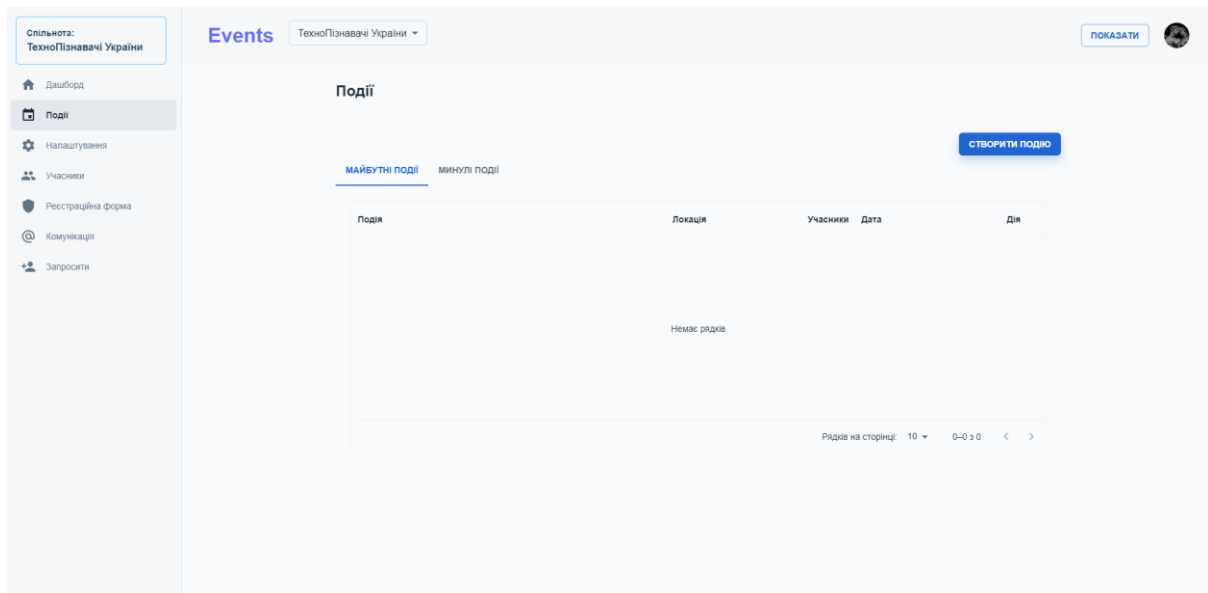


Рис. 3.38 – Події спільноти

Натиснувши на кнопку «Створити спільноту», відбувається перехід на сторінку із формою створення спільноти (рис. 3.39).

Рис. 3.39 – Форма створення події

Для місця проведення є вибір або онлайн, для цього потрібно ввести посилання, або офлайн, потрібно ввести адресу (рис. 3.40). Також є можливість встановити ліміт на максимальну кількість учасників у події. Якщо учасники

події мають бути підтверджені, перед тим як зареєструватися на неї, то потрібно увімкнути відповідне налаштування.

☐ Онлайн ☒ Офлайн

Місто *

Вулиця *

Назва локації *

Поштовий індекс

Кількість учасників

☐ Необмежена кількість учасників

☒ Максимальна кількість учасників

Осіб *

☐ Підтвердження участі

ОПУБЛІКУВАТИ ЗАРАЗ

Рис. 3.40 – Налаштування місця проведення та максимальної кількості осіб

Заповнивши поля форми та відправивши її, було створено нову подію. Її можна побачити серед списку (рис. 3.41).

Події

СТВОРИТИ ПОДІЮ

МАЙБУТНІ ПОДІЇ

МИНУЛІ ПОДІЇ

Подія	Локація	Учасники	Дата	Дія
Технологічна Конференція 2024: Інновації для Майбутнього	Онлайн	1	18 червня 2024 р.	Редагувати подію Відмінити подію Видалити подію

Рис. 3.41 – Результат створення події

Створену подію можна редагувати, відмінити або видалити. Натиснувши на кнопку редагування, користувача перенесе на панель організатора події, на сторінку редагування події (рис. 3.42).

Рис. 3.42 – Сторінка редагування події

Для події можна додавати спікерів. Для цього потрібно перейти на сторінку «Спікери» в боковому меню. Натиснувши на кнопку «Додати спікера» з’явиться вікно із формою (рис. 3.43).

Рис. 3.43 – Форма додавання спікера

Додавши спікерів, було отримано список із ними (рис. 3.44).

Спікери

ДОДАТИ СПІКЕРА

Ім'я	Посада	Компанія	Дія
Андрій Коваленко	Головний технологічний офіцер	Tech Innovations Inc.	РЕДАГУВАТИ ВИДАЛИТИ
Ольга Мельник	Викладачка та дослідниця	Київський Політехнічний Інститут	РЕДАГУВАТИ ВИДАЛИТИ
Максим Ткаченко	CEO та засновник	StartUp Hub	РЕДАГУВАТИ ВИДАЛИТИ

Рис. 3.44 – Список спікерів

Додавши спікерів, є можливість додати доповіді події. Для цього потрібно перейти на сторінку «Програма». Натиснувши на кнопку «Додати доповідь», з’являється вікно із формою (рис. 3.45).

Розклад

18 червня 2024 р. ▾

Назва

ДОДАТИ ДОПОВІДЬ

Додати нову доповідь

Назва *

Початок *

12:00 ⌚

Тривалість (хвилини) *

Рівень *

▾

Опис *

Спікери *

▾

ЗАКРИТИ

ЗБЕРЕГТИ

Рис. 3.45 – Форма додавання доповіді

Якщо подія проводиться протягом декількох днів, у списку можна вибрати дату, щоб відфільтрувати список та на яку додається поточна доповідь (рис. 3.46).

18 червня 2024 р. ▾

18 червня 2024 р.

Назва	Початок/ Кінець	Тривалість, хв	Рівень	Спікери	Дія
Майбутнє Штучного Інтелекту	12:00 - 13:00	60	Середній	Андрій Коваленко	РЕДАГУВАТИ ВИДАЛИТИ
Нанотехнології: Від Теорії до Практики	13:00 - 14:00	60	Початковий	Ольга Мельник, Максим Ткаченко	РЕДАГУВАТИ ВИДАЛИТИ
Інновації та Стартупи: Як Перетворити Ідею на Успішний Бізнес	14:00 - 15:00	60	Професійний	Максим Ткаченко	РЕДАГУВАТИ ВИДАЛИТИ

ДОДАТИ ДОПОВІДЬ

Рис. 3.46 – Список доповідей

На сторінці «Завантаження» організатор може додавати картинки до події, натиснувши на кнопку «Завантажити картинки» (рис. 3.47).

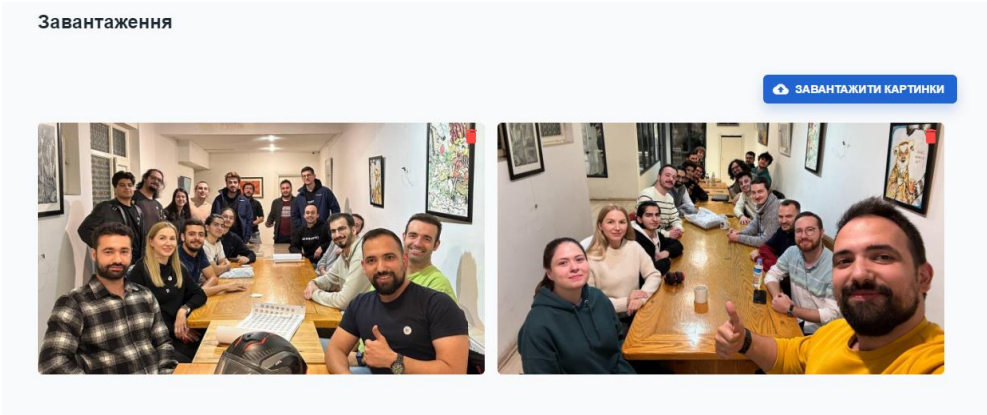


Рис. 3.47 – Картинки події

В результаті, подію було додано на сторінку спільноти (рис. 3.48) та створено окрему сторінку із деталями її опису (рис. 3.49 – 3.50).

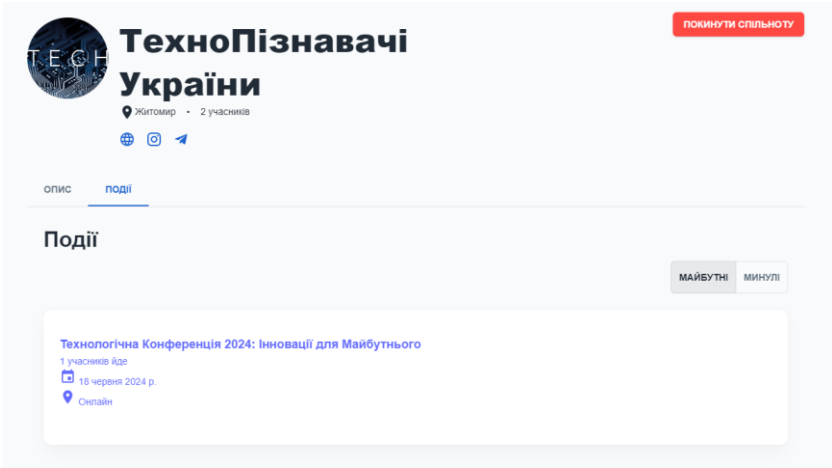


Рис. 3.48 – Подія на сторінці спільноти

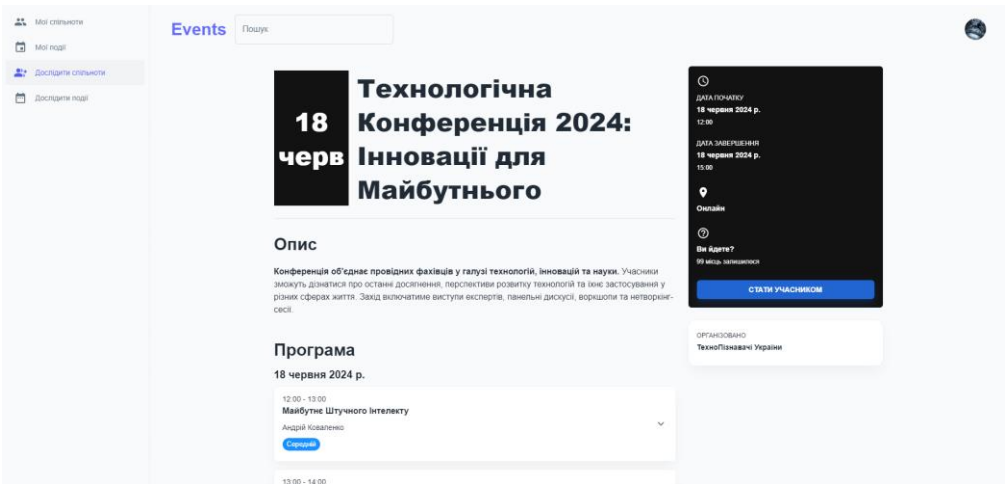


Рис. 3.49 – Сторінка створеної події

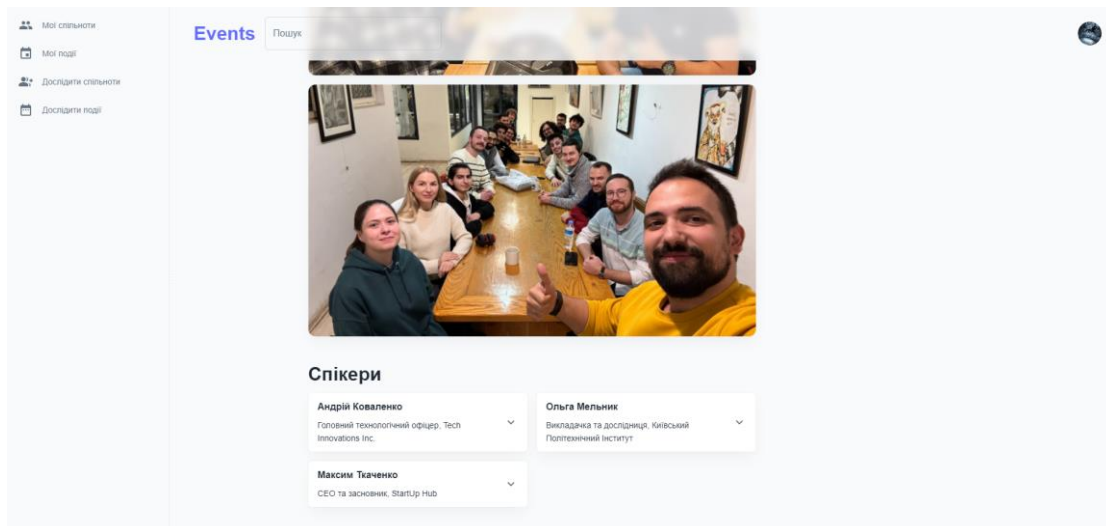


Рис. 3.50 – Сторінка створеної події

Подавши заявку на участь у події, вона потрапляє до списку очікування (рис. 3.51). Організатор може або затвердити або відхилити її. Затвердивши її, запис про учасника потрапляє у відповідний список (рис. 3.52).

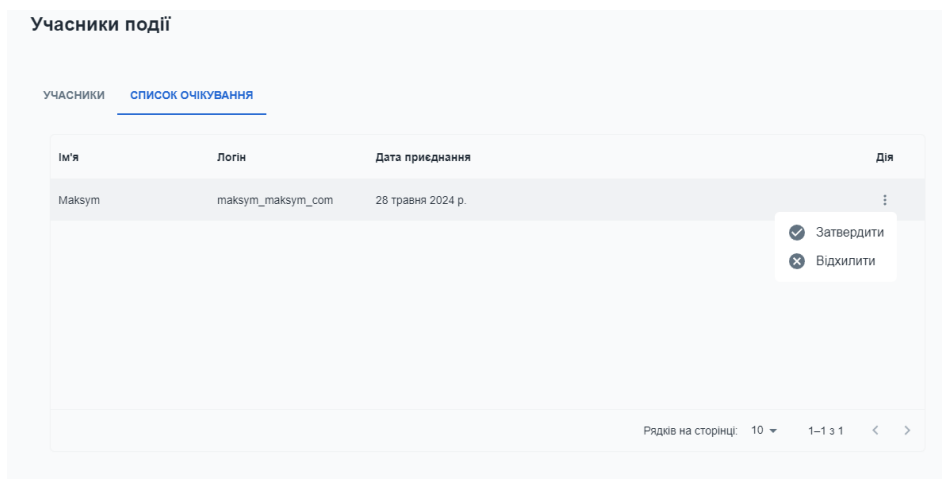


Рис. 3.51 – Список очікування

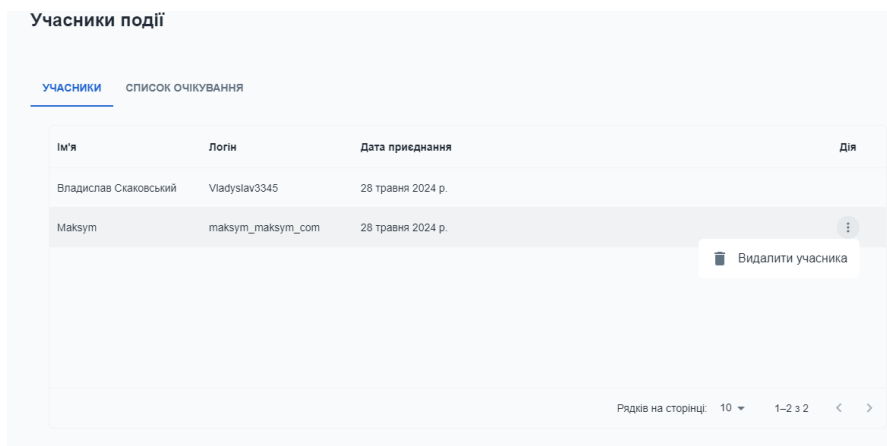


Рис. 3.52 – Список учасників події

Так само як учасникам спільноти, організатор може надсилати листи учасникам події. Для цього потрібно перейти на сторінку «Комунікація» (рис. 3.53). Організатор може обрати кому надсилати лист: або підтвердженим учасникам, або не підтвердженим учасникам, або і тим і іншим.

Комунікація

Лист відвідувачам події

☐ Надіслати учасникам, які ще не підтвердили участь

☐ Надіслати учасникам, які підтвердили участь

Тема *

Сповіщення *

← → Paragraph **B** *I*

НАДІСЛАТИ

Рис. 3.53 – Комунікація з учасниками події

Окрім видалення, подію можна відмінити. При цьому усім учасникам події буде надіслано сповіщення на електронну пошту (рис. 3.54). Також подія буде позначена відповідним чином на її сторінці та інших місцях (рис. 3.55). Для цього потрібно вибрати відповідну опцію на сторінці списку подій спільноти.

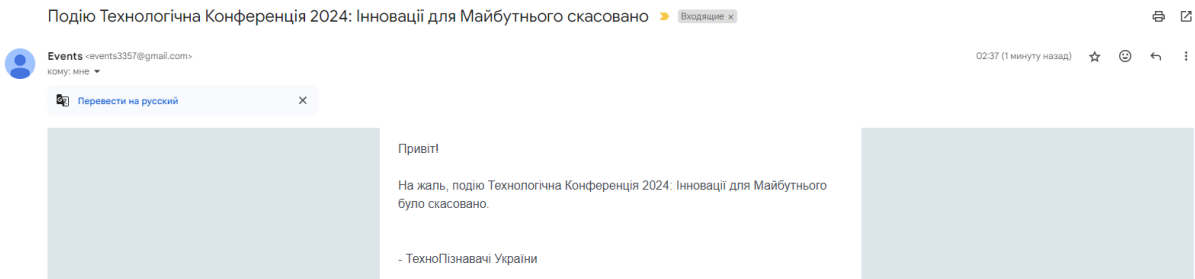


Рис. 3.54 – Лист про скасування події

18 черв

Технологічна Конференція 2024: Інновації для Майбутнього

Ця подія була скасована

Опис

Конференція об'єднає провідних фахівців у галузі технологій, інновацій та науки. Учасники зможуть дізнатися про останні досягнення, перспективи розвитку технологій та їхнє застосування у різних сферах життя. Захід включатиме виступи експертів, панельні дискусії, воркшопи та нетворкінг-сесії.

ДАТА ПОЧАТКУ
18 червня 2024 р.
12:00

ДАТА ЗАВЕРШЕННЯ
18 червня 2024 р.
15:00

Онлайн
Перейти до адреси події

ОРГАНІЗОВАНО
ТехноПізнавачі України

Рис. 3.55 – Сторінка скасованої події

Для управління обліковими записами розроблено роль адміністратора. Для входу на панель адміністратора, потрібно перейти по посиланню /admin/account/login (рис. 3.56).

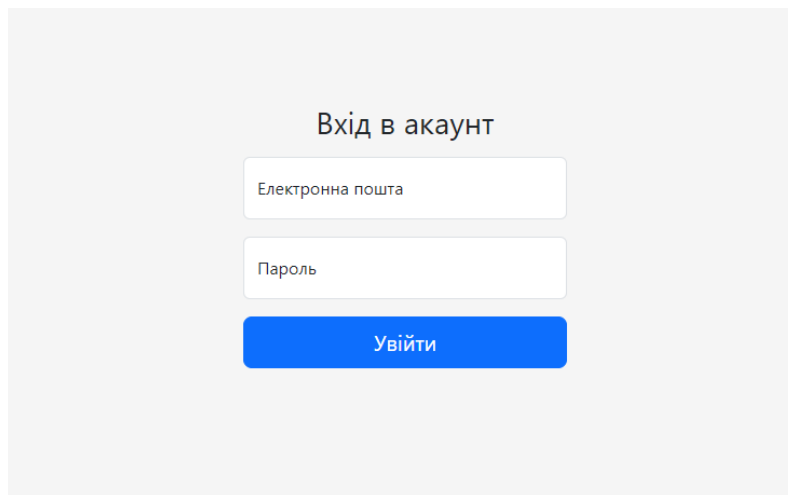


Рис. 3.56 – Сторінка входу до панелі адміністратора

Увійшовши до неї, адміністратора зустрічає сторінка управління обліковими записами (рис. 3.57). Представлено список усіх наявних облікових записів у системі. Статус «Організатор» отримують облікові записи, які створили хоча б одну спільноту. Звідси він може блокувати користувачам вхід до свого облікового запису (рис. 3.58).

Events				Вийти
Користувачі				
#	Ім'я	Електронна пошта	Статус	
3b6ea791-1ce8-41e0-8ea0-b6aa7c127fa2	Admin	admin@admin.com	Адміністратор	
880b6f23-7653-41e2-b06e-c9f6190467b4	Олександр	oleksandr@oleksandr.com	Організатор	🔒
8a449371-aa81-4966-9439-25ccd5ba551c	Владислав Скаковський	vlad.sl.2003@gmail.com	Організатор	🔒
c854f908-1563-4315-b43a-370721bc6fe	Андрій	adrii@test-events.com.ua	Відвідувач	🔒
d27a949f-fa72-49cd-ac4a-22912e17ffb0	Maksym	maksym@maksym.com	Відвідувач	🔒
f2038c08-1a7d-43ef-a420-bb3f49ea2a1c	Влад ІПЗ	ipz201_svo@student.ztu.edu.ua	Відвідувач	🔒

Рис. 3.57 – Сторінка управління облікового запису

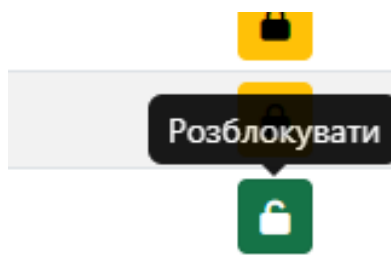


Рис. 3.58 – Результат блокування облікового запису

3.3. Тестування платформи

Для перевірки та забезпечення стабільної роботи системи, було проведено тестування.

На найнижчому рівні, рівні окремих одиниць коду, методів та функцій, частину додатку було покрито юніт-тестами. Було описано сорок два тест-кейси для чотирнадцяти класів (рис. 3.59). Помилки та неточності було виправлено, усі тести пройшли успішно.

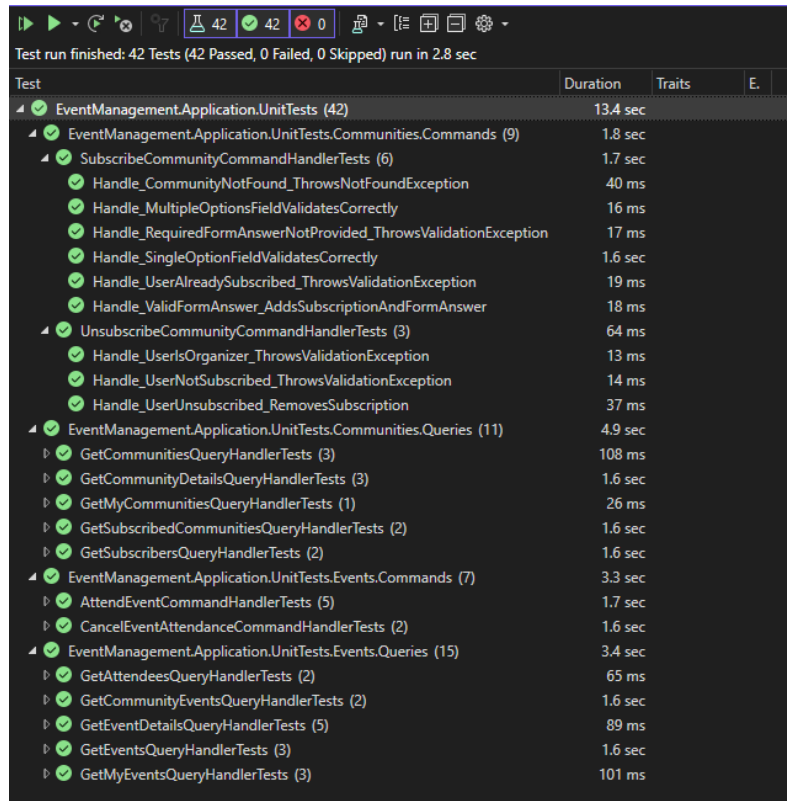


Рис. 3.59 – Юніт-тестування

Для перевірки загальної роботи додатку було використано техніки мануального тестування. Критичні помилки було виявлені та виправлені.

За допомогою розширення для браузера Accessibility Insights for Web було перевірено доступність сторінок для людей із обмеженими можливостями, було отримано результат перевірки сторінки події методом FastPass (рис. 3.60).

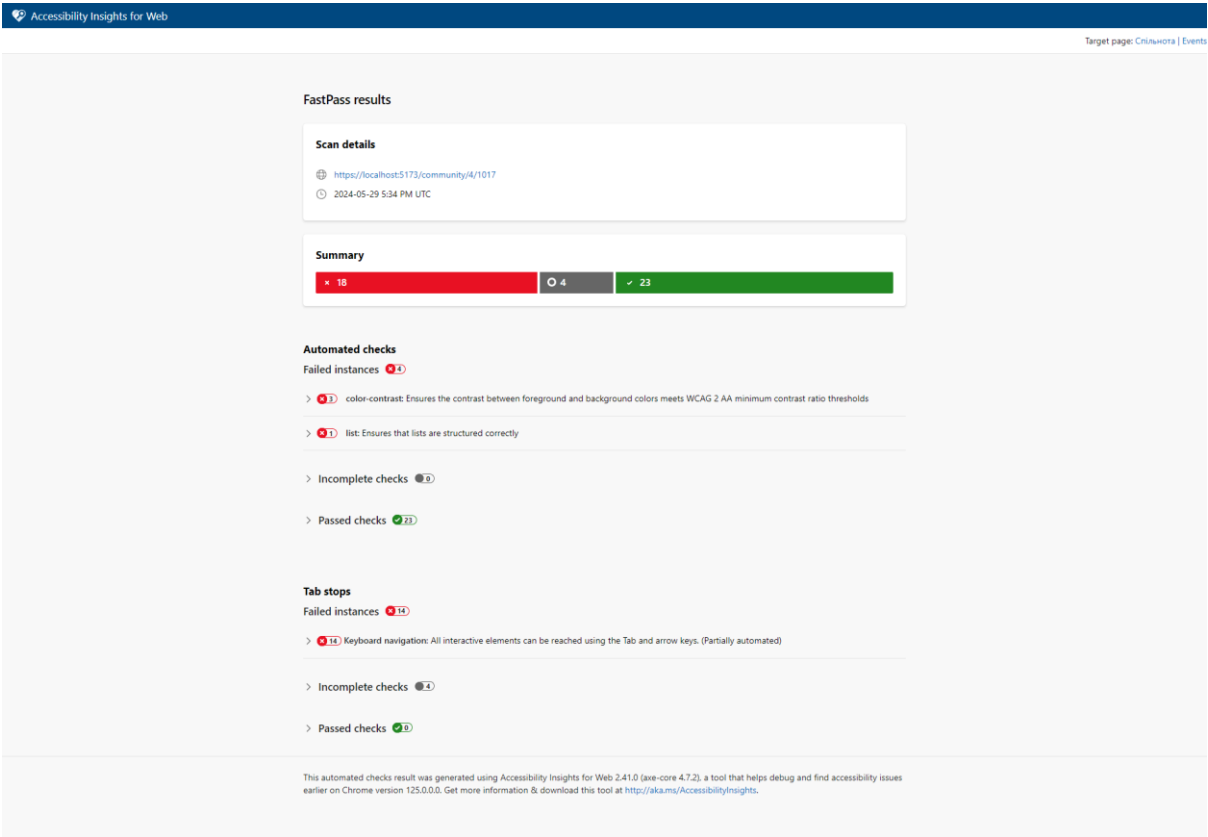


Рис. 3.60 – Результат перевірки інструментом Accessibility Insights for Web

За допомогою інструментів розробки в барузері було перевірено адаптивність застосунку, зокрема перевірено сторінку події для смартфона iPhone 14 Pro Max (рис. 3.61).

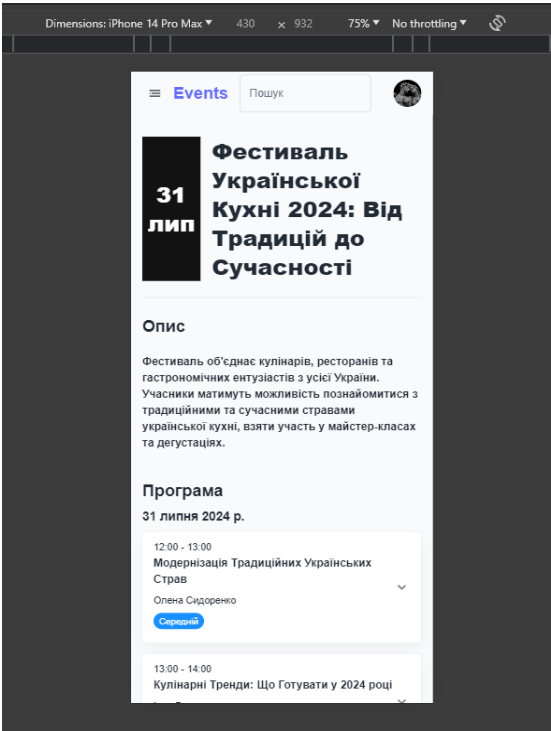


Рис. 3.61 – Адаптивність сторінки події

Висновки до третього розділу

В даному розділі було висвітлено процес розгортання додатку, описано компоненти, які підлягають розгортанню та відповідні середовища, показано зв'язки між ними. Для наочності було використано діаграму розгортання.

Було описано інтерфейс програмного застосунку, продемонстровано основний функціонал додатку. Для відвідувача, організатора та адміністратора показано варіанти використання системи у вигляді скріншотів.

Під час тестування було написано юніт-тести до серверної частини застосунку. Застосовано мануальне тестування для загальної перевірки роботи системи. Виправлено знайдені помилки. За допомогою браузерних інструментів Accessibility Insights for Web та Devtools було перевірено доступність та адаптивність для мобільних пристроїв.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було проаналізовано напрямки застосування інформаційних технологій для реалізації системи організації спільнот та подій. Було поставлено задачу та проаналізовано існуючі програмні продукти. Виділено три найбільш релевантних аналоги, порівняно їх функціонал, визначено основні модулі для розробки. Обґрунтовано вибір архітектури, інструментальних засобів, мов програмування та фреймворків, які буде використано. Для застосунку було обрано монолітну архітектуру.

Було спроектовано систему для управління спільнотами та подіями. Виділено трьох акторів: Учасник, Організатор та Адміністратор. Визначено варіанти використання для кожного із них. Визначено об'єктно-орієнтовану структуру системи, представлено діаграми класів сутностей додатку та декількох використаних патернів проєктування – Unit of Work та Proxy. Розроблено базу даних для застосунку, описано її таблиці. Спроектовано алгоритми реєстрації на подію та додавання нової події. Для їх зображення використано діаграми активності та послідовності відповідно. Після проєктування, було реалізовано функціонал системи для управління спільнотами та подіями.

Продемонстровано інтерфейс та порядок роботи із додатком. Описано алгоритм встановлення та налаштування параметрів платформи. Наведено скріншоти вебінтерфейсу користувача. Описано алгоритм роботи із системою для кожного виду користувачів. Проведено тестування застосунку, використано техніки мануального тестування, юніт-тестування, браузерні інструменти для перевірки доступності та адаптивності. виправлено критичні помилки. Додаток має інтуїтивний інтерфейс, який дозволяє користувача без великих зусиль створювати спільнотами та подіями і ставати їх учасниками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Скаковський В.О., Савіцький Р.С. Проектування архітектури системи управління спільнотами та подіями. Тези доповідей XIV Міжнародної науково-технічної конференції Інформаційно-комп'ютерні технології - 2023, 28-29 березня 2024 року. Житомир: «Житомирська політехніка», 2024. С.113-114.
2. Редполітика форуму [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/forum-articles-howto/>.
3. DOU [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/>.
4. Як використовувати ДОУ для промо події, конференції або тренінгу [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/partners/>.
5. Facebook [Електронний ресурс] – Режим доступу до ресурсу: <https://igroup.com.ua/seo-articles/facebook/>.
6. Мартин Ф. Patterns of Enterprise Application Architecture / Фаулер Мартин., 2003. – 533 с.
7. Чиста архітектура: Мистецтво розроблення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. – Харків: Вид-во «Ранок»: Фабула, 2021. – 368 с.
8. «Архітектура програмного забезпечення — це завжди пошук компромісів». Чим займається архітектор ПЗ: огляд професії від Олександра Савченка, Сіклум [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.ua/news/triam-1695152337>.
9. Посвістак В. С. Клієнт-серверна архітектура та її використання при розробці програмного забезпечення / В. С. Посвістак, Т. І. Демківська // Інформаційні технології в науці, виробництві та підприємництві: збірник наукових праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій / за заг. наук. ред. В. Ю. Щербаня. – Київ: Освіта України; ФОП Маслаков, 2020. – С. 78-81.
10. Морозов, А., Вакалюк, Т., Кубрак, Ю., Зосімович, Д. (2022). Аналіз факторів впливу на архітектури програмних систем. Information Technology:

					ІПЗ.КР.Б-121-24-ПЗ	
						83

Computer Science, Software Engineering and Cyber Security, 1, 44–52, doi: <https://doi.org/10.32782/IT/2022-1-7>.

11. Comparing Three-Layered and Clean Architecture for Web Development [Електронний ресурс] – Режим доступу до ресурсу: <https://betterprogramming.pub/comparing-three-layered-and-clean-architecture-for-web-development-533bda5a1df0>.

12. A tour of the C# language [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>.

13. Overview of ASP.NET Core [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0>.

14. Коротун, О., Вакалюк, Т., Зубрицький, В., & Гордієнко, І. (2022). ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ. Таврійський науковий вісник. Серія: Технічні науки, (1), 36-46. <https://doi.org/10.32851/tnv-tech.2022.1.4>.

15. Entity Framework Core [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/ef/core/>.

16. Elasticsearch [Електронний ресурс] – Режим доступу до ресурсу: <https://www.elastic.co/>.

17. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/>.

ДОДАТКИ

ЛІСТИНГ ВИХІДНОГО КОДУ

```

using MediatR;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace EventManagement.WebApi.Controllers;

[ApiController]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
[Route("api/[controller]")]
public abstract class ApiControllerBase : ControllerBase
{
    private ISender _mediator = null!;

    protected ISender Mediator => _mediator ??=
        HttpContext.RequestServices.GetRequiredService<ISender>();
}

using EventManagement.Application.Events.Commands.AttendEvent;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace EventManagement.WebApi.Controllers;

[Authorize]
public class AttendanceController : ApiControllerBase
{
    [HttpPost]
    public async Task<ActionResult> AttendEvent(AttendEventCommand command)
    {
        await Mediator.Send(command);

        return NoContent();
    }
}

using System.Reflection;
using EventManagement.Application.Common.Exceptions;
using EventManagement.Application.Common.Interfaces;
using EventManagement.Application.Common.Security;
using MediatR;

namespace EventManagement.Application.Common.Behaviors;

internal class AuthorizationBehavior<TRequest, TResponse> :
    IPipelineBehavior<TRequest, TResponse> where TRequest : notnull
{
    private readonly ICurrentUserService _currentUserService;
    private readonly IIdentityService _identityService;

    public AuthorizationBehavior(
        ICurrentUserService currentUserService,
        IIdentityService identityService)
    {
        _currentUserService = currentUserService;
        _identityService = identityService;
    }

    public async Task<TResponse> Handle(TRequest request,
        RequestHandlerDelegate<TResponse> next, CancellationToken cancellationToken)
    {

```

```

        var authorizeAttributes =
request.GetType().GetCustomAttributes<AuthorizeAttribute>();

        if (authorizeAttributes.Any())
        {
            // Must be authenticated user
            if (_currentUserService.UserId == null)
            {
                throw new UnauthorizedAccessException();
            }

            if (await _identityService.IsUserLockedAsync(_currentUserService.UserId))
            {
                throw new UnauthorizedAccessException();
            }

            // Role-based authorization
            var authorizeAttributesWithRoles = authorizeAttributes.Where(a =>
!string.IsNullOrEmpty(a.Roles));

            if (authorizeAttributesWithRoles.Any())
            {
                var authorized = false;

                foreach (var roles in authorizeAttributesWithRoles.Select(a =>
a.Roles.Split(', ')))
                {
                    foreach (var role in roles)
                    {
                        var isInRole = await
_identityService.IsInRoleAsync(_currentUserService.UserId, role.Trim());
                        if (isInRole)
                        {
                            authorized = true;
                            break;
                        }
                    }
                }

                // Must be a member of at least one role in roles
                if (!authorized)
                {
                    throw new ForbiddenAccessException();
                }
            }

            // Policy-based authorization
            var authorizeAttributesWithPolicies = authorizeAttributes.Where(a =>
!string.IsNullOrEmpty(a.Policy));
            if (authorizeAttributesWithPolicies.Any())
            {
                foreach (var policy in authorizeAttributesWithPolicies.Select(a =>
a.Policy))
                {
                    var authorized = await
_identityService.AuthorizeAsync(_currentUserService.UserId, policy);

                    if (!authorized)
                    {
                        throw new ForbiddenAccessException();
                    }
                }
            }
        }

        // User is authorized / authorization not required

```

```

        return await next();
    }
}

using MediatR;
using Microsoft.Extensions.Logging;

namespace EventManagement.Application.Common.Behaviors;

internal class UnhandledExceptionHandlerBehavior<TRequest, TResponse> :
IPipelineBehavior<TRequest, TResponse>
    where TRequest : notnull
{
    private readonly ILogger<TRequest> _logger;

    public UnhandledExceptionHandlerBehavior(ILogger<TRequest> logger)
    {
        _logger = logger;
    }

    public async Task<TResponse> Handle(TRequest request,
RequestHandlerDelegate<TResponse> next, CancellationToken cancellationToken)
    {
        try
        {
            return await next();
        }
        catch (Exception ex)
        {
            var requestName = typeof(TRequest).Name;

            _logger.LogError(ex, "Unhandled Exception for Request {Name} {@Request}",
requestName, request);

            throw;
        }
    }
}

namespace EventManagement.Application.Communities.Commands.SubscribeCommunity;

public sealed record FormAnswerDto(ICollection<FormFieldDto> FieldAnswers);

public sealed record FormFieldDto(
    string Name,
    object Value);

using EventManagement.Application.Common.Exceptions;
using EventManagement.Application.Common.Interfaces;
using EventManagement.Application.Services.Search;
using EventManagement.Domain.Entities;
using EventManagement.Domain.Entities.Community;
using EventManagement.Domain.Entities.Form.Answer;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace EventManagement.Application.Communities.Commands.SubscribeCommunity;

public sealed record SubscribeCommunityCommand(int CommunityId, FormAnswerDto?
FormAnswer = null) : IRequest;

internal class SubscribeCommunityCommandHandler(
    IApplicationDbContext context,
    ICurrentUserAccessor currentUserAccessor) :
IRequestHandler<SubscribeCommunityCommand>

```

```

{
    private readonly IApplicationDbContext _context = context;
    private readonly ICurrentUserAccessor _currentUserAccessor = currentUserAccessor;

    public async Task Handle(SubscribeCommunityCommand request, CancellationToken
cancellationToken)
    {
        var userId = _currentUserAccessor.UserId!;

        var community = await _context.Communities
            .Include(c => c.SubscriptionForm)
            .ThenInclude(f => f.Form)
            .FirstOrDefaultAsync(c => c.Id == request.CommunityId, cancellationToken)
            ?? throw new NotFoundException(nameof(Community), request.CommunityId);

        var userCommunity = await _context.Subscriptions
            .FirstOrDefaultAsync(s => s.UserId == userId && s.CommunityId ==
request.CommunityId, cancellationToken);

        if (userCommunity is not null)
            throw new ValidationException("Користувач вже підписаний на спільноту");

        var formAnswerRequired = community.SubscriptionForm.Form.Fields.Count != 0;

        if (formAnswerRequired)
        {
            if (request.FormAnswer is null)
                throw new ValidationException("Форма відповідей обов'язкова");

            var fieldAnswersInput = request.FormAnswer.FieldAnswers.ToDictionary(a =>
a.Name);
            var fieldAnswers = new List<FormFieldAnswer>();
            foreach (var field in community.SubscriptionForm.Form.Fields)
            {
                if (!fieldAnswersInput.TryGetValue(field.Name, out var fieldAnswer))
                    throw new ValidationException($"Поле {field.Name} обов'язкове для
заповнення");

                if (!field.Validate(fieldAnswer.Value))
                    throw new ValidationException($"Поле {field.Name} має невірне
значення");

                fieldAnswers.Add(new FormFieldAnswer { FormField = field, Value =
fieldAnswer.Value });
            }

            var formAnswer = new FormAnswer
            {
                FormId = community.SubscriptionForm.FormId,
                UserId = userId
            };
            formAnswer.FieldAnswers.AddRange(fieldAnswers);

            await _context.FormAnswers.AddAsync(formAnswer, cancellationToken);
        }

        await _context.Subscriptions.AddAsync(new Subscription
        {
            CommunityId = request.CommunityId,
            UserId = userId,
        }, cancellationToken);

        await _context.SaveChangesAsync(cancellationToken);
    }
}

```

```

using FluentValidation;

namespace EventManagement.Application.Communities.Commands.SubscribeCommunity;

public class SubscribeCommunityCommandValidator :
AbstractValidator<SubscribeCommunityCommand>
{
    public SubscribeCommunityCommandValidator()
    {
        RuleFor(x => x.CommunityId)
            .NotEmpty()
            .WithMessage("CommunityId is required.");
    }
}

using EventManagement.Application.Common.Exceptions;
using EventManagement.Application.Common.Interfaces;
using EventManagement.Domain.Entities.Community;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace EventManagement.Application.Communities.Commands.UnsubscribeCommunity;

public sealed record UnsubscribeCommunityCommand(int CommunityId) : IRequest;

internal sealed class UnsubscribeCommunityCommandHandler(
    IApplicationDbContext context,
    ICurrentUserAccessor currentUserAccessor)
    : IRequestHandler<UnsubscribeCommunityCommand>
{
    private readonly IApplicationDbContext _context = context;
    private readonly ICurrentUserAccessor _currentUserAccessor = currentUserAccessor;

    public async Task Handle(UnsubscribeCommunityCommand request, CancellationToken
cancellationToken)
    {
        var userId = _currentUserAccessor.UserId;

        var community = await _context.Communities
            .FirstOrDefaultAsync(c => c.Id == request.CommunityId, cancellationToken)
            ?? throw new NotFoundException(nameof(Community), request.CommunityId);

        var userCommunity = await _context.Subscriptions
            .FirstOrDefaultAsync(s => s.UserId == userId && s.CommunityId ==
request.CommunityId, cancellationToken)
            ?? throw new ValidationException("Користувач не підписаний на спільноту");

        if (community.OrganizerId == userId)
            throw new ValidationException("Організатор не може відписатися від своєї
спільноти");

        _context.Subscriptions.Remove(userCommunity);

        await _context.SaveChangesAsync(cancellationToken);
    }
}

using FluentValidation;

namespace EventManagement.Application.Communities.Commands.UnsubscribeCommunity;

public sealed class UnsubscribeCommunityCommandValidator
    : AbstractValidator<UnsubscribeCommunityCommand>
{

```

```

        public UnsubscribeCommunityCommandValidator()
        {
            RuleFor(x => x.CommunityId)
                .NotEmpty()
                .WithMessage("CommunityId is required.");
        }
    }

namespace EventManagement.Application.Communities.Queries.GetCommunities;

public sealed record CommunityDto(
    int Id,
    string Name,
    string Location,
    string? CommunityImageUrl,
    string Domain,
    int SubscribersCount);

using EventManagement.Application.Common.Interfaces;
using EventManagement.Application.Common.Pagination;
using EventManagement.Application.Common.Security;
using EventManagement.Application.Common.Services.Documents;
using EventManagement.Application.Common.Services.Search;
using EventManagement.Application.Services.Search;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace EventManagement.Application.Communities.Queries.GetCommunities;

[Authorize]
public sealed record GetCommunitiesQuery(
    string? SortBy,
    string? SortOrder,
    int Page,
    int PageSize,
    string[]? Location,
    string[]? Domain) : PagedRequest(Page, PageSize),
    IRequest<PagedList<CommunityDto>>;

internal sealed class GetCommunitiesQueryHandler(
    IApplicationDbContext context,
    ICommunitiesSearchService communitiesSearchService,
    IFileStorageService fileStorageService) : IRequestHandler<GetCommunitiesQuery,
    PagedList<CommunityDto>>
{
    private readonly IApplicationDbContext _context = context;
    private readonly ICommunitiesSearchService _communitiesSearchService =
communitiesSearchService;
    private readonly IFileStorageService _fileStorageService = fileStorageService;

    public async Task<PagedList<CommunityDto>> Handle(GetCommunitiesQuery request,
    CancellationToken cancellationToken)
    {
        var searchRequest = new SearchRequest<CommunityIndexDocument>
        {
            Page = request.Page - 1,
            PageSize = request.PageSize,
            SortBy = request.SortBy,
            IsSortAscending = request.SortOrder ==
SearchRequestSortingOrder.Ascending,
        };

        if (request.Location is not null)
        {

```

```

        searchRequest.Filters.Add(new TextFilter<CommunityIndexDocument>(c =>
c.Location, request.Location));
    }

    if (request.Domain is not null)
    {
        searchRequest.Filters.Add(new TextFilter<CommunityIndexDocument>(c =>
c.Domain, request.Domain));
    }

    var searchResult = await _communitiesSearchService.SearchAsync(searchRequest,
cancellationTokens);
    var communities = await _context.Communities
        .Where(c => searchResult.Results.Select(r => r.Id).Contains(c.Id))
        .ToDictionaryAsync(c => c.Id, cancellationTokens);

    var images = communities.Values.Select(c => c.CommunityImage)
        .Where(i => i is not null)
        .Cast<string>()
        .ToList();

    var communityImages = await _fileStorageService.GetFileUrlsAsync(images,
cancellationTokens);

    var dtos = new List<CommunityDto>();
    foreach (var result in searchResult.Results)
    {
        var community = communities[result.Id];
        string? communityImage = null;
        if (community.CommunityImage is not null &&
communityImages.TryGetValue(community.CommunityImage, out var imageUrl))
            communityImage = imageUrl.ToString();

        dtos.Add(new CommunityDto(
            result.Id,
            result.Name,
            result.Location,
            communityImage,
            result.Domain,
            result.SubscribersCount));
    }

    return new PagedList<CommunityDto>(
        dtos,
        searchResult.Page,
        searchResult.PageSize,
        searchResult.Total);
}

using EventManagement.Domain.Entities;

namespace EventManagement.Application.Communities.Queries.GetSubscribers;

internal static class GetSubscribersMapper
{
    public static SubscriberDto ToDto(this User user) => new(user.Id!, user.Name);
}

using EventManagement.Application.Common.Interfaces;
using EventManagement.Application.Common.Pagination;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace EventManagement.Application.Communities.Queries.GetSubscribers;

```

```

public sealed record GetSubscribersQuery(
    int CommunityId,
    int Page,
    int PageSize) : PagedRequest(Page, PageSize), IRequest<PagedList<SubscriberDto>>;

internal sealed class GetSubscribersQueryHandler(
    IApplicationDbContext context,
    IUserService userService)
    : IRequestHandler<GetSubscribersQuery, PagedList<SubscriberDto>>
{
    private readonly IApplicationDbContext _context = context;
    private readonly IUserService _userService = userService;

    public async Task<PagedList<SubscriberDto>> Handle(GetSubscribersQuery request,
        CancellationToken cancellationToken)
    {
        var subscriptionsQuery = _context.Subscriptions.Where(s => s.CommunityId ==
            request.CommunityId);

        var totalCount = await subscriptionsQuery.CountAsync(cancellationToken);

        var subscribersIds = await subscriptionsQuery.Skip((request.Page - 1) *
            request.PageSize)
            .Take(request.PageSize)
            .Select(s => s.UserId)
            .ToListAsync(cancellationToken);

        var users = await _userService.GetUsersByIdListAsync(subscribersIds,
            cancellationToken);
        var dtos = users.Select(u => u.ToDto()).ToList();

        return PagedList<SubscriberDto>.Create(dtos, request.Page, request.PageSize,
            totalCount);
    }
}

using EventManagement.Application.Common.Pagination;
using FluentValidation;

namespace EventManagement.Application.Communities.Queries.GetSubscribers;

public sealed class GetSubscribersValidator :
    PagedRequestValidator<GetSubscribersQuery>
{
    public GetSubscribersValidator(): base()
    {
        RuleFor(x => x.CommunityId).NotEmpty();
    }
}

namespace EventManagement.Application.Communities.Queries.GetSubscribers;

public sealed record SubscriberDto(string Id, string Name);

namespace EventManagement.Domain.Common;

public abstract class AuditableEntity
{
    public DateTime Created { get; set; }

    public string? CreatedBy { get; set; }

    public DateTime? LastModified { get; set; }
}

```

```

        public string? LastModifiedBy { get; set; }
    }

    namespace EventManagement.Domain.Common;

    public interface IHasDomainEvent
    {
        public List<DomainEvent> DomainEvents { get; set; }
    }

    public abstract class DomainEvent
    {
        protected DomainEvent()
        {
            DateOccurred = DateTimeOffset.UtcNow;
        }
        public bool IsPublished { get; set; }
        public DateTimeOffset DateOccurred { get; protected set; } = DateTime.UtcNow;
    }

    using EventManagement.Domain.Entities.CommunityEvent;

    namespace EventManagement.Domain.Entities;

    public class Attendee : AuditableEntity
    {
        public int Id { get; set; }

        public string UserId { get; set; } = default!;

        public int EventId { get; set; } = default!;

        public Event Event { get; set; } = default!;

        public AttendeeStatus Status { get; set; }

    }

    public enum AttendeeStatus
    {
        Pending = 0,
        Confirmed = 1,
        Cancelled = 2
    }

    using EventManagement.Domain.Entities.CommunityEvent;

    namespace EventManagement.Domain.Entities;

    public class Session : AuditableEntity
    {
        public int Id { get; set; }

        public string Title { get; set; } = default!;

        public DateTime StartTime { get; set; }

        public TimeSpan Duration { get; set; }

        public DateTime EndTime => StartTime + Duration;

        public string Description { get; set; } = default!;

        public int EventId { get; set; } = default!;

        public Event Event { get; set; } = default!;
    }

```

```

    public ICollection<Speaker> Speakers { get; set; } = [];

    public string Level { get; set; } = default!;
}

public static class SessionLevels
{
    public const string Beginner = "Beginner";

    public const string Intermediate = "Intermediate";

    public const string Advanced = "Advanced";

    public const string Expert = "Expert";

    public static IEnumerable<string> All => [Beginner, Intermediate, Advanced,
Expert];
}

import { Container, Stack, Avatar, Grid, Typography, Box, Button, Tabs, Tab,
IconButton } from "@mui/material";
import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";
import { axios } from '../api';
import { useParams, Link as RouterLink } from "react-router-dom";
import { SyntheticEvent, useState } from "react";
import { TabContext, TabList } from "@mui/lab";
import DescriptionTabPanel from "../DescriptionTabPanel";
import EventsTabPanel from "../EventsTabPanel";
import LocationOnIcon from '@mui/icons-material/LocationOn';
import SubscribeCommunityButton from "../SubscribeCommunityButton";
import UnsubscribeCommunityButton from "../UnsubscribeCommunityButton";
import CommunityOrganizerButtons from "../CommunityOrganizerButtons";
import { SocialMedia } from "../types";
import CommunitySocialMedia from "../communitySocialMedia";

const CommunityDetails = () => {
    const { communityId: communityIdParam } = useParams();
    const communityId = Number(communityIdParam);
    const { data } = useCommunityDetails(communityId);
    const [tabIndex, setTabIndex] = useState('2');

    const handleChange = (event: SyntheticEvent, newTabIndex: string) => {
        setTabIndex(newTabIndex);
    };

    return (
        <Container>
            <Stack direction="row" alignItems="center" justifyContent="space-between"
mb={1}>
                <Grid container>
                    <Grid item xs={12} md={8}>
                        <Stack direction="row" sx={{ pt: 1, mb: 3 }}>
                            <Avatar alt="community image" src={data?.communityImageUrl || undefined}
sx={{ width: 120, height: 120, mr: 2 }} />
                            <Stack direction="column">
                                <Typography variant="h2">
                                    {data?.name}
                                </Typography>
                                <Stack direction="row" mb={2}>
                                    <LocationOnIcon />
                                    <Typography variant="body2" sx={{ mr: 2 }}>
                                        {data?.location}
                                    </Typography>
                                    <Typography variant="body2" sx={{ mr: 2 }}>

```

```

        &#8226;
      </Typography>
      <Typography variant="body2" sx={{ mr: 2 }}>
        {data?.subscriberCount} учасників
      </Typography>
    </Stack>
    {data && <CommunitySocialMedia socialMedia={data?.socialMedia} />}
  </Stack>
</Grid>
<Grid item xs={12} md={4}>
  <Box mb={3} sx={{ display: 'flex', justifyContent: 'end' }}>
    {data && (data.isOrganizer ? (
      <CommunityOrganizerButtons communityId={communityId} />
    ) : data.isSubscribed ? (
      <UnsubscribeCommunityButton communityId={communityId} />
    ) : (
      <SubscribeCommunityButton
        isShowForm={data.requiresFormAnswer}
        communityId={communityId}
        formId={data.formId}
      />
    ))}
  </Box>
</Grid>
</Grid>
</Stack>
<Box>
  <TabContext value={tabIndex}>
    <Box sx={{ borderBottom: 1, borderColor: 'divider' }}>
      <TabList onChange={handleChange}>
        <Tab label="Опис" value="1" />
        <Tab label="Події" value="2" />
      </TabList>
    </Box>
    <DescriptionTabPanel value="1" description={data?.description} />
    <EventsTabPanel value="2" />
  </TabContext>
</Box>
</Container>
);
};

export default CommunityDetails;

function useCommunityDetails(communityId: number) {
  return useQuery({
    queryKey: ['community', { id: communityId }],
    queryFn: async () => {
      const res = await
      axios.get<GetCommunityDetailsQueryResult>(`/api/communities/${communityId}`);
      return res.data;
    },
  })
}

export type GetCommunityDetailsQueryResult = {
  id: number;
  name: string;
  location: string;
  shortDescription: string | null;
  description: string | null;
  subscriberCount: number;
  isSubscribed: boolean;
  isOrganizer: boolean;
  requiresFormAnswer: boolean;
}

```

```

    formId: number;
    communityImageUrl: string | null;
    socialMedia: SocialMedia;
  }

import { Box, Stack } from "@mui/material";
import { TextFieldElement, RadioButtonGroup, CheckboxButtonGroup, FormContainer,
useForm } from "react-hook-form-mui";
import { FormField } from "../common/queries";
import { LoadingButton } from "@mui/lab";
import * as yup from 'yup';
import { yupResolver } from "@hookform/resolvers/yup";
import { useMemo } from "react";

type Props = {
  fields: FormField[];
  onSubmit: (data: FormInputs) => void;
  isSubmitting: boolean;
};

const Form = ({ fields, onSubmit, isSubmitting }: Props) => {
  const schema: yup.ObjectSchema<FormInputs> = useMemo(() => yup.object().shape(
    fields.reduce((acc, field) => {
      switch (field.type) {
        case 'ShortText':
        case 'LongText':
          let textFieldSchema = field.type === 'ShortText'
            ? yup.string().max(255, 'Максимальна довжина тексту 255 символів')
            : yup.string().max(1000, 'Максимальна довжина тексту 1000 символів');

          if (field.isRequired)
            textFieldSchema = textFieldSchema.required('Це поле є обов\`язковим');

          return {
            ...acc,
            [field.name]: textFieldSchema,
          };
        case 'SingleOption':
          let singleOptionFieldSchema = yup.string();
          if (field.isRequired)
            singleOptionFieldSchema = singleOptionFieldSchema.required('Це поле є
обов\`язковим');
          return {
            ...acc,
            [field.name]: singleOptionFieldSchema,
          };
        case 'MultipleOptions':
          let multipleOptionsFieldSchema = yup.array().of(yup.string());
          if (field.isRequired)
            multipleOptionsFieldSchema = multipleOptionsFieldSchema.min(1, "Це поле є
обов\`язковим").required('Це поле є обов\`язковим');
          return {
            ...acc,
            [field.name]: multipleOptionsFieldSchema,
          };
        default:
          return acc;
      }
    }, {}),
  ), [yup, fields]);

  const form = useForm<FormInputs>({
    resolver: yupResolver(schema),
  });
  const { watch } = form;

```

```

return (
  <Box>
    <FormContainer<FormInputs> onSuccess={onSubmit} formContext={form}>
      <Stack spacing={2}>
        {fields.map(field => {
          switch (field.type) {
            case 'ShortText':
              return (
                <TextFieldElement
                  key={field.name}
                  name={field.name}
                  label={field.name}
                  required={field.isRequired}
                  helperText={field.description}
                />
              );
            case 'LongText':
              return (
                <TextFieldElement
                  key={field.name}
                  name={field.name}
                  label={field.name}
                  required={field.isRequired}
                  helperText={field.description}
                  multiline
                  rows={4}
                />
              );
            case 'SingleOption':
              return (
                <RadioButtonGroup
                  key={field.name}
                  name={field.name}
                  label={field.name}
                  required={field.isRequired}
                  options={field.options.map(o => ({ id: o, label: o })))}
                />
              );
            case 'MultipleOptions':
              return (
                <CheckboxButtonGroup
                  key={field.name}
                  name={field.name}
                  label={field.name}
                  required={field.isRequired}
                  options={field.options.map(o => ({ id: o, label: o })))}
                />
              );
            default:
              return null;
          }
        })}
        <LoadingButton type="submit" variant="contained"
loading={isSubmitting}>Стати учасником</LoadingButton>
      </Stack>
    </FormContainer>
  </Box>
);
};

export default Form;

type FormInputs = Record<string, string | string[]>;

```

```

import { Card, CardActionArea, CardContent, Stack, Typography, Alert, Box } from
"@mui/material";
import { Event } from "../types";
import LocationOnIcon from '@mui/icons-material/LocationOn';
import { Link } from "react-router-dom";
import { formatAsTime } from "../../utils/dateFormatters";

type Props = {
  event: Event;
}

const EventCard = ({ event }: Props) => {
  return (
    <Card sx={{ mb: 2 }}>
      <Link to={`/${event.community.id}/${event.id}`}>
        <CardActionArea>
          <CardContent>
            <Typography variant="h5">{event.name}</Typography>
            <Stack direction="row" spacing={2} mb={2}>
              <Typography
variant="subtitle2">{formatAsTime(event.startDate)}</Typography>
              <Typography variant="subtitle2">Від {event.community.name}</Typography>
            </Stack>
            <Box mb={2}>
              <Typography variant="subtitle2" gutterBottom sx={{ verticalAlign:
'middle' }}><LocationOnIcon /> {event.venue.type === 'Online' ? 'Онлайн' :
event.venue.address.locationName}</Typography>
              <Typography variant="body1">{event.attendeesCount} учасників
йде</Typography>
            </Box>
            {event.isCancelled && (
              <Alert severity="error">
                Ця подія була скасована
              </Alert>
            )}
          </CardContent>
        </CardActionArea>
      </Link>
    </Card>
  );
};

export default EventCard;

```