МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА/ПРОЄКТ
на здобуття освітнього ступеня бакалавра

на тему: «**Розробка системи подій та анонсів: модуль організації та відправки сповіщень»**

Виконав: студент 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Калініченко Микола Олексійович

Керівник:
Красюк Богдан Віталійович

Рецензент:
Гаврильчик Леонід Сергійович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № ____ від « ___ » _____ 2022 р.

Острог, 2022

Міністерство освіти і науки України

Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри економіко-математичного моделювання
та інформаційних технологій

_____ Ольга КРИВИЦЬКА

«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу/проєкт студента

Калініченка Миколи Олексійовича

1. Тема роботи/проєкту: Розробка системи подій та анонсів: модуль організації та відправки сповіщень.
керівник роботи/проєкту: Красюк Богдан Віталійович.

Затверджено наказом ректора НаУОА від 29 жовтня 2021 року №110

2. Термін здачі студентом закінченої роботи/проєкту: 03 червня 2022 року

3. Вихідні дані до роботи/проєкту: дана робота полягає у розробці системи подій та анонсів для студентів та викладачів Національного університету «Острозька академія»

4. Перелік завдань, які належить виконати: розробка серверної частини, розробка бази даних, розробка інтерфейсу, розробка телеграм боту.

5. Перелік графічного матеріалу: рисунки, таблиці.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Красюк Б. В.	01.12.2021р.	01.12.2021р.
2	Красюк Б. В.	01.12.2021р.	01.12.2021р.
3	Красюк Б. В.	01.12.2021р.	01.12.2021р.

7. Дата видачі завдання: 01.12.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1.	Постановка технічного завдання	до 01.12.2021	
2.	Розробка архітектури проекту, погодження функціоналу	до 20.12.2021	
3.	Розробка бази даних	до 08.01.2022	
4.	Розробка макетів інтерфейсу веб застосунку	до 24.02.2022	
5.	Розробка серверної частини проєкту	до 15.05.2022	
6.	Розробка клієнтської частини проєкту		
7.	Розробка телеграм боту		
8.	Попередній захист кваліфікаційної роботи/проєкту	до 01.06.2022р.	
9.	Здача кваліфікаційної роботи/проєкту на кафедру	03.06.2022 р.	

Студент: _____ Микола КАЛІНІЧЕНКО

Керівник кваліфікаційної роботи: _____ Богдан КРАСЮК

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: Розробка системи подій та анонсів: модуль організації та відправки сповіщень.
Автор: Калініченко М.О.
Науковий керівник: Красюк Б.В.

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *106 с., 55 рис., 8 табл., 4 додатки, 19 джерел.*
Ключові слова: система подій, події, база даних, серверна частина, клієнтська частина, телеграм бот.
Короткий зміст праці:

Метою кваліфікаційної роботи/проєкту було урізноманітнення соціального життя студентів «Острозької академії», шляхом створення програмного застосунку для відстеження подій університету. Даний проєкт реалізує собою створення подій для студентів з можливістю їхнього пошуку, фільтрацією за датою, факультетом та категорією та створення сповіщень для нагадування конкретної події. Користувачами додатку є студенти та викладачі вищих навчальних закладів. Для створення серверної частини даного проекту було використано середовища розробки Visual Studio 2022, Visual Studio Code. Платформою для проекту став фреймворк ASP.NET Core. Для шару доступу для даних було використано Entity Framework Core. При розробці було використано наступні бібліотеки та набори інструментів: Hangfire, Serilog, Postman, Swagger, Automapper. Системи контролю версій - Git та Github. Для розробки клієнтської частини було використано середовища розробки Visual Studio Code та інструмент для проектування інтерфейсу Adobe XD. В ході розробки було використано бібліотеки ReactJS, Axios, MUI. Для розробки бота в Telegram було використано пакет функцій Telegram.Bot, середовище розробки Visual Studio 2022.

The purpose of the final project was to diversify a students' social life by creating a software product for having an access to university events. This project provides creating events for students and professors with ability to search the particular event, filter events by date, faculty or category and also creating a notification for particular events. The users of this web application are students and professors of higher educational institutions. Visual Studio 2022 development environment, Visual Studio Code, was used to create the server part of this project. The platform for the project was the ASP.NET Core framework. Entity Framework Core was used for the data access layer. The following libraries and toolkits were used in the development: Hangfire, Serilog, Postman, Swagger, Automapper. Version control systems - Git and Github. The Visual Studio Code development environment and the Adobe XD interface design tool were used to develop the client side of the application. ReactJS, Axios, MUI libraries were used during the development for client side. Telegram.Bot, Visual Studio 2022 development environment, was used to develop the Telegram bot.

_____

ЗМІСТ

ВСТУП

Дедалі більше незручних процесів знаходять своє рішення шляхом створення програмного продукту, який є зручним та доступним. В епоху стрімкого розвитку технологій це є ідеальним підходом до вирішення проблеми. Все, що потрібно для доступу – мобільний або комп'ютерний пристрій та доступ до інтернету. Звичні речі для людини 21 століття.

На даний момент майже всі процеси, які мали модель розробки ще за часів не технологічного суспільства заміщаються мобільними додатками або веб-додатками. В «Острозькій академії» можна знайти чимало прикладів. Паперовий розклад вже давно не актуальний, тому є його веб-версія. Студенти та викладачі можуть переглядати новини та організаційну інформацію не на дошці новин, а на загальному сайті університету. Абітурієнтам не обов'язково телефонувати або їхати до університету, щоб дізнатись інформацію для вступу, натомість для цього є окреме рішення – інтерактивний сайт для абітурієнта. І це лише декілька прикладів, як програмні застосунки допомогли оптимізувати організаційну роботу академії.

Від важливих організаційних процесів можна переходити до вирішення соціальних проблем. Чи зручною є афіша заходів? Можливо, і зручною, якщо б студенти піднімали голови вище своїх телефонів. Але якщо студент знаходиться не в академії? Не зручною, бо це змушує студента або викладача йти до університету. Є рішення! Створення соціальних мереж. В цьому рішенні є, як і плюси, так і мінуси. Плюси: бюджетно, доступно, як для студентів, так і для викладачів. Мінуси: відсутність великої кількості функцій. Тому для вирішення цієї проблеми і розробляється даний проект.

Даний продукт поєднує в собі весь функціонал, якого бракувало у соціальних мережах із використанням звичних та зручних технологій для студента та викладачів.

Принцип простий: якщо користувач хоче запланувати своє проведення часу, він може відвідати веб-додаток «Події ОА». Користувач має можливість переглянути актуальні події, які відбудуться найближчим часом, перевірити наявність окремої події, використовуючи пошук та зручну навігацію, або просто переглянути всі події та обрати ту, яка найбільше зацікавила. Після цього, користувач може створити собі нагадування, щоб не забути про подію.

Залучення сучасних технологій, зрозумілий та простий інтерфейс дає змогу користувачам покращити своє соціальне життя та зробити це зручно та швидко.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ ТА ТЕХНІЧНЕ ЗАВДАННЯ

Проект передбачає створення єдиного застосунку, що дасть можливість користувачеві як адміністратору створювати події та зробити їх анонсування. В свою чергу, користувач в ролі клієнта системи матиме змогу стежити за подіями та отримувати сповіщення в разі наближення до фактичної дати проведення.

Взаємодія користувачів із програмним забезпеченням відбувається за допомогою веб застосунку.

Таким чином адміністратор, після проходження механізму автентифікації має доступ до функціоналу роботи із подіями та категоріями подій – створенням, редагуванням, переглядом, пошуком, видаленням. Клієнт в свою чергу може здійснювати пошук подій, фільтрувати їх за категоріями, підписуватись на обрані подій або на цілу категорію подій, переглядати прикріплені до них посилання з трансляціями тощо. Підписавшись на подію або категорію, користувач має змогу отримувати сповіщення про наближення події на обрану ним соціальну мережу чи месенджер.

Технічне завдання проекту полягає у наступному:

- механізм аутентифікації та авторизації для адміністратора;
- на сайті користувач може переглянути список подій та анонсів з можливістю підписатися на подію, категорію загалом. Має мати змогу фільтрувати події за категоріями, шукати події;
- механізм нагадувань про подію через Telegram бота:
    - за тиждень;
    - за день;
    - за годину;
    - про те, що подія розпочнеться через X днів;

- реалізувати роль адміністратора/менеджера системи який має змогу посторінково переглядати, сортувати, редагувати, шукати всі події;
- механізм додавання посилання на подію (наприклад посилання на відео/трансляцію на платформі YouTube).

РОЗДІЛ 2
ОРГАНІЗАЦІЯ ПРОЦЕСУ РОЗРОБКИ

Процес розробки організований із використанням методології “Scrum” за принципом “Agile”.

Принцип гнучкої розробки програмного забезпечення (Agile) відноситься до групи методологій з розробки програмного забезпечення, що засновані на ітераційній розробці, де вимоги до системи чи програмного продукту розвиваються завдяки співпраці між самоорганізованими та міжфункціональними командами. Такі методи та процеси зазвичай сприяють дисциплінованому процесу управління проектом, який заохочує часті перевірки та адаптації, лідерству, що заохочує командну роботу, самоорганізацію та відповідальність, використанню набору найкращих інженерних практик, що призначені для швидкого постачання високоякісного програмного забезпечення, і бізнес підхід, який узгоджує розробку з потребами клієнтів і цілями компанії. До гнучкої розробки відноситься будь-який процес розробки, який відповідає концепціям Agile-маніфесту. Цей маніфест був розроблений групою з чотирнадцяти провідних діячів сфери розробки програмного забезпечення [7].

Суть принципу Agile в тому, що «люди та їх взаємодія ставляться вище за процеси та інструменти розробки. Працюючий продукт є важливішим за вичерпну документацію. Співпраця із замовником важливіша за узгодження умов угоди. Готовність до змін важливіша за проходження початкового плану».

Scrum - підмножина Agile. Це легка організаційна структура процесів гнучкої розробки, яка є найбільш використовуваною. Вона допомагає командам працювати разом. Подібно до команди в регбі (звідки й пішла назва), яка тренується до великої гри, scrum заохочує команди вчитися на досвіді, самоорганізуватися, працюючи над проблемою, і думати над перемогами та поразками з метою вдосконалення.

Методологія Scrum полягала у поділі робочого процесу на рівні – спринти. Спринт – це короткий, обмежений у часі період, коли команда працює над виконанням встановленого обсягу роботи. Спринти є основою Scrum і Agile методологій, і їх правильне виконання спринтів допоможе команді постачати краще програмне забезпечення з меншою кількістю проблем в процесі розробки [8]. Наприкінці спринту, щочетверга групи обговорювали результати в онлайн режимі з використанням сервісу відеотелефонного зв’язку системи Google Meet, ставили питання керівнику у разі виникнення проблем та продовжували роботу з початком нового спринта.

РОЗДІЛ 3
ОГЛЯД НАЯВНИХ АНАЛОГІВ

На цей час аналог системи подій та анонсів є. Це сайт events.oa.edu.ua.

Табл. 3.1. Порівняння аналогу з проектом

Загальний опис	Система представляє собою календар подій у Національному університеті “Острозька академія”.
Функціонал	Користувач має змогу: - переглядати події та сортувати їх за категоріями та датами - шукати події - додавати події у Google Calendar та Outlook - переглядати час, що залишився до події
Недоліки	Основним недоліком системи є відсутність гнучкості сповіщення користувача. Основними користувачами системи є студенти, які рідко використовують Google Calendar або Outlook.
Наші переваги	Щоб покращити соціальне життя студентів, ми розробляємо систему, що надає користувачу простий та зрозумілий інтерфейс. Для цього ми використовуємо оповіщення через соціальні мережі - Telegram Bot.

РОЗДІЛ 4
ОПИС ПРЕДМЕТНОГО СЕРЕДОВИЩА

Розробку програмного забезпечення було розділено на декілька етапів – дизайн, клієнтську та серверну частини.

Кожен з цих процесів вимагав використання певного, специфічного для конкретних задач програмного середовища.

В ході розробки проекту було використано Microsoft Visual Studio 2022 для розробки серверної частини, Visual Studio Code для розробки інтерфейсу, Adobe XD для проектування та малювання дизайну інтерфейсу.

4.1. Visual Studio 2022

Visual Studio 2022 – інтегроване середовище розробки програмного забезпечення розроблене компанією Microsoft. Це програмне забезпечення дозволяє розробляти додатки для різних платформ на різних мовах програмування, в тому числі на ASP.NET [1].

Переваги Visual Studio 2022:

- проекти в повній мірі використовують функції Visual Studio для економії часу та підвищення продуктивності. Наприклад функції та пропозиції IntelliSense, перевірка синтаксису в редакторі, підсвітка синтаксису.
- файли проекту Visual Studio зберігають деталі будь-яких змін, які вносяться в середовище розробки.
- Visual Studio побудована в архітектурі, що підтримує можливість використання доповнень (Add-Ins), – плагінів від сторонніх розробників, що дає змогу розширювати можливості середовища розробки.
- Інтеграція з Git, що дозволяє зручно працювати із цією системою контролю версій.

4.2. Visual Studio Code

Visual Studio Code – кросплатформовий редактор коду, призначенням якого є створення та налагодження сучасних веб- та хмарних додатків [2].

Переваги Visual Studio Code:

- містить засоби рефакторингу коду, роботи з Git, зневадження коду як і у VS 2022.
- можливість встановлювати розширення, що дозволяє підвищити продуктивність та комфорт написання проекту.
- зручні засоби для написання коду для веб застосунків.

4.3. Adobe XD

Adobe XD – це векторний інструмент для проектування інтерфейсу для веб-програм і мобільних додатків. Мета використання – попередньо переглянути результат роботи безпосередньо на мобільних пристроях. Adobe XD дає змогу створювати каркаси веб-сайтів і створювати прототипи [3].

Переваги Adobe XD:

- простота інтерфейсу;
- можливість використання великої кількості плагінів;
- функціонал, що спрощує розробку макетів;
- прототипування (можливість з’єднати всі сторінки та переглянути навігацію);
- простий експорт елементів;

4.4. Postman

Postman – це програма, яка використовується для тестування API. Це HTTP-клієнт, який тестує HTTP-запити, використовуючи графічний інтерфейс

користувача, за допомогою якого можна отримати різні типи відповідей від сервера. Postman користується величезною популярністю завдяки своєму функціоналу, зручному та приємному графічному інтерфейсу, а також наявності детальної документації [4].

Переваги Postman:

Користувачі Postman можуть легко отримувати доступ до своїх файлів, увійшовши у свій обліковий запис на пристрої з інстальованою програмою Postman або розширенням для браузера.

Postman підтримує всі можливі методи HTTP, збереження прогресу, перетворення з API в код, зміну середовища розробки API та багато іншого.

Для відповідей HTTP у Postman підтримується чимало кодів статусу, щоб користувачі могли перевірити відповідь сервера. Це успішні запити (200 Ok), порожня відповідь (204 No Content), поганий (невідповідний) запит (400 Bad Request) і несанкціонований доступ (401 Unauthorized), і це лише деякі з них.

4.5. Ngrok

Ngrok – це платформа, яка за допомогою встановленої утиліти дозволяє організувати віддалений доступ на веб-сервер або якийсь інший сервіс, запущений ПК. Доступ організується через створений під час запуску ngrok безпечний тунель [5].

Переваги Ngrok:

Ngrok - це зворотній проксі-сервер, який встановлює безпечний канал між загальнодоступною кінцевою точкою та локально працюючим веб-сервером.

Ngrok може захоплювати та аналізувати трафік на всіх каналах, що зручно для подальшого аналізу та відтворення.

Дану платформу можна використовувати для доступу до веб-сайту, розгорнутому на локальному сервері, через зовнішню мережу.

Ngrok також підтримує відображення портів рівня TCP, не обмежуючись певною службою.

Підтримка платформ Mac OS X, Linux та Windows.

4.6. Github

Git – розподілена система контролю версій, яка дає змогу розробникам відслідковувати зміни у файлах та працювати над одним проектом разом із колегами.

GitHub - сервіс онлайн-хостингу репозиторіїв, що володіє всіма функціями розподіленого контролю версій та функціональністю управління вихідним кодом - все, що підтримує Git і навіть більше [6].

Переваги GitHub:

- Безкоштовне обслуговування, хоча є платні.

- Дуже швидкий пошук у структурі репозиторіїв.

- Велика спільнота і легко знайти допомогу.

- Він пропонує практичні інструменти для співпраці та гарну інтеграцію з Git.

- Легко інтегрується з іншими сервісами.

4.7. Hangfire

Hangfire – це бібліотека з відкритим кодом, яка дозволяє розробникам з максимальною легкістю планувати події у фоновому режимі. Це дуже гнучка бібліотека, яка пропонує різноманітні функції, необхідні для того, щоб зробити завдання планування роботи простим та зручним [10].

Переваги Hangfire:

- Інтегрована інформаційна панель, де можна переглядати роботи на черзі та при бажанні керувати ними вручну;
- Керування запитами у окремій базі даних
- Безпека робіт, адже навіть при випадкових зупинках системи заплановані роботи є збереженими та готовими до виконання після перезапуску системи.
- Механізм повторення спроб у разі виникнення виключень

4.8. Тестування API

Для тестування роботи програмного забезпечення серверної частини проекту було використано Postman та Swagger, оскільки ці інструменти є досить простими у використанні та є популярними.

Postman – дозволяє тестувати API за допомогою графічного інтерфейсу користувача, а також веб-інструменту, де користувачі зможуть отримати доступ до API та працювати з ними будь-де, де вони можуть мати підключення до Інтернету.

Крім того, Postman надає функції спільного доступу, які дозволяють легко обмінюватися викликами HTTP з іншими членами організації. Зокрема під час тестування проекту, ми створили структуровані компоненти папки, які в свою чергу містили групи запитів на виконання.

Swagger - це набір інструментів з відкритим кодом для написання API на основі REST. Це спрощує процес написання API, вказуючи стандарти та надаючи інструменти, необхідні для написання безпечних, продуктивних і масштабованих API.

Перевагою Swagger є те, що він дозволяє взаємодіяти з REST API за допомогою Swagger UI Framework і таким чином дає чітке уявлення про те, як API реагує на параметри та яку відповідь від сервера повертає.

Також до тестування серверної частини можна віднести логування та створення лог файлів. Логування (ведення журналу) це процес створення та зберігання лог файлів – файлів, які записують події, що відбуваються в роботі певної системи чи програмного продукту. Таким чином розробник, що впроваджує систему логування визначає де, в якій формі і в якому форматі будуть створюватись файли та які записи будуть з’являтися в процесі роботи програми. Таким чином визначивши таку структуру, розробник програмного забезпечення буде мати змогу відслідковувати помилки в системі і таким чином покращувати роботу продукту в подальшій розробці.

В даній системі використовується бібліотека Serilog. Перевагами цієї бібліотеки є:

- Структуроване логування та простота використання
- Чудова документація та спільнота
- Конфігурація на основі C#

4.9. Висновки

Таким чином ми обрали перераховані вище середовища розробки програмного забезпечення для розробки дизайну, клієнтської та серверної частин. Це програмне забезпечення в першу чергу є актуальним, тобто підтримується розробниками і в разі виникнення проблем, користувач має змогу отримати консультацію та вирішити проблеми. Воно є зручним у використанні, має інтеграцію з наявними системами контролю версій. Підходить до розробки програмного забезпечення із обраними бібліотеками, фреймворками тощо.

Також використання вищенаведених бібліотек, фреймворків, засобів тестування та інших наборів інструментів зумовлене простотою використання, надійністю, популярністю серед розробників, можливістю розширення тощо.

РОЗДІЛ 5
ОПИС АРХІТЕКТУРИ РІШЕННЯ

5.1. База даних

База даних – це організована сукупність структурованої інформації або даних, які зазвичай зберігаються в електронній формі в комп'ютерній системі. База даних зазвичай контролюється системою управління базами даних (СУБД). Разом дані та СУБД із програмними додатками, які з ними пов'язані, називають системою баз даних, часто скорочено до просто бази даних.

Дані в найпоширеніших типах баз даних, які є сьогодні, зазвичай представлені в рядках і стовпцях у множинах таблиць, для того, аби зробити обробку та запити на отримання даних ефективними. Таким чином, дані можна легко отримувати, керувати, змінювати, оновлювати, контролювати та впорядковувати. Більшість баз даних використовує мову структурованих запитів (SQL) для запису та запиту даних. Прикладами систем керування такими базами даних є MySQL. MariaDB. Oracle.

SQL – це мова програмування, що використовується майже всіма реляційними базами даних для запитів, маніпулювання та визначення даних, а також для забезпечення контролю доступу. SQL вперше був розроблений в IBM у 1970-х роках з Oracle як основним учасником, що призвело до впровадження стандарту SQL ANSI. SQL спонукав багато розширень від таких компаній, як IBM, Oracle і Microsoft. Незважаючи на те, що з'являються все нові мови програмування, SQL все ще широко використовується і по сьогодні.

Реляційні бази даних це такі колекції даних, в яких є заздалегідь встановлені зв'язки. В свою чергу існують і нереляційні бази даних (NoSQL), які дозволяють зберігати і маніпулювати неструктурованими та напівструктурованими даними (на відміну від реляційної бази даних, яка визначає, як повинні складатися всі

дані, що є в базі даних). Прикладами таких баз даних є MongoDB, CouchDB, CouchBase, Cassandra, HBase, Redis, Riak, Neo4J [11].

Система керування базою даних представлена у вигляді продукту MSSQL. Ця система розробляється і підтримується корпорацією Microsoft. На сервері розміщується база даних, що створюється за допомогою методу “Code First”.

Згідно цього підходу, спочатку в проекті, який буде працювати із базою даних описуються моделі - сутності бази даних, що представляють певну таблицю в ній, та зв’язки між ними. Таким чином моделі стають моделями предметної області або доменними моделями, тому потрібно свідомо підходити до розробки моделей. А решту роботи виконає Entity Framework. В цьому і є головна перевага “Code First” підходу, коли моделі системи стають моделями даних, на яких спирається структура Entity Framework. Потім за допомогою цих моделей створюється відповідна база даних, яка буде мати описаний взаємозв’язок між сутностями.

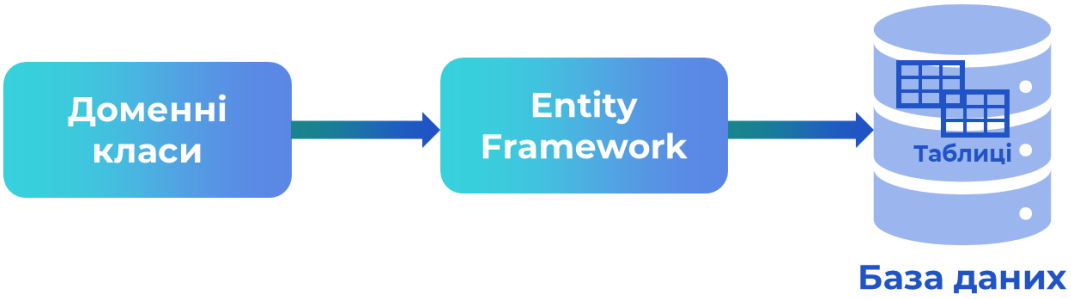


Рис. 5.1. Схема методу “Code First”

Для початку було описано сутності для бази даних. В проекті будуть представлені такі таблиці як Events, Categories, Subscriptions, SubscriptionEvents, EventCategories, NotificationTypeSubscriptions і також таблиці створені за допомогою Identity для збереження інформації про користувачів, таблиці необхідні для роботи Hangfire та для роботи Telegram бота.

ASP.NET Identity – це бібліотека керування користувачами Microsoft для ASP.NET. Вона включає в себе такі функції, як хешування паролів, перевірка пароля, зберігання користувачів і керування твердженнями. Зазвичай вона також має базову автентифікацію, використовуючи власні файли cookie та багатофакторну автентифікацію. У деяких конфігураціях він навіть може мати власний інтерфейс користувача [12].

В даному проекті було розроблено наступну схему бази даних:

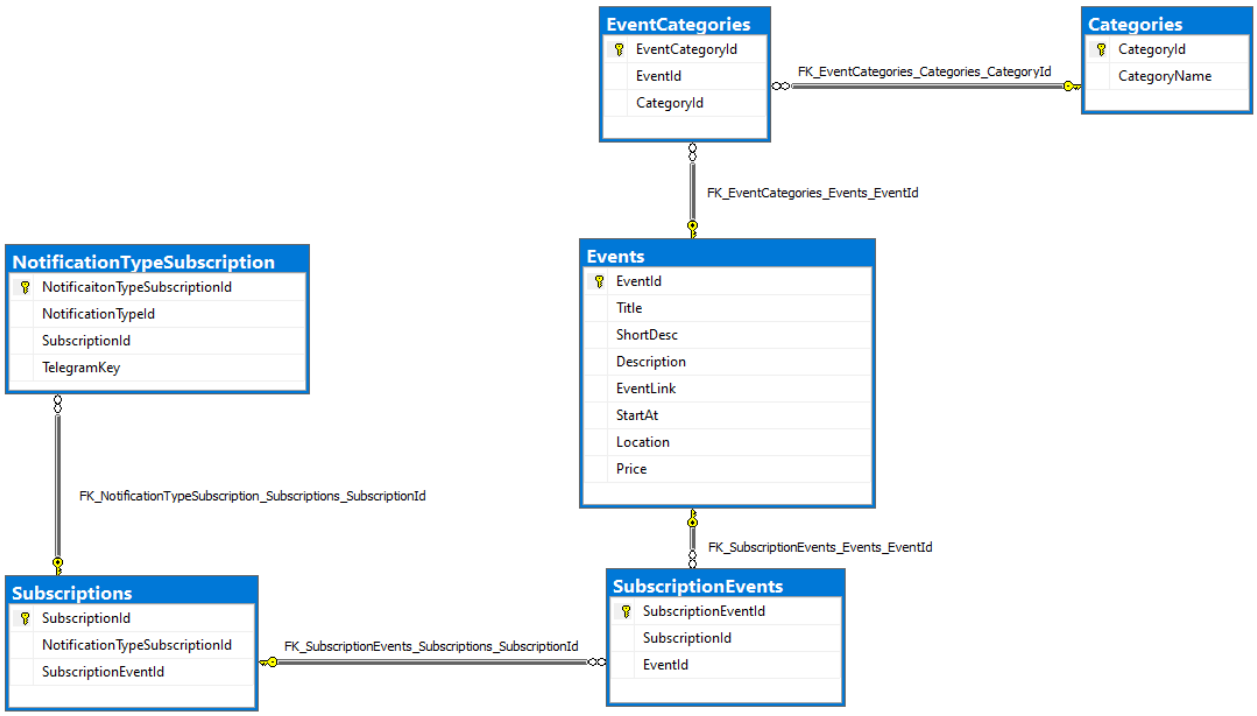


Рис. 5.2. Схема бази даних

Кожна подія в базі даних характеризується наявністю заголовку, короткого та розширеного опису, посилання на подію, у разі якщо є потокова трансляція на сервіс YouTube або в Google Meet тощо. Також подія характеризується датою та часом настання, місцем проведення та ціною квитка чи входу, якщо така встановлюється. Кожна подія має зв'язок із сутністю категорія, оскільки кожна подія може входити в певну категорію або в декілька. Внаслідок необхідності розбиття зв’язку багато-до-багатьох потрібна додаткова таблиця. Цією таблицею буде EventCategories.

Для збереження ідентифікатора користувача в соціальній мережі і для подальшого надсилання йому повідомлення в соцмережу було створено таблицю NotificationTypeSubscription. Також для обліку підписок було створено таблицю Subscriptions. А результатом зв’язку багато-до-багатьох між Subscriptions і Events є таблиця SubscriptionEvents, оскільки до окремої події може бути підписано багато користувачів і до однієї підписки можна поставити у відповідність багато подій.

5.2. Entity Framework Core

Entity Framework (EF) Core – це легка, розширювана, кросплатформна версія популярної технології доступу до даних Entity Framework.

EF Core може слугувати об’єктно-реляційним відображенням (O/RM) – бібліотекою коду, яка автоматизує передачу даних, що зберігаються в таблицях реляційної бази даних, в об’єкти, які частіше використовуються в коді програми. Таким чином це відображення:

- Дозволяє розробникам .NET працювати з базою даних за допомогою об’єктів .NET.
- Усуває потребу в більшості коду доступу до даних, який зазвичай потрібно написати.

EF Core підтримує багато рушіїв баз даних.

Основними перевагами є:

- Entity Framework допомагає скоротити час та витрати на розробку.
- Він надає автоматично згенерований код і дозволяє розробникам візуально проектувати моделі та відображати бази даних.
- Можливість легко відображати бізнес-об'єкти.
- Можливість легко виконувати швидкі операції CRUD в програмах .NET.

З EF Core доступ до даних здійснюється за допомогою моделі. Модель складається з класів сутностей і об’єкта контексту, який представляє сеанс з базою даних. Об’єкт контексту дозволяє робити запити та зберігати дані.

EF підтримує такі підходи до розробки моделі:

- Створення моделі з наявної бази даних.
- Написання коду моделі вручну відповідно до бази даних.
- Використання EF міграцій після створення моделі, щоб створити базу даних з моделі. Міграції дозволяють розвивати базу даних у міру зміни моделі [13].

В даному проекті було використано третій підхід – використання міграцій, з існуючих моделей для створення бази даних. Міграції - забезпечують спосіб поступового оновлення схеми бази даних, задля підтримки її синхронізації з моделлю даних програми, зберігаючи наявні дані в базі даних.

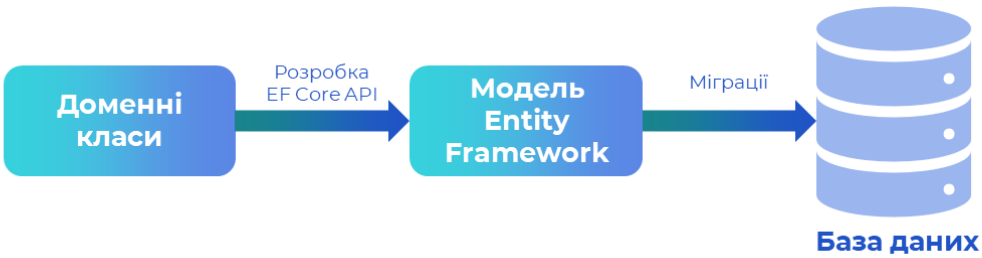


Рис. 5.3. Міграції в Entity Framework Core

Також було використано Language-Integrated Query (LINQ) – набір технологій, заснований на інтеграції можливостей запитів безпосередньо в мову C#. Запити LINQ – це першокласна мовна конструкція в C# .NET, як і класи, методи, події. LINQ забезпечує узгоджений досвід запитів до об'єктів запитів (LINQ to Objects), до реляційних баз даних (LINQ to SQL) і до XML (LINQ to XML).

LINQ (Language Integrated Query) – це єдиний синтаксис запиту в C# і VB.NET для отримання даних з різних джерел і форматів. Він інтегрований в C# або VB, тим самим усуваючи невідповідність між мовами програмування та базами даних, а також забезпечуючи єдиний інтерфейс запитів для різних типів джерел даних.

Наприклад, SQL – це мова структурованих запитів, яка використовується для збереження та отримання даних з бази даних. Таким же чином LINQ – це структурований синтаксис запитів, створений на C# та VB.NET для отримання даних із різних типів джерел даних, таких як колекції, ADO.Net DataSet, XML Docs, веб сервіс та MS SQL Server та інші бази даних.

5.3. Що таке багатошарова архітектура

Багатошарова архітектура – це архітектура програмного забезпечення, в якій різні програмні компоненти, організовані за шарами (рівнями), забезпечують певну виділену функціональність. Оскільки кожен шар є окремим, вносити зміни до кожного шару легше, ніж братися за всю архітектуру.

Найпростіша форма багатошарової архітектури – трирівнева архітектура. Вона складається з наступних рівнів:

- **Рівень презентації**: це перший і найвищий шар, який присутній у програмі. Цей рівень надає послуги презентації, тобто презентацію вмісту кінцевому користувачеві через графічний інтерфейс. Доступ до цього рівня

можна отримати через будь-який тип клієнтського пристрою, як-от настільний комп’ютер, ноутбук, планшет, мобільний телефон тощо. Для представлення вмісту важливо, щоб цей рівень взаємодіяв з іншими рівнями, які передують йому.

- **Рівень програми**: це середній рівень цієї архітектури. Це рівень, на якому виконується бізнес-логіка програми. Бізнес-логіка – це набір специфічних для програмного рішення функцій системи, що є характерними саме для неї. Компоненти цього рівня зазвичай працюють на одному або кількох серверах системи.
- **Рівень доступу до даних**: це найнижчий рівень цієї архітектури і в основному пов’язаний із зберіганням та отриманням даних програми. Дані програми зазвичай зберігаються на сервері баз даних, файловому сервері або будь-якому іншому пристрої чи носії, який підтримує логіку доступу до даних і забезпечує необхідні кроки для забезпечення доступу лише до даних без надання доступу до механізмів зберігання та пошуку даних. Це робиться на рівні даних шляхом надання API для рівня програми. Надання цього API забезпечує повну прозорість операцій з даними, які виконуються на цьому рівні, не впливаючи на рівень програми. Наприклад, оновлення або оновлення систем цього рівня не впливають на рівень додатків цієї архітектури [14].

На схемі нижче показано, як працює проста багатошарова архітектура з 3 рівнями:

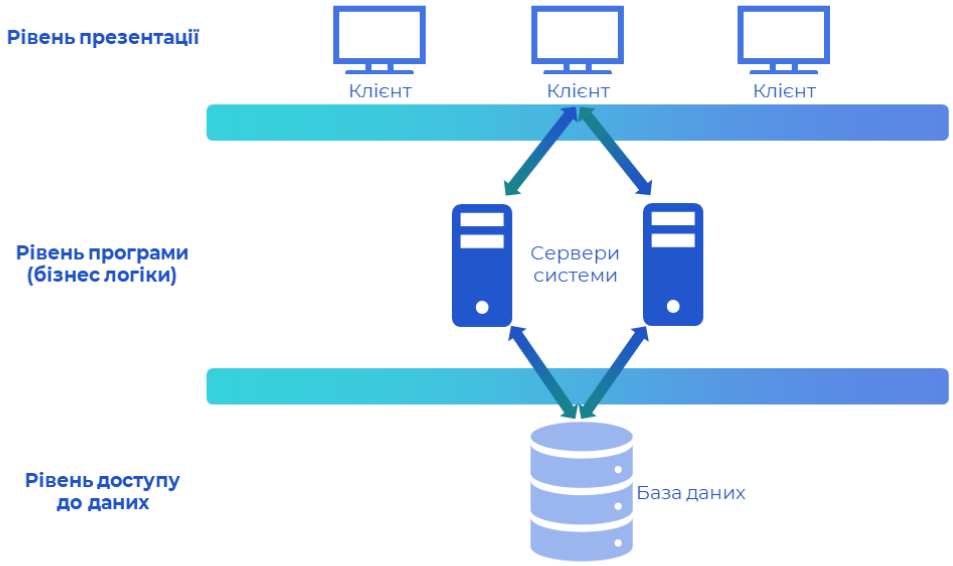


Рис. 5.4. Проста багатошарова архітектура

5.4. Переваги багатошарової архітектури

Багатошарова архітектура програмного забезпечення має ряд переваг, тому останніми роками вона стала настільки популярним архітектурним шаблоном. Найголовніше, що порівнева сегрегація дозволяє відповідно керувати та підтримувати кожен рівень. Теоретично це повинно значно спростити спосіб управління програмною інфраструктурою.

Багаторівневий підхід особливо добре підходить для швидкої та відносно безризикової розробки додатків веб-масштабу, виробничого рівня та розміщених у хмарі. Це також полегшує оновлення будь-яких застарілих систем – коли ваша архітектура розбита на кілька шарів, зміни, які потрібно внести, повинні бути простішими та менш масштабними, ніж вони могли б бути в іншому випадку. Різні команди можуть працювати над різними рівнями системи. Таким чином можна бути впевненим, що фахівці з дизайну та презентації працюють над рівнем презентації, а експерти з баз даних працюють на рівні даних.

Оскільки програма поділена на незалежні рівні, можна легко повторно використовувати кожен рівень для інших програмних проектів. Наприклад, якщо є потреба використовувати ту саму програму, але для іншого набору даних, можна просто відтворити рівні логіки та презентації, а потім створити новий рівень даних [16].

5.5. Реалізація багатошарової архітектури на проекті

Багатошарова архітектура в проекті представлена чотирма рівнями – рівнем відображення, бізнес логіки, доступу до даних та об’єктів передачі даних:

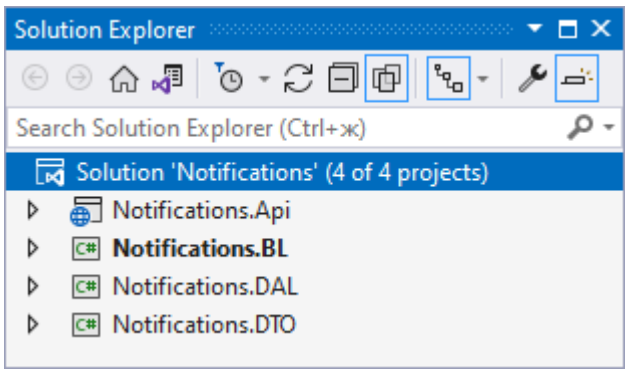


Рис. 5.5. Багаторівнева архітектура проекту

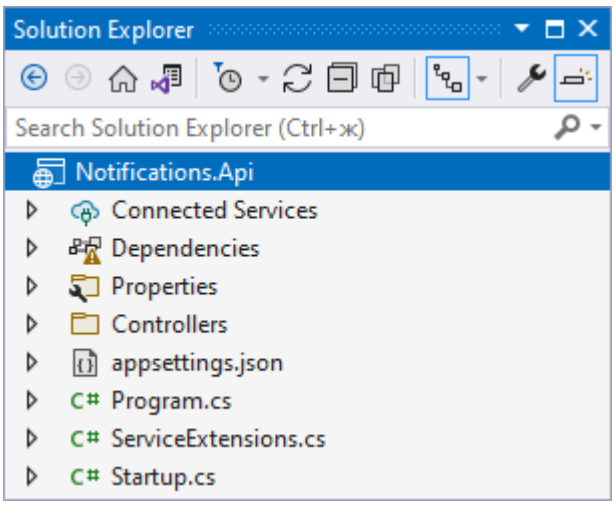


Рис. 5.6. Рівень презентації проекту

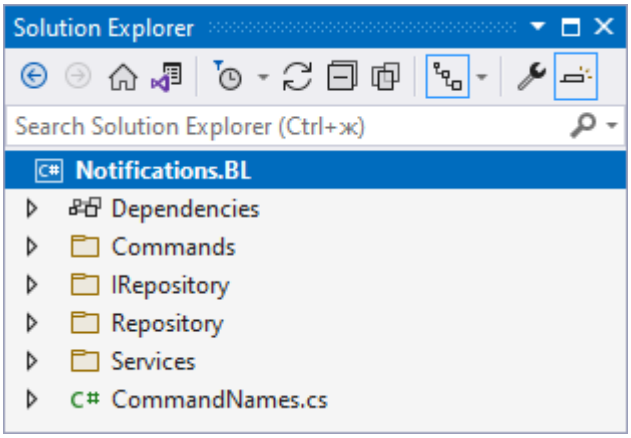


Рис. 5.7. Рівень бізнес логіки проекту

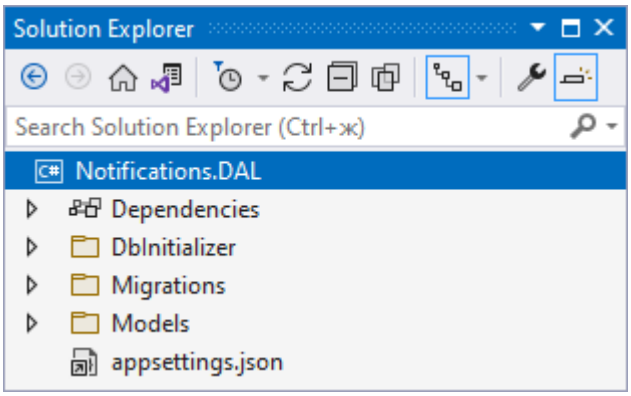


Рис. 5.8. Шар доступу до даних

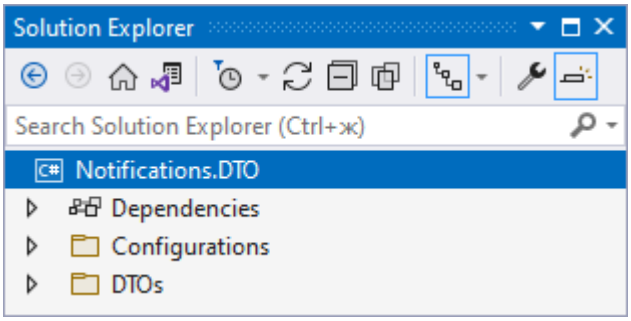


Рис. 5.9. Рівень об’єктів передачі даних

Для взаємодії з рівнем доступу до даних було використано шаблони проектування репозиторій (Repository) та концепт Unit of Work.

Репозиторій – це не що інше, як клас, визначений для сутності, з усіма можливими операціями для цієї конкретної сутності. Наприклад, репозиторій

для сутності Подія матиме основні операції CRUD та будь-які інші можливі операції, пов’язані з нею.

Шаблон репозиторію можна реалізувати такими способами:

- Один репозиторій на сутність (незагальний). Цей тип реалізації передбачає використання одного класу репозиторію для кожної сутності. Наприклад, якщо у вас є дві сутності Подія та Категорія, кожна сутність матиме власний репозиторій.
- Загальний репозиторій: загальний репозиторій – це той, який можна використовувати для всіх сутностей, іншими словами, він може використовуватися для Події, Категорії або будь-якої іншої сутності.

Unit of Work визначають як окрему транзакцію, яка включає в себе декілька операцій вставки/оновлення/видалення тощо. Простими словами, це означає, що для певної дії користувача (скажімо, реєстрації на веб сайті), усі транзакції, як-от вставка/оновлення/видалення тощо, виконуються в одній транзакції, а не в кількох транзакціях з базою даних. Це означає, що Unit of work тут включає операції вставки/оновлення/видалення, все в одній транзакції.

Додаток А

5.6. Висновки

Отже, архітектуру даного рішення представляють:

- База даних. Вона складається із сутностей “Event”, “Category”, “User” та інших необхідних для роботи системи сутностей. Подія може асоціюватися з однією або багатьма категоріями. Вона має назву, опис, дату проведення та іншу важливу інформацію. В свою чергу з категорією може асоціюватися багато подій. Для обліку користувачів і була створена база даних користувачів.

- Серверна частина. Вона складається з програмного застосунку на платформі ASP.NET Core. В свою чергу цей застосунок реалізований за принципом багатошарової архітектури, тобто розділений на рівні, де кожен рівень організовує та виконую відповідну частину функціоналу серверної частини: рівень презентації, доступу до даних, бізнес логіки та об’єктів для передачі даних.

РОЗДІЛ 6
ОПИС КОДУ ТА ІНТЕРФЕЙСУ ПРОГРАМИ

Розробка функціоналу для клієнтської та серверної частини орієнтуються на наступну схему:

Рис. 6.1. Архітектура функціоналу проекту

6.1. Серверна частина

Серверна частина (Back-End) – представлена у вигляді реалізації програмного застосунку на базі ASP.NET.

6.1.1. ASP.NET Core

ASP.NET Core – це безкоштовний веб фреймворк з відкритим вихідним кодом, розроблений Microsoft. Він надає функції, які дозволяють створювати

Back-End для сучасних веб-додатків, а також веб-API. Мовою програмування, яка використовується для розробки ASP.NET Core, є C# або будь-яка інша мова програмування на основі .NET.

Переваги ASP.NET Core:

- Легкий високопродуктивний веб-фреймворк.
- Інтеграція сучасної платформи інтерфейсу користувача - ASP.NET Core підтримує сучасні фреймворки для клієнтської частини, як-от AngularJs, ReactJs та React with Redux тощо.
- Має можливість розміщення на IIS, Apache, Docker або Self Hosting.
- Кросплатформенний – веб-додаток ASP.NET Core може працювати на інструментах розробки Windows, Mac, Linux.
- Підтримка вбудованої ін’єкції залежностей (Dependency Injection)
- Open-Source – це веб-фреймворк з відкритим вихідним кодом і орієнтований на спільноту розробників.
- Послідовне керування версіями програм – ASP.NET Core працює на .NET Core, який підтримує одночасний запуск кількох версій програм.
- Уніфікована база для створення веб-інтерфейсу та веб-API.

Web API – це інтерфейс програмування, який забезпечує зв’язок або взаємодію між програмними додатками. Web API часто використовується для надання інтерфейсу для веб-сайтів і клієнтських програм для доступу до даних. API можна використовувати для доступу до даних із бази даних та збереження даних назад у базу даних.

Зокрема для побудови серверної частини було використано фреймворк ASP.NET Web API. Він спрощує створення веб-сервісу HTTP, що охоплює велике коло клієнтів, включаючи браузер, мобільні додатки, настільні програми та інтернет речей (IoT).

6.1.2. Рівень Notifications.Api

Рівень Notifications.Api містить в собі директорію з контролерами, які обробляють запити користувача відповідно визначеним чином, здійснюють валідацію даних **(**перевірку їх відповідності згідно поставлених вимог**),** надсилають відповідь з даними та статус обробки запиту.

Наступна таблиця описує контролери та їх основні передбачені методи.

Табл. 6.1. Контролери рівня представлення

Контролер	Методи
AccountController	Містить методи для авторизації користувача системи. Відбувається за допомогою Google.
CategoryController	Містить методи для створення, перегляд, отримання категорії за ідентифікатором, оновлення, видалення категорії. Методи для підписки та відписки на категорію, перегляду подій, що відносяться до категорії.
EventController	Містить методи для створення, перегляд, отримання категорії за ідентифікатором, оновлення, видалення події. Методи для підписки та відписки на подію, додавання події до категорії, пошуку за назвою, описом, датою проведення, виведення списку подій, на які користувач підписався.
NotificationController	Містить методи для надсилання сповіщень користувачам про настання події, у разі наближення до дати та часу її проведення.

Продовження табл. 6.1.

TelegramBotController	Містить методи для обробки запитів користувача на отримання інформації та механізму підписки.

Serilog.

Логування загалом є хорошою практикою, оскільки воно дозволяє вести облік виконаних дій в програмі, часу їх виконання, момент виконання, результат виконання. Таким чином можна відслідковувати виключення та інші проблеми як при розробці програмного забезпечення так і на етапі підтримки. В проекті використовується логування за допомогою бібліотеки Serilog.

З кожним виконаним запуском застосунку чи запитом у папку Logs записуються результат виконаних дій, таким чином формуються файли з обліком всіх подій системи та їх результати.

```
var host = CreateHostBuilder(args).Build();

Log.Logger = new LoggerConfiguration()
    .WriteTo.File(
        path: $"{Environment.CurrentDirectory}\\Logs\\log-.txt",
        outputTemplate: "{Timestamp:dd-MM-yyyy HH:mm:ss.fff zzz} [{Level:u3}] {Message:lj}{NewLine}{Exception}",
        rollingInterval: RollingInterval.Day,
        restrictedToMinimumLevel: LogEventLevel.Information
    )
    .WriteTo.Console().CreateLogger();
try
{
    Log.Information(messageTemplate: "Notifications App Is Starting");
    SeedDatabase(host);
    host.Run();
}
catch (Exception ex)
{
    Log.Fatal(ex, messageTemplate: "Notifications App failed to start");
}
finally
{
    Log.CloseAndFlush();
}
```

Рис. 6.2. Приклад використання логування Serilog

Swagger в проекті налаштовується на рівні відображення. Таким чином можемо отримати зручний користувацький інтерфейс для виконання запитів.

Додаток Б

Hangfire в проекті також конфігурується на рівні відображення та використовується в контролері NotificationController.

В результаті встановлення можна перейти на панель керування відкладеними та іншими роботами:

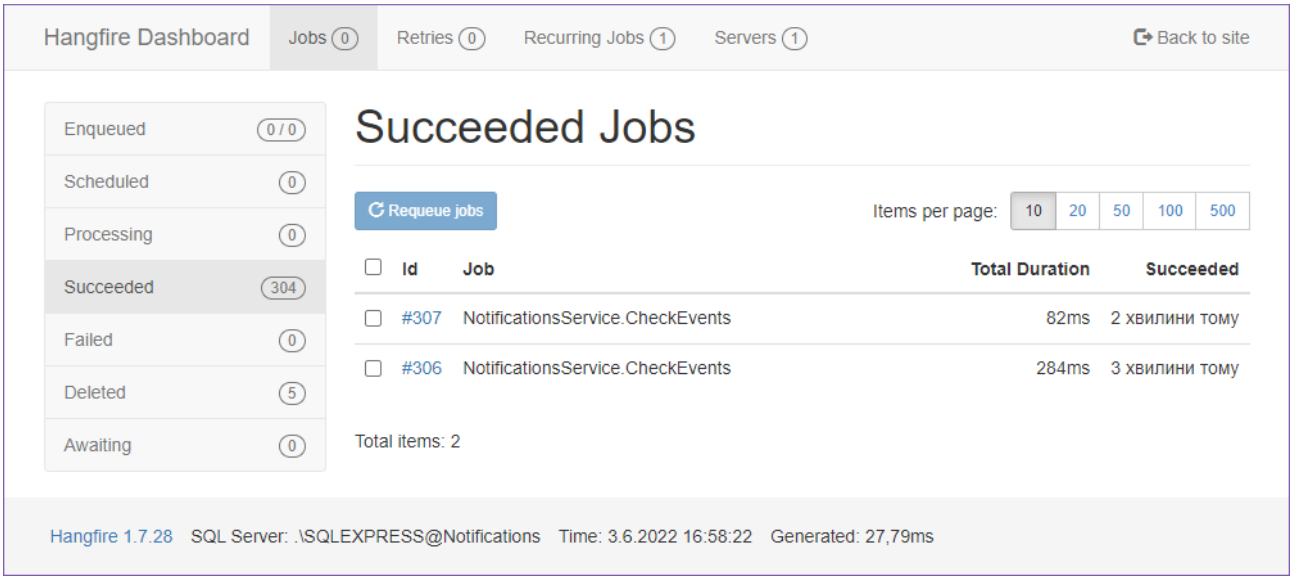


Рис. 6.3. Панель керування Hangfire

6.1.3. Рівень Notifications.BL

Рівень Notifications.BL містить бізнес логіку – сервіси, що є унікальними саме для даного проекту. Робота з базами даних, логування чи інші схожі функції зазвичай присутні у всіх застосунках, а ось надсилання сповіщень про настання певної події користувачу в соціальну мережу або виведення списку подій, на які підписаний користувач є притаманними саме цьому проекту.

В директорії Services знаходиться клас NotificationsService який і відповідає за механізм підписки чи відписки на подію чи категорію в цілому,

додавання категорії до певної події тощо. Також в директоріях IRepository та Repository містяться інтерфейси та реалізації патерну репозиторій та класу Unit of Work. Цей патерн забезпечує абстракцію даних, таким чином програма може працювати з абстракцією даних у вигляді представлення колекції цих даних. Додавання, видалення, оновлення та вибір елементів із цієї колекції здійснюється за допомогою низки простих методів, без необхідності мати справу з підключенням до бази даних чи іншими командами.

В директорії Commands знаходяться команди на отримання інформації з Telegram боту:

- Абстрактний базовий клас для всіх команд;
- Клас для початкової команди боту;
- Отримання довідки про команди;
- Отримання однієї або декількох подій;
- Отримання категорії;
- Отримання подій, на які підписаний користувач;
- Підписка на події та відписка;

6.1.4. Рівень Notifications.DAL

Рівень Notifications.DAL містить реалізовані моделі (сутності бази даних) програми та опис взаємозв’язків між ними, ініціалізатор бази даних, що дозволяє заповнити базу даних певним початковим набором даних та міграції, для оновлення схеми бази даних.

В директорії Models знаходяться всі моделі системи: ApplicationUser, Category, Event, EventCategory, NotificationsContext, NotificationTypeSubscription, RequestParams, Subscription, SubscriptionEvent.

NotificationsContext - клас, який представляє сесію з базою даних, використовуючи яку можна працювати з нею. Також в цьому класі описуються зв’язки між сутностями та інші параметри. RequestParams це спеціальна модель, яка містить необхідну інформацію для пагінації – формування послідовності сторінок із спільним контентом, наприклад сторінками подій.

В директорії Telegram знаходяться необхідні для роботи бота моделі, зокрема для збереження інформації про користувача, або мати змогу в подальшому проводити операції отримання даних, підписки та сповіщення.

6.1.5. Рівень Notifications.DTO

Рівень Notifications.DTO (Data Transfer Object, фактично шаблон проектування для передачі даних між рівнями) – рівень з об’єктами, які надсилаються на клієнтську частину внаслідок обробки запитів, або які використовуються для внутрішньої обробки між рівнями застосунку.

Основною перевагою використання такого підходу полягає в тому, щоб точніше контролювати, які властивості ми відкриваємо для користувача, а також показувати сукупності об’єктів, а не окремі сутності.

В ньому ми маємо директорію з конфігураціями для мапінгу (процесу узгодження, коли дані одного набору збігаються з даними іншого набору). Таким чином ми фільтруємо дані, що отримала система від користувача і після їх перевірки обробляємо і надсилаємо до бази даних. В директорії DTOs власне і розміщенні об’єкти з якими буде взаємодіяти користувач системи.

Для конфігурації мапінгу було використано бібліотеку AutoMapper.

AutoMapper в C# - це бібліотека, яка використовується для відображення даних від одного об’єкта до іншого. Він діє як сервіс для узгодження

властивостей та іншої інформації між двома об’єктами і перетворює один тип об’єкта в інший.

До появи AutoMapper, якщо розробник хотів призначити одну властивість об’єкту іншій властивості об’єкта, він виконував довгу процедуру. Він повинен був відобразити кожну властивість цих двох різних об’єктів. Припустимо, що в одному класі ми маємо 30 властивостей, і ми хочемо зіставити це з іншим класом, що має 30 властивостей, тоді нам доведеться відобразити цю властивість один за одним 30 разів.

Додаток Г

В результаті встановлення конфігурації можна використати вбудований метод бібліотеки для автоматичного мапінгу одного класу в інший.

6.2. Клієнтська частина

Клієнтська частина – представлена у вигляді застосунку написаному за допомогою бібліотеки ReactJS. Основною перевагою цієї бібліотеки є багаторазові компоненти, які ми можемо повторно використовувати, тим самим зменшуючи зусилля на розробку та забезпечуючи легший та простіший потік роботи.

Проект на React JS складається з компонентів та ресурсів для відображення клієнтської частини. Компоненти – функціональні елементи написані на мові JavaScript. Вони в свою чергу використовують ресурси та інші файли для відображення інформації, її оновлення, надсилання запитів клієнтської частини та взаємодії користувача із застосунком. Взаємодію з серверною частиною вони виконують за допомогою Axios - бібліотекою, яка використовується для створення HTTP-запитів, зокрема на Back-End. Таким чином в проекті на React нам знадобиться отримати дані із зовнішнього джерела.

Всі частини ReactJS та саму бібліотеку описуємо детальніше в наступних пунктах.

6.2.1. React JS

React JS – це безкоштовна бібліотека JavaScript із відкритим вихідним кодом для створення інтерфейсів користувача на основі компонентів інтерфейсу користувача. Його підтримують Meta (Facebook) і спільнота окремих розробників і компаній. React можна використовувати як базу для розробки односторінкових , мобільних або серверних програм. Однак React займається лише керуванням станом і відтворенням цього стану в DOM, тому створення додатків React зазвичай вимагає використання додаткових бібліотек для маршрутизації, а також певної функціональності на стороні клієнта.

Чому варто обирати React?

React є надзвичайно гнучким. Вивчивши його, ми маємо змогу використовувати його на різноманітних платформах для створення якісних інтерфейсів користувача.

React.js надає пакет create-react-app , який дозволяє миттєво почати розробку. Він забезпечує швидку розробку та невеликий API. API React дуже простий у освоєнні. У ньому дуже мало концепцій для вивчення.

React має підтримку та ресурси Facebook. React активно використовується в додатку Facebook, на веб-сайті та в Instagram. Вони використовують понад 50 тисяч компонентів React у своєму виробничому середовищі. Чотири найкращі співробітники React на GitHub є штатними співробітниками Facebook. Тому React є досі актуальним, через його безперервну підтримку та розширення функціоналу.

Під час розробки, ви можете використовувати пакет React NPM. NPM – це скорочення від *node package manager*, онлайн-каталогу, який містить різні вже зареєстровані бібліотеки з відкритим кодом. Модулі NPM використовують різні функції, коли вони встановлені в програму за допомогою команди NPM npm install. Це забезпечує вам доступ до різноманітних бібліотек, що в свою чергу пришвидшують розробку проекту.

React має чудову продуктивність. Команда React зрозуміла, що JavaScript швидкий, але оновлення DOM робить його повільним, і вони знайшли найефективніший та найрозумніший спосіб оновлення DOM. До React більшість фреймворків і бібліотек оновлювали DOM нерозумно, щоб відображати новий стан. Це призвело до змін значної частини сторінок.

React відстежує значення стану кожного компонента за допомогою Virtual DOM . Коли стан компонента змінюється, React порівнює існуючий стан DOM з тим, як має виглядати новий DOM. Після цього він знаходить найшвидший спосіб оновлення DOM. Проста модель програмування React дозволяє йому автоматично змінювати стан, коли дані оновлюються. Це відбувається в пам'яті, тому це швидко.

Розмір бібліотеки React також невеликий. Це менше 6 КБ. Це значно менше, ніж у інших бібліотеках та фреймворках.

React – чудовий інструмент для створення інтерактивних програм для мобільних, веб- та інших платформ. Популярність і використання React зростають з кожним днем не просто так. Як розробнику, кодування в React покращує ваші навички JavaScript, мову, на яку сьогодні припадає майже 90% веб-розробки.

6.2.2. Архітектура клієнтської частини, компоненти

Основною структурною одиницею в React є компонент – блок додатку. Це може бути що завгодно – спливаюче вікно, кнопка, меню, уривок тексту або контейнер, щоб утримувати все це всередині та передавати деякі дані. Необхідно просто надати React інформацію про те, як мають виглядати всі компоненти залежно від конкретних параметрів і він подбає про решту внесених відповідних змін до подання відповідно до змін у стані. Це називається «декларативне програмування», що прискорює та полегшує розробку навіть найскладніших інтерфейсних програм. React також дуже ефективний у внесенні змін до уявлень, оскільки він віртуалізує вміст усієї програми і спочатку вносить зміни до цього віртуального «подання», а потім визначає найкращий і найшвидший спосіб відобразити зміни на екрані.

Тож ефективнішим способом побудови архітектури веб додатку є поділ його на окремі компоненти, які можна використовувати повторно в будь-якій частині додатку просто імпортувавши.

Архітектура клієнтської частини виглядає наступним чином:

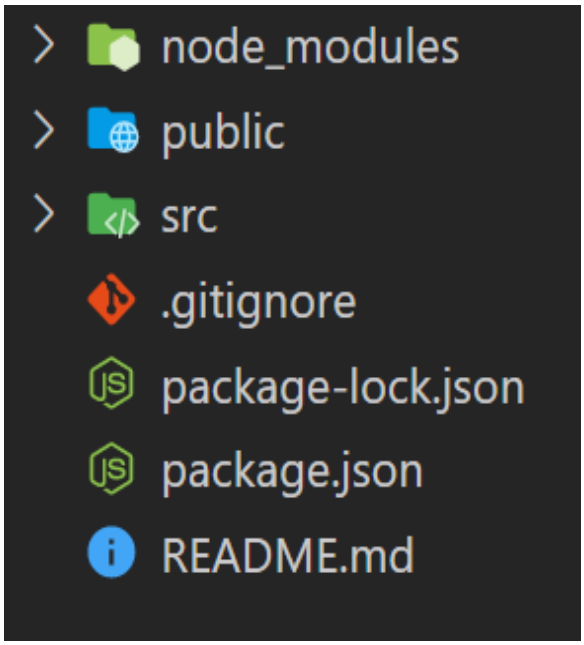


Рис. 6.4. Структура Front-End

Головні папки з якими ми будемо працювати - це **public** та **src**.

У папці **public**, ми зберігаємо файл index.html. Сюди ж ми можемо додати свої шрифти, якщо ми не хочемо використовувати ті, що за замовчуванням. Наш шрифт - “Aeroport”.

У папці **src** ми будемо зберігати всі наші компоненти, з яких буде складатись наш веб-додаток. Є сенс розділити їх по папкам. У папці Admin зберігаються компоненти, які будуть використані для відображення адмін панелі. У папці Events зберігаються компоненти для сторінок виводу всіх подій та окремої події. Папка MainPage містить компоненти, що відображаються на головній сторінці. Папка shared містить компоненти, які ми повторно використовуємо, в нашому випадку - це Header та Footer. Папка utils містить додаткові функціональні комопоненти, в нашому випадку це компонент для перевірки даних користувача при вході в адмін панель. Папка context містить компонент AuthContext, що в свою чергу зберігає дані про користувача, що увійшов у систему.

Файл App.js відповідає за наші маршрути та при яких умовах виводити, той чи інший компонент. Файл App.css відповідає за стилі по всьому веб-додатку. Інші файли є необхідними для підтримки коректної роботи.

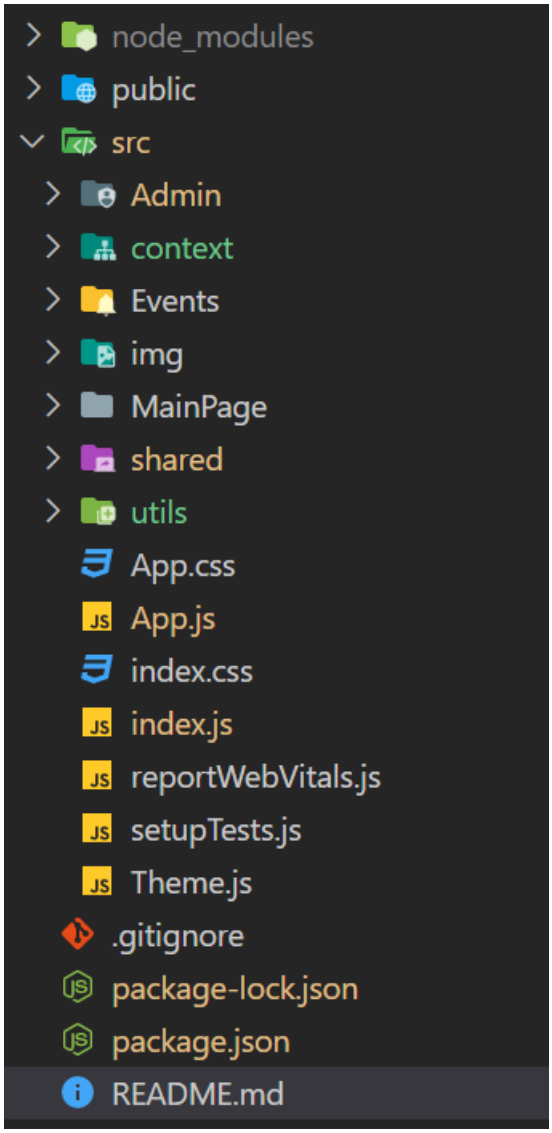


Рис. 6.5. Вміст папок Front-End частини

Перейдемо до опису окремих компонентів:

Головна сторінка складається з наступних компонентів:

- AllEventsMainPage
- BeNotificated
- Faculties
- FilterEventsByDate
- IntroMainPage

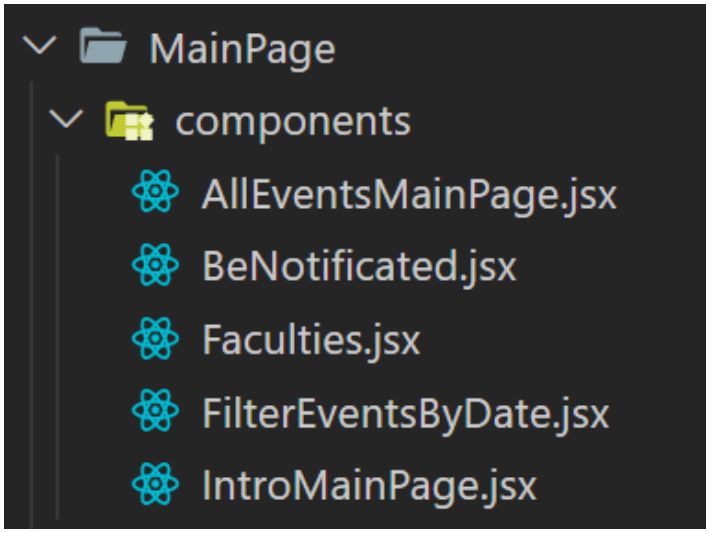


Рис. 6.6. Компоненти головної сторінки

Табл. 6.2. Компоненти головної сторінки

Назва компоненту	**Опис**
AllEventsMainPage	Даний компонент виводить події, які користувач, може переглянути у скороченому вигляді, та в разі зацікавленості перейти до конкретної події
BeNotificated	Даний компонент відповідає за візуалізацію кроків, які необхідно пройти користувачеві для отримання сповіщень про події
Faculties	Даний компонент представляє собою таблицю, що дає змогу користувачеві перейти за окремим факультетом та переглянути актуальні події для нього

Продовження табл. 6.2.

FilterEventsByDate	Даний компонент дозволяє користувачеві переглянути події, що будуть відбуватись сьогодні, завтра, цього тижня та цих вихідних. Всі дані представлені у короткому форматі
IntroMainPage	Даний компонент представляє собою візуалізацію мети нашого застосунку “ОА - всі події в одному місці”

Подання цілої сторінки реалізоване в окремому компоненті MainPage, де ми імпортуємо всі вищезгадані компоненти.

Також на цій сторінці ми виводимо ще три компоненти, що знаходяться у папці **shared**, з приводу частого використання:

Табл. 6.3. Спільні компоненти Front-End частини

Header	Даний компонент надає коротку навігацію по веб-додатку та надає можливість користувачеві шукати подію, підписатись на нотифікації та переглянути події за датами
UpcomingEvents	Даний компонент виводить події, що відбудуться найближчим часом

Продовження табл. 6.3.

Footer	Даний компонент представляє собою візуалізацію слогану нашого застосунку “ОА - тут не нудно”

Компоненти Events:

- Events
- EventItem
- Pagination
- SearchEvents

Табл. 6.4. Компоненти Events

Назва компоненту	**Опис**
Events	Даний компонент надсилає запит на отримання масиву усіх подій та за допомогою методу map передає дані у наступний компонент для відображення окремої події
EventItem	Даний компонент відповідає за відображення окремої події
Pagination	Даний компонент представляє собою функціонал для реалізації пагінаціїї (поділу всіх подій на сторінки)
SearchEvents	Даний компонент відповідає за змогу користувачеві шукати події за назвою або описом

Вивід та подання усіх вищезгаданих компонент реалізоване у компоненті AllEvents – сторінка для відображення всіх подій з пошуком та пагінацією та EventDetails – сторінковий компонент для відображення усіх деталей про конкретну подію.

Компоненти Admin:

- AdminEvents
- AdminEventItem
- AdminPanel
- AdminLogin
- CreateEvent
- EditEvent
- CategoriesPanel

Табл. 6.5. Компоненти Admin

Назва компоненту	**Опис**
AdminEvents	Даний компонент надсилає запит на отримання масиву усіх подій та за допомогою методу map передає дані у наступний компонент для відображення окремої події
AdminEventItem	Даний компонент відповідає за відображення окремої події на адмін панелі
AdminPanel	Даний компонент є сторінковим та відображає усі події з можливістю пошуку та пагінацією

Продовження табл. 6.5.

AdminLogin	Даний компонент відповідає за змогу користувачеві увійти у систему, як адміністратору
CreateEvent	Даний компонент реалізує форму з валідацією для створення події та надсилає запит на серверну частину
EditEvent	Даний компонент реалізує форму з валідацією для релагування події та надсилає запит на збереження даних
CategoriesPanel	Даний компонент відображає усі наявні категорії для подій та дає змогу створювати нові категорії або видаляти наявні

6.2.3. React Hooks

Хуки – це нове доповнення в React 16.8. Вони дозволяють використовувати стан та інші функції React без написання класу.

Хуки вирішують широкий спектр, здавалося б, не пов’язаних проблем у React, з якими розробники зіткнулися протягом п’яти років написання й підтримки десятків тисяч компонентів, зазвичай це проблема важкого розуміння складних компонентів. За допомогою Hooks можливо витягти логіку стану з компонента, щоб його можна було незалежно тестувати та повторно використовувати. Хуки дозволяють повторно використовувати логіку стану без зміни ієрархії компонентів.

Якщо у компоненті стає зрозуміло, що потрібно додати до нього якийсь стан, раніше потрібно було перетворити його в клас. Тепер можливо використовувати Hook всередині існуючого компонента.

Табл. 6.6. Хуки в проекті

Назва хуку	**Опис**
useState	Дозволяє нам відстежувати стан у компоненті функції
useEffect	Дозволяє виконувати побічні ефекти у компонентах: отримання даних, безпосереднє оновлення DOM та таймери. useEffect приймає два аргументи. Другий аргумент необов'язковий. useEffect(<function>, <dependency>)
useForm	Використовується для легкого керування формами
useContext	Дозволяє глобально керувати станом, який можна передавати у всі компоненти. Наприклад, для збереження даних про користувача, що увійшов у систему

Варто відзначати, що в React JS для керуванням стану також існує Redux. Головна відмінність Redux від хуків – це збереження стану глобально, в той час

як хуки використовують для збереження та керуванням стану в одній або декількох компонентах. На рівні даного проекту, було достатньо використовувати хуки.

6.2.4. Axios

Axios – це клієнтська бібліотека HTTP, яка дозволяє робити запити до заданої кінцевої точки. Це може бути, наприклад, зовнішній API або власний сервер.

Існує ряд різноманітних бібліотек, які можна використовувати для виконання цих запитів, але в Axios є ряд переваг:

1. Він має хороші налаштування за замовчуванням для роботи з даними JSON. На відміну від альтернатив, таких як Fetch API, вам часто не потрібно встановлювати заголовки. Або виконайте виснажливі завдання, як-от перетворення тіла запиту в рядок JSON.
2. Axios має назви функцій, які відповідають будь-яким методам HTTP. Щоб виконати запит GET, ви використовуєте .get() метод.
3. Axios робить більше з меншою кількістю коду. На відміну від Fetch API, вам потрібен лише один .then() зворотний виклик, щоб отримати доступ до запитаних даних JSON.
4. Axios має кращу обробку помилок. Axios видає для вас 400 і 500 помилок діапазону. На відміну від Fetch API, де ви повинні перевірити код статусу та викинути помилку самостійно.[18]

6.2.5. React Routes

React Router – це повнофункціональна бібліотека маршрутизації на стороні клієнта і сервера для React, бібліотеки JavaScript для створення інтерфейсів користувача. React Router працює скрізь, де працює React.

Деякі компоненти React Router:

BrowserRouter: є рекомендованим інтерфейсом для запуску React Router у веб-браузері. BrowserRouter зберігає поточне місцезнаходження в адресному рядку браузера за допомогою чистих URL-адрес і здійснює навігацію за допомогою вбудованого стека історії браузера. [19]

Routes і **Route** є основними способами відтворення компонентів у React Router на основі поточного location. Route є своєрідним if: якщо path відповідає поточній URL-адресі, він відображає його element - нашу компоненту. Параметр caseSensitive визначає, чи слід виконувати відповідність з урахуванням регістру (за замовчуванням false).

Щоразу, коли місце розташування змінюється, Routes переглядає всі його children Route елементи, щоб знайти найкращу відповідність, і відтворює цю гілку інтерфейсу користувача.

Окрім маршрутів бібліотека React Router реалізує функціонал ще деяких часто використовуваних елементів:

Link – це елемент, який дозволяє користувачеві переходити на іншу сторінку, натиснувши на неї. У react-router-dom, Link відображає доступний <a> елемент із дійсним href, шляхом, що вказує на шлях, на який він посилається.

NavLink – це особливий тип, Link який містить інформацію, чи є він «активним». Це корисно під час створення навігаційного меню, такого як ланцюжок або набір вкладок, де ми хочемо показати, яка з них вибрана в даний момент. Він також надає корисний контекст для допоміжних технологій, таких як програми зчитування з екрана.

Елемент **Navigate** змінює поточне розташування під час візуалізації. Це обгортка компонента useNavigate, яка приймає всі ті самі аргументи, що й props. Це надає нам змогу «перекидати» користувача на інші сторінки, наприклад після успішного входу в систему, користувач автоматично опиняється у адмін панелі.

6.2.6. Стиліазція, MUI

MUI – це фреймворк CSS, який відповідає вказівкам Google Material Design. Він був розроблений з нуля, щоб бути швидким і зручним для розробників.

MUI надає базовий набір компонентів і допоміжних методів, які розробники можуть використовувати для створення швидких і зручних для користувачів застосунків. Це вже готові рішення, що бережуть час ждя розробки застосунків. MUI надали нашому проекту готові форми, кнопки, іконки, контейнери.

Крім цього, MUI надає змогу налаштувати їхню вже існуючу тему.
Ви маєте змогу змінити системні кольори, типографіку та багато іншого.
Тема визначає колір компонентів, рівень тіні, непрозорість елементів тощо.

Теми дозволяють застосувати послідовний тон до програми. Це дозволяє налаштувати всі аспекти дизайну проекту, щоб задовольнити конкретні потреби бізнесу або бренду. Щоб забезпечити більшу узгодженість між додатками, доступні також світлі та темні типи тем. За замовчуванням компоненти використовують тип світлої теми.

Всі налаштування доступні у файлі Theme.js. Головним кольором MUI по замовчуванню є блакитний, у нашому застосунку – чорний. MUI дозволяє змінити палітру кольорів наступним чином:

```
palette:{
    background:{
        default: '#fff'
    },
    primary:{
        main: '#000',
    },
    secondary:{
        main: '#fff',
    }
```

Рис. 6.7. Зміна палітри кольорів

Таким чином ми маємо змогу використовувати елементи вже з готовим дизайном та змінювати деякі властивості стилей у відповідності до нашого дизайну.

6.3. Telegram Bot

6.3.1. Bot API

Для того, щоб взаємодіяти із телеграм-ботом через API, необхідно встановити відповідний пакет NuGet:

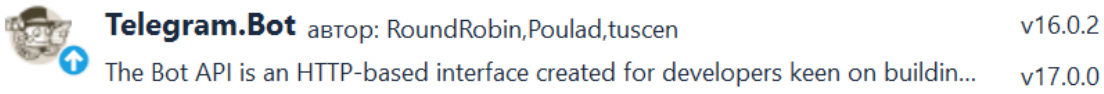


Рис. 6.8. Пакет NuGet для роботи з API телеграм-бота

Даний пакет представляє собою HTTP-інтерфейс для роботи з ботами в телеграмі.

6.3.2. Контролер

Щоб обробляти запити від користувача, які зберігаються на веб-сервері потрібний контролер, який в свою чергу містить метод, що приймає ці запити у вигляді об’єкта «update».

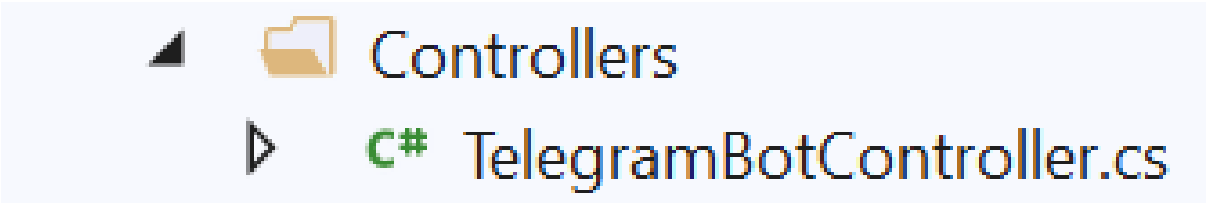


Рис. 6.9. Контролер телеграм-бота

6.3.3. Сервіси

Далі, необхідно зробити сервіси, які міститимуть методи щоб взаємодіяти із користувачем і запитами у вигляді команд.

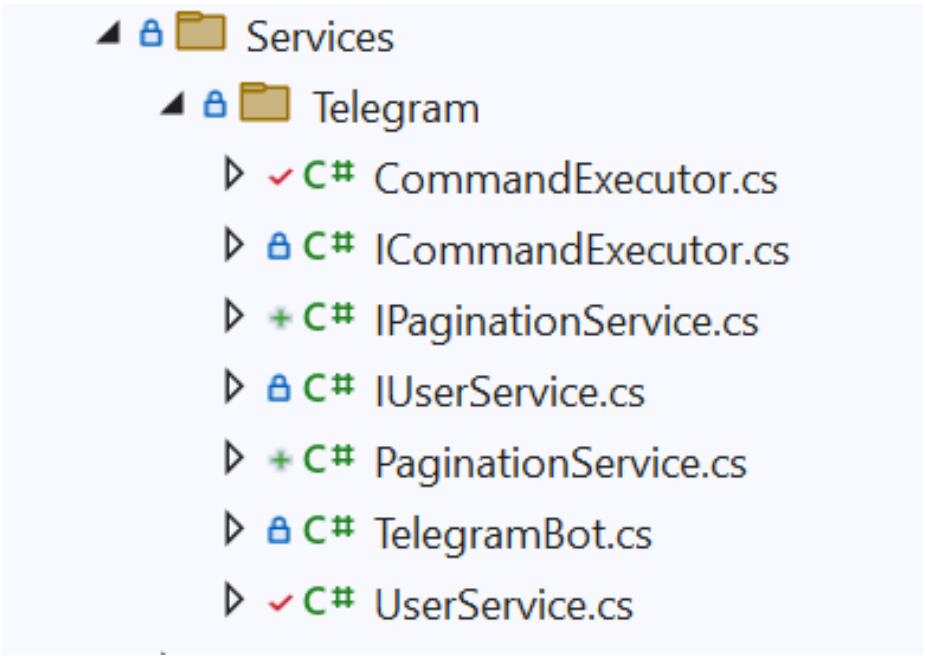


Рис. 6.10. Сервіси

В даному випадку маємо три основних сервіси:

- CommandExecutor – сервіс, який наслідує інтерфейс ICommandExecutor і містить методи для взаємодії між запитами користувача та відповідями зі сторони сервера. Тобто він визначає коли і яку команду потрібно виконувати.
- UserService – сервіс, що наслідує інтерфейс IUserService та містить логіку реєстрації нового користувача, а також отримання користувача, якщо такий вже є в базі даних. Це відбувається для того, щоб сервер розумів, що в даному випадку він працює з тим самим користувачем, а не реєстрував кожний раз нового.
- PaginationService – сервіс, що наслідує інтерфейс IPaginationService і призначений для реалізації механізму пагінації в боті шляхом тимчасового зберігання даних в оперативній пам’яті на основі хеш-таблиці за допомогою сервісу кешування даних Memcached, а також їх вилучення для подальшої роботи з ними.
- TelegramBot – сервіс, для створення та отримання телеграм-бота.

6.3.4. Команди

Для того, щоб бот виконував конкретні дії у відповідь на запити користувача потрібні команди.

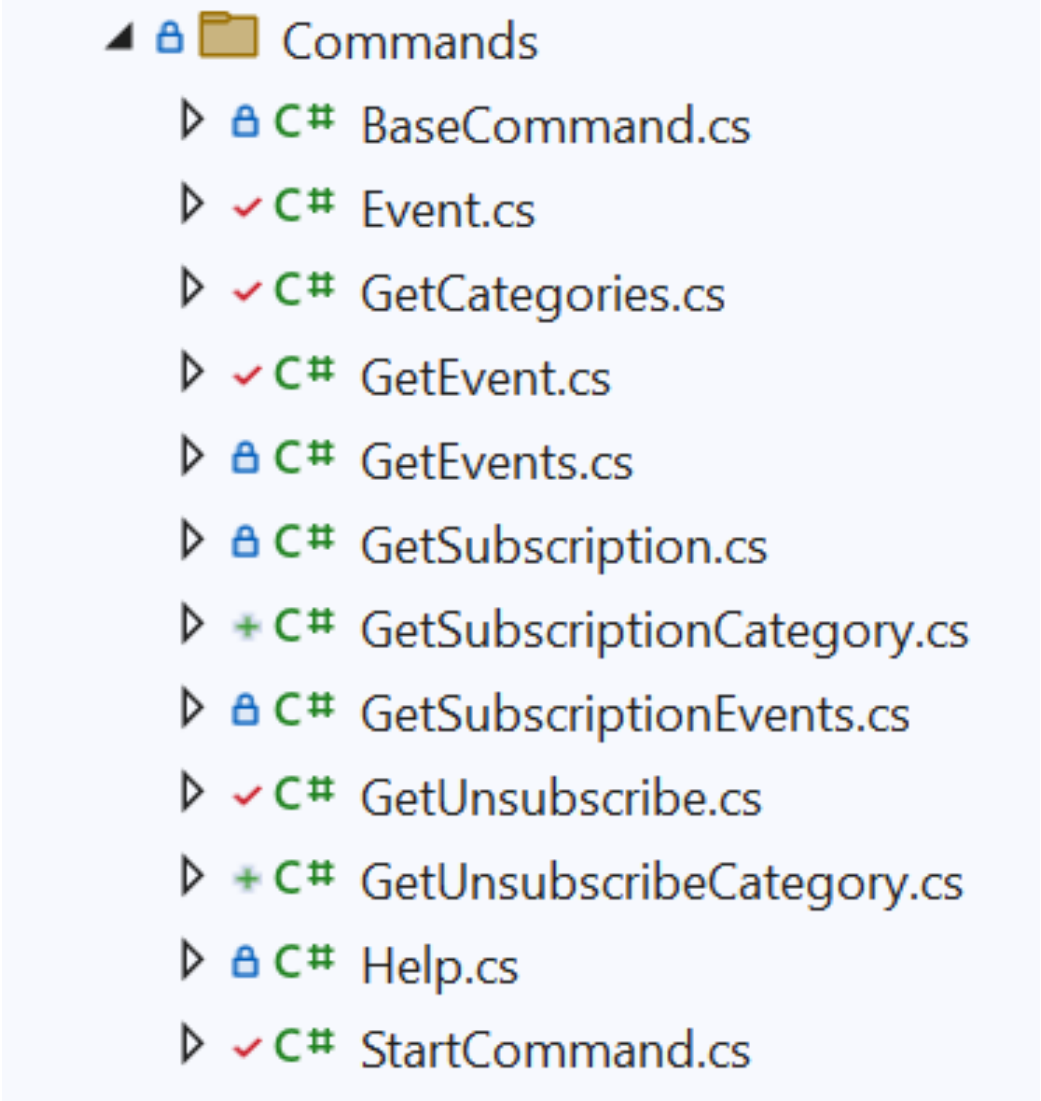


Рис. 6.11. Команди

Бот містить дванадцять команд, які виконують наступні дії:

- BaseCommand – базова команда для всіх команд.
- Event – команда для запиту від сервера, яку саме подію користувач бажає вивести.
- GetEvent – команда, яка повертає конкретну подію, якщо така існує в базі даних.
- GetEvents – команда, яка повертає список всіх подій.
- GetCategories – команда, яка повертає всі категорії, що є в базі даних, а також всі події, які відносяться до конкретної категорії.

- GetSubscription – команда для підписки користувача на подію при натисканні на кнопку «Підписатися».
- GetSubscriptionCategory – команда для підписки користувача на категорію при натисканні на кнопку «Підписатися». Підписка на категорію передбачає підписку користувача на всі події, що відносяться до даної категорії.
- GetSubscriptionEvents – команда, яка повертає список всіх подій на які підписаний користувач.
- GetUnsubsribe – команда для скасування підписки користувача на подію при натисканні на кнопку «Відписатися».
- GetUnsubscribeCategory – команда для скасування підписки користувача на категорію при натисканні на кнопку «Відписатися». Скасування підписки на категорію передбачає скасування підписки користувача на всі події, що відносяться до даної категорії.
- Help – команда для виведення списку всіх доступних команд.
- StartCommand – команда що виконується автоматично при запуску бота і реєструє користувача, якщо він не зареєстрований.

6.3.5. Сутності

Також необхідно створити відповідні сутності для того, щоб взаємодіяти з ботом:

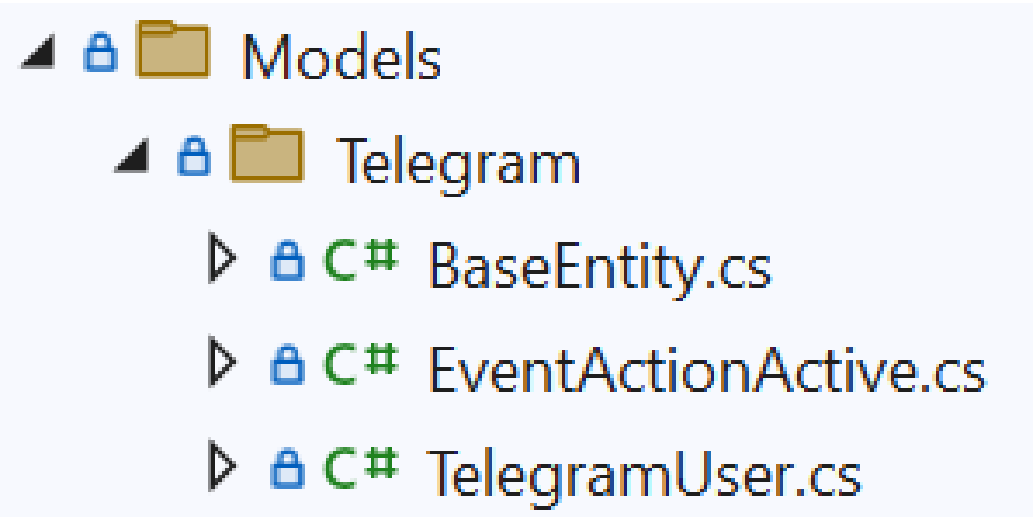


Рис. 6.12. Сутності

Для того, щоб спростити керування визначенням всіх команд за допомогою сервісу, варто додати спеціальний клас, що зберігатиме назви усіх цих команд.

Рис. 6.13. Клас, що зберігає назви всіх команд для телеграм-бота

6.3.6. Мережева утиліта ngrok

Отже, щоб безпечно перевірити працездатність телеграм-бота, можна скористатися такою мережевою утилітою як ngrok.

Дана утиліта дозволить користувачам ділитися і отримувати доступ до будь-яких локально розміщених веб-додатків через загальнодоступний веб-URL, що надається за допомогою домена Ngrok.io.

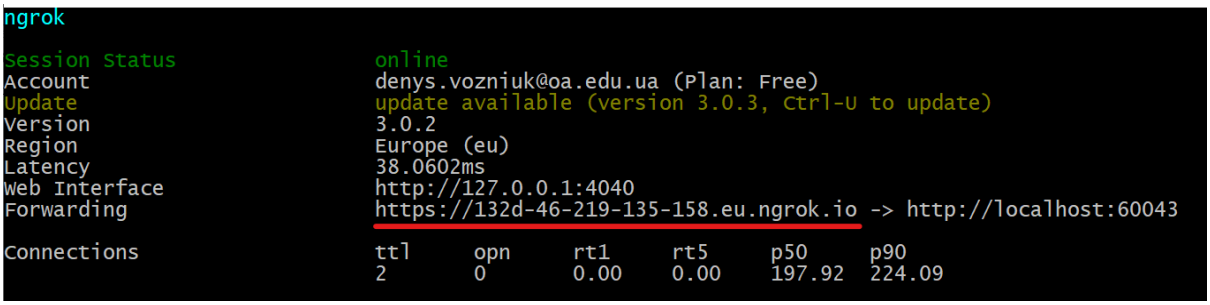


Рис. 6.14. Інтерфейс команди ngrok

Щоб отримувати всі запити від користувача через ngrok на віддалений сервер, необхідно підключити його до проекту, для цього у файлі appsettings.json потрібно добавити ще одну конфігурацію “Url”, якій призначити адресу виділену на рисунку «Рис 7.».

6.3.7. Взаємодія із базою даних

Для взаємодії із базою даних, потрібно додати відповідні таблиці: TelegramUsers та TelegramEvent. Кожна таблиця має відповідні поля і призначення.

TelegramUsers – таблиця для зберігання всіх користувачів та інформації про них. Початково вона є порожньою:

	Id	ChatId	Username	FirstName	LastName	CreatedAt
▶*	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 6.15. Таблиця TelegramUsers

TelegramEvent – таблиця для перевірки виконання умови, коли користувач хоче отримати інформацію про конкретну подію. Дана таблиця більш необхідна для серверної частини, аби спростити певну логіку реалізації.

6.4. Висновки

В результаті роботи було використано систему керування базою даних MSSQL, розроблений компанією Microsoft веб фреймворк ASP.NET Core, бібліотеку ReactJS та бібліотеку Telegram.Bot для написання серверної, клієнтської частин, а також механізму сповіщення користувача через соціальну мережу. В ході виконання було побудовано багатошарову архітектуру з декількома рівнями на серверній частині. Розроблені рівні презентації, доступу до даних тощо. Розроблено клієнтську частину застосунку для взаємодії із рівнем презентації серверної частини. Та використано набір засобів для розробки модулю системи сповіщення користувача.

РОЗДІЛ 7
АВТОРИЗАЦІЯ ТА АУТЕНТИФІКАЦІЯ

Хоча аутентифікація та авторизація часто використовуються як взаємозамінні, вони є окремими процесами, які використовуються для захисту організації від кібератак. Оскільки порушення даних продовжує зростати як за частотою, так і за обсягом, аутентифікація та авторизація є першою лінією захисту, яка запобігає потраплянню конфіденційних даних у чужі руки. У результаті надійні методи аутентифікації та авторизації мають бути важливою частиною загальної стратегії безпеки кожної системи чи організації.

Аутентифікація – це процес перевірки того, ким є окремий користувач, тоді як авторизація – це процес перевірки, до яких конкретних програм, файлів і даних користувач має доступ. Це схоже на ситуацію з авіакомпанією, яка повинна визначити, які люди можуть потрапити на борт літака. Перший крок – це підтвердити особу пасажира, щоб переконатися, що він є тим, за кого себе видає. Після визначення особи пасажира другим кроком є перевірка будь-яких спеціальних послуг, до яких пасажир має доступ, чи то політ першим класом, чи відвідування VIP-зони.

У цифровому світі аутентифікація та авторизація досягають цих же цілей. Аутентифікація використовується для підтвердження того, що користувачі дійсно є тими, ким вони себе представляють. Після підтвердження авторизація використовується для надання користувачеві дозволу на доступ до різних рівнів інформації та виконання конкретних функцій залежно від правил, встановлених для різних типів користувачів.

7.1. Авторизація

Авторизацію можна охарактеризувати наступними рисами:

- Визначає, до яких ресурсів може отримати доступ користувач.
- Працює через налаштування, які впроваджуються та підтримуються організацією.
- Завжди відбувається після аутентифікації.
- Користувач не бачить її і не може змінити.

Приклад: після того, як їхній рівень доступу буде перевірено, співробітники та менеджери з персоналу можуть отримати доступ до різних рівнів даних на основі дозволів, встановлених організацією.

Поширені методи авторизації

Після автентифікації користувача застосовуються засоби контролю авторизації, щоб забезпечити доступ користувачів до потрібних даних і виконання певних функцій, таких як додавання або видалення інформації, на основі дозволів, наданих організацією. Ці дозволи можна призначати на рівні програми, операційної системи або інфраструктури. Два поширені методи авторизації включають:

- *Контроль доступу на основі ролей (RBAC):* цей метод авторизації надає користувачам доступ до інформації на основі їхньої ролі в організації. Наприклад, усі співробітники компанії можуть переглядати, але не змінювати свою особисту інформацію, таку як оплата, час відпустки та дані рахунків. Тим не менш, менеджери з персоналу можуть отримати доступ до інформації про всіх співробітників із можливістю додавати, видаляти та змінювати ці дані. Призначаючи дозволи відповідно до ролі кожної особи, організації можуть забезпечити продуктивність кожного користувача, обмежуючи при цьому доступ до конфіденційної інформації.

- *Контроль доступу на основі атрибутів (ABAC):* ABAC надає користувачам дозволи на більш детальному рівні, ніж RBAC, використовуючи ряд конкретних атрибутів. Вони можуть включати такі атрибути користувача, як ім’я користувача, роль, організація, ідентифікатор тощо. Він може включати атрибути середовища, такі як час доступу, розташування даних та поточні рівні організаційної загрози. Він може включати такі атрибути ресурсу, як власник ресурсу, ім’я файлу та рівень чутливості даних. ABAC є більш складним процесом авторизації, ніж RBAC, призначений для додаткового обмеження доступу. Наприклад, замість того, щоб дозволити всім менеджерам з персоналу в організації змінювати кадрові дані співробітників, доступ можна обмежити певними географічними місцями або годинами дня, щоб підтримувати жорсткі обмеження безпеки.

7.2. Аутентифікація

Аутентифікацію можна охарактеризувати наступними рисами:

- Підтверджує, ким є користувач.
- Здійснюється за допомогою паролів, одноразових пін-кодів, біометричної інформації та іншої інформації, наданої або введеної користувачем.
- Є першим кроком хорошого процесу керування ідентифікацією та доступом.
- Користувач бачить і частково змінює.
- Приклад: підтвердивши свою особу, співробітники можуть отримати доступ до програми управління персоналом, яка включає їх особисту інформацію про зарплату, час відпустки та інші дані.

Поширені методи аутентифікації

Хоча ідентичність користувача історично перевірялася за допомогою комбінації імені користувача та пароля, сучасні методи аутентифікації зазвичай покладаються на три класи інформації:

- *Що ви знаєте:* найчастіше це пароль. Але це також може бути відповіддю на таємне запитання або одноразовий PIN-код, який надає користувачеві доступ лише до одного сеансу або транзакції.
- *Що ви маєте:* це може бути мобільний пристрій або додаток, маркер (token) безпеки або цифрова ідентифікаційна картка.
- *Що ви:* це біометричні дані, такі як відбиток пальця, сканування сітківки або розпізнавання обличчя.

Часто ці типи інформації об’єднуються за допомогою кількох рівнів аутентифікації. Наприклад, користувача можуть попросити ввести ім’я користувача та пароль для завершення покупки в Інтернеті. Як тільки це буде підтверджено, одноразовий PIN-код може бути надісланий на мобільний телефон користувача як другий рівень безпеки. Поєднуючи кілька методів аутентифікації з послідовними протоколами аутентифікації, організації можуть забезпечити безпеку, а також сумісність між системами [16].

7.3. Реалізація авторизації та аутентифікації

Серверна частина:

Для автентифікації користувача було обрано JWT та автентифікацію за допомогою системи Google.

JWT, або JSON Web Token, – це відкритий стандарт, який використовується для обміну інформацією безпеки між двома сторонами – клієнтом і сервером. Кожен JWT містить закодовані об’єкти JSON, включаючи набір вимог (Claims).

JWT підписуються за допомогою криптографічного алгоритму, щоб гарантувати, що вимоги не можуть бути змінені після видачі токена.

JSON означає JavaScript Object Notation і є текстовим форматом для передачі даних через веб-додатки. Він зберігає інформацію в легкодоступному вигляді, як для розробників, так і для комп'ютерів. Він може використовуватися як формат даних будь-якою мовою програмування і швидко стає кращим синтаксисом для API, перевершуючи XML.

Що таке токени?

Токени – це рядок даних, який представляє щось інше, наприклад, ідентичність. У разі аутентифікації маркер, не заснований на JWT, являє собою рядок символів, які дозволяють одержувачу підтвердити ідентичність відправника. Важливою відмінністю тут є відсутність сенсу в самих символах.

JWT відрізняються від інших веб-токенів тим, що містять набір тверджень (Claims). Вони використовуються для передачі інформації між двома сторонами. Що це за твердження, залежить від конкретного випадку використання. Наприклад, твердження може містити інформацію про те, хто видав токен, як довго він дійсний або які дозволи надано користувачу.

JWT – це рядок, що складається з трьох частин, розділених крапками, і виглядає приблизно так: «eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9**.**eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ**.**SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c»

Після декодування отримуються два рядки у форматі JSON:

- Заголовок і інформація.
- Підпис.

Заголовок JOSE (JSON Object Signing and Encryption) містить тип токена – в даному випадку JWT і алгоритм підпису.

Інформація містить твердження. Вона відображається як рядок JSON, який зазвичай містить не більше десятка полів, щоб JWT був компактним. Ця інформація зазвичай використовується сервером для перевірки того, що користувач має дозвіл виконувати дію.

Підпис гарантує, що токен не був змінений. Сторона, яка створює JWT, підписує заголовок і інформацію секретом, який відомий як емітенту, так і одержувачу, або закритим ключем, відомим лише відправнику. Коли токен використовується, сторона, що отримує, перевіряє, що заголовок і інформація відповідають підпису.

Додаток В

Клієнтська частина:

Процес автентифікації був реалізований за допомогою хуку useContext. Даний хук використовується для можливості глобального збереження даних та передачі їх між різними компонентами.

Для цього ми створюємо компонент AuthContext, який відповідає за створення контексту, та AuthProvider, який зберігає інформацію про користувача, який увійшов у систему. Даний компонент містить також методи для входу користувача, виходу користувача та реєстрації нового користувача.

Після входу у систему, на клієнтську частину нам повертається пара токенів – access token та refresh token. Ці дані ми зберігаємо в стан у компоненті AuthProvider та розміщуємо у local storage. Таким чином, доступ до токенів у нас є завжди. Щоб отримати їх, нам необхідно ініціалізувати змінну tokens, імпортувати useContext та передати наш створений контекст AuthContext: *const {tokens} = useContext(AuthContext)*. Тепер ми маємо змогу перевірити чи є наявні

токени, якщо так – користувач має доступ до компонентів панелі адміністратора, ні – користувача повертає на сторінку для входу у систему.

Щоб вийти із системи, ми видаляємо токени з local storage та стану, який їх зберігав.

Для оновлення пари токенів існує метод updateToken, якому ми надаємо певний інтервал часу за яким він буде викликатись та відправляємо запит на серверну часину.

7.4. Висновки

Отже механізм автентифікації в системі передбачає перевірку того, ким є певний користувач. Тобто провівши аутентифікацію система визначає, чи є окремий користувач дійсно тим, за кого він себе видає. Механізм авторизації передбачає визначення рівня доступу до системи конкретного користувача. Тобто провівши авторизацію система визначає, чи має право користувач на певний функціонал системи.

РОЗДІЛ 8
РЕЗУЛЬТАТ РОБОТИ НАД ІНТЕРФЕЙСОМ

8.1. Дизайн рішення

Перед розробкою коду були також реалізовані макети сторінок сайту. Їхня розробка відбувалась у графічному редакторі Adobe XD. Ця робота була зроблена для визначення вигляду додатку та візуалізації компонентів, що відповідають за реалізацію поставлених умов у технічному завданні. Першим етапом був пошук різноманітних стилів та вибір палітри, оскільки головними користувачами системи є молодь (студенти), для яких вигляд інтерфейсу визначає чи будуть вони користуватись даною системою чи ні. Потім було здійснено реалізацію UI Kit – набір базових UI елементів (форми, кнопки, заголовки, набір піктограм). Останнім етапом був розподіл компонент, що відповідають за функціонал системи, на сторінки.

8.2. Головна сторінка

Головна сторінка – це перша сторінка, яку бачить користувач перейшовши до застосунку.

На головній сторінці користувач має змогу:

- переглянути події, що відбудуться скоро
- переглянути події за факультетом
- переглянути всі події
- переглянути події, сортуючи їх за датою "Сьогодні", "Завтра", "Цього тижня", "Цих вихідних"
- підписатись на події, перейшовши до телеграм бота

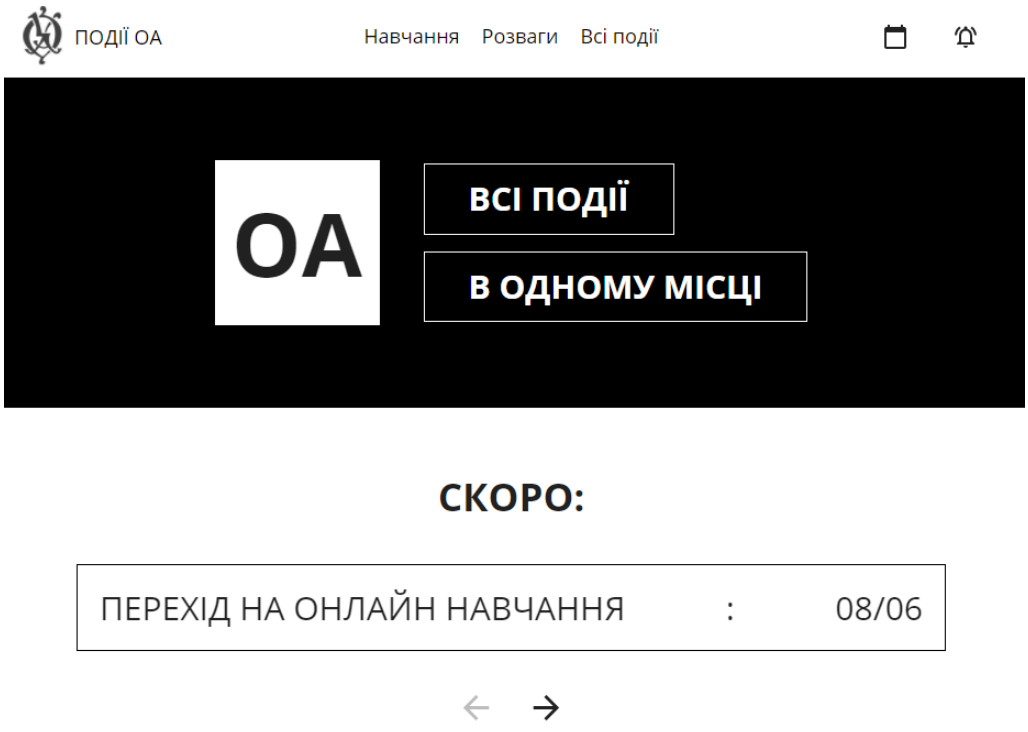


Рис. 8.1. Інтерфейс головної сторінки

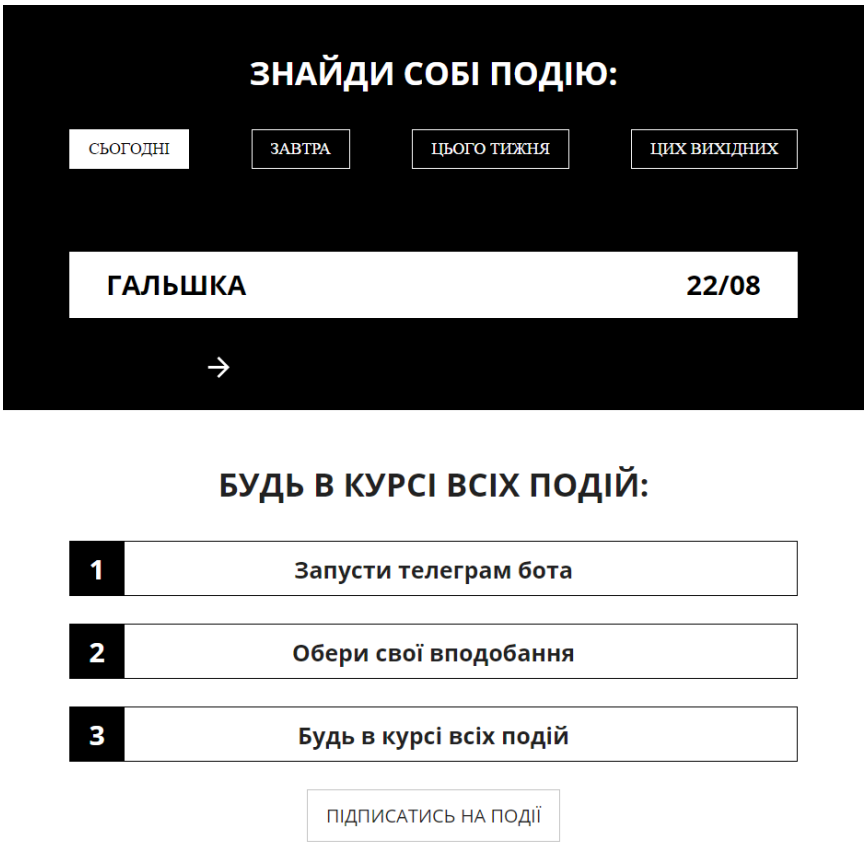


Рис. 8.2. Інтерфейс головної сторінки

8.3. Інтерфейс відображення подій

Для надання користувачу інформації, щодо наявних подій у системі, було реалізовано дві сторінки. Сторінка «Всі події» та сторінка «Деталі події».

На сторінці «Всі події» користувач має змогу:

- переглянути список всіх подій
- сортувати події за датою та категорією
- шукати події

Для зручності відображень подій було також додано пагінацію.

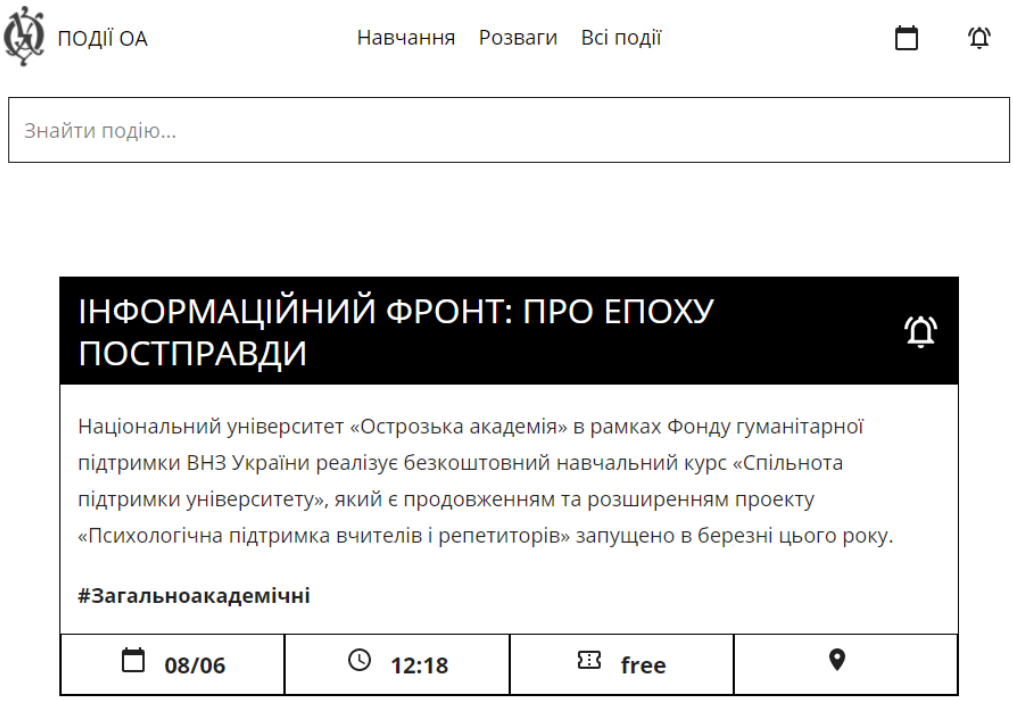


Рис. 8.3. Інтерфейс сторінки всіх подій

На сторінці «Деталі події» користувач має змогу:

- підписатись на подію, перейшовши до телеграм боту
- переглянути головну та розширену інформацію про конкретну подію

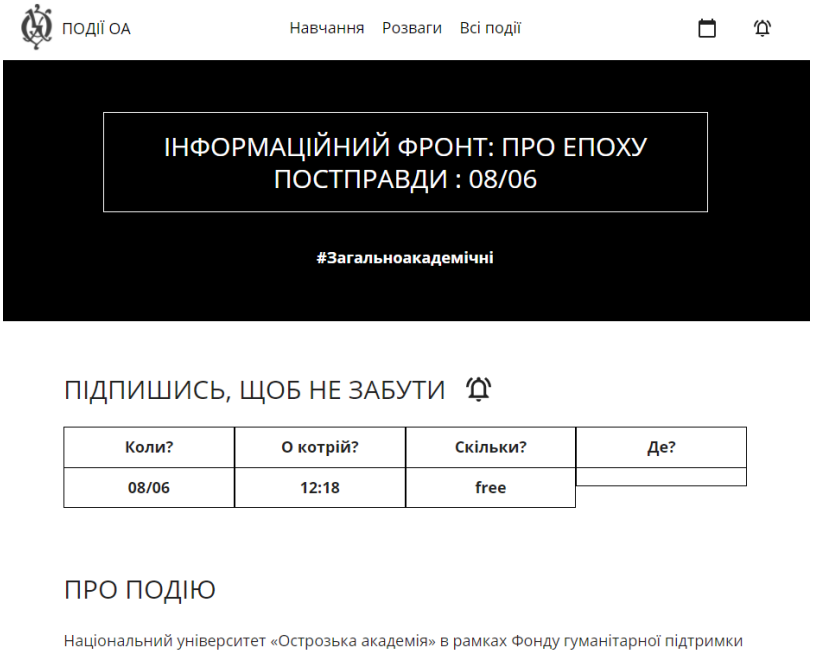


Рис. 8.4. Інтерфейс сторінки «Деталі події»

8.4. Інтерфейс адміністратора

Інтерфейс адміністратора представлений чотирма сторінками:

Табл. 8.1. Інтерфейс адміністратора

Назва сторінки	**Функціонал сторінки**
Панель адміністратора	Адміністратор має змогу: - переглянути список всіх подій - шукати події - видаляти події
Сторінка «Створити подію»	Адміністратор має змогу створити нову подію у системі
Сторінка «Редагувати подію»	Адміністратор має змогу редагувати наявну подію у системі
Панель категорій	Адміністратор має змогу додавати нові категорії або видаляти наявні

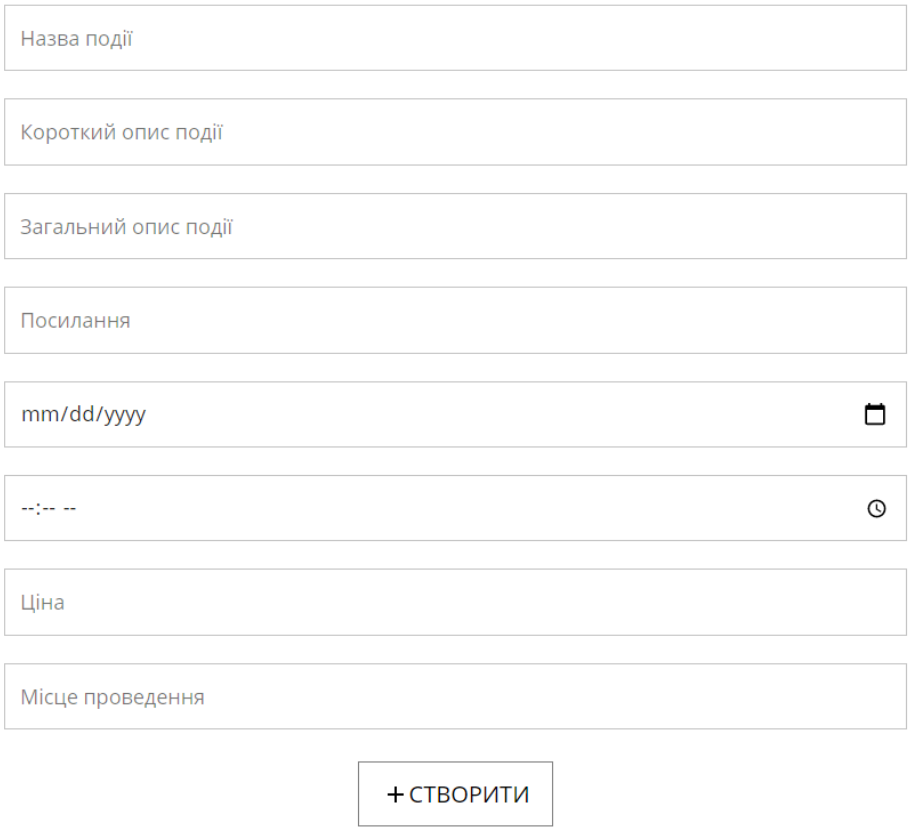


Рис. 8.5. Інтерфейс форми для створення події

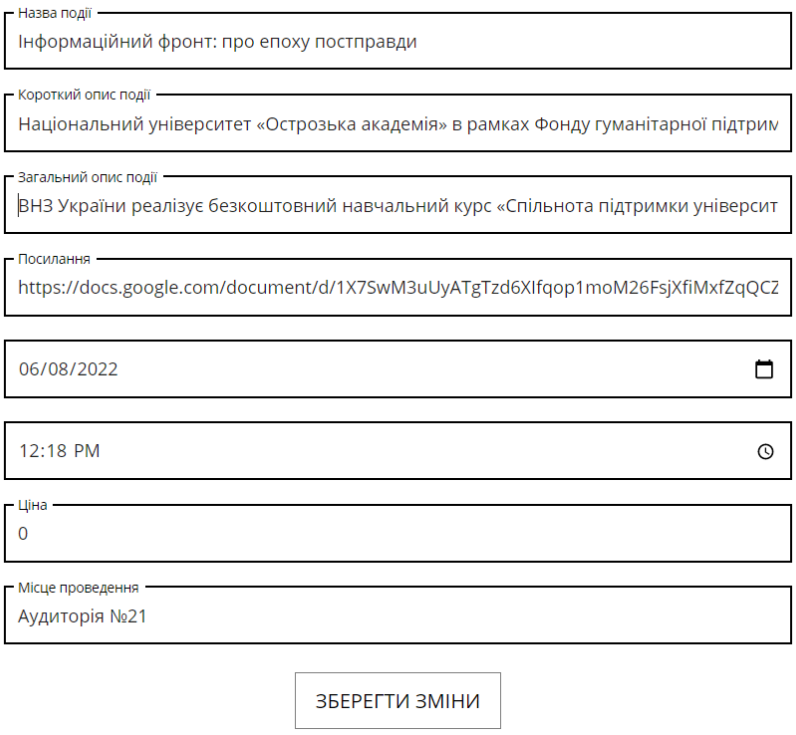


Рис. 8.6. Інтерфейс форми для редагування події

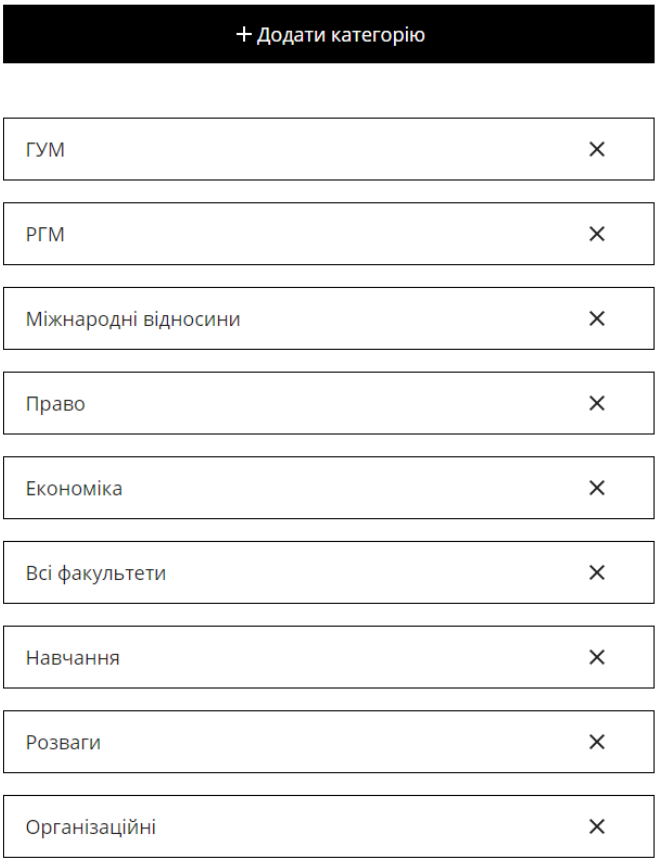


Рис. 8.7. Інтерфейс панелі для керування категоріями подій

8.5. Авторизація

Вхід у систему представлений формою для вводу електронної скриньки та паролю:

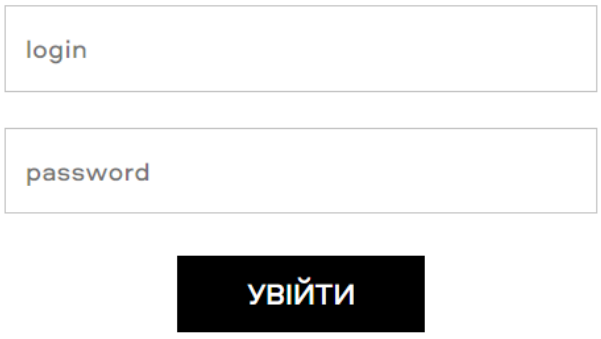


Рис. 8.8. Інтерфейс форми для входу у систему в ролі адміністратора

8.6. Telegram Bot

8.6.1. Запуск телеграм-бота. Команда «/start»

Отже, запустивши бота і тим самим, виконавши команду «/start», користувач отримує наступне повідомлення на екрані:

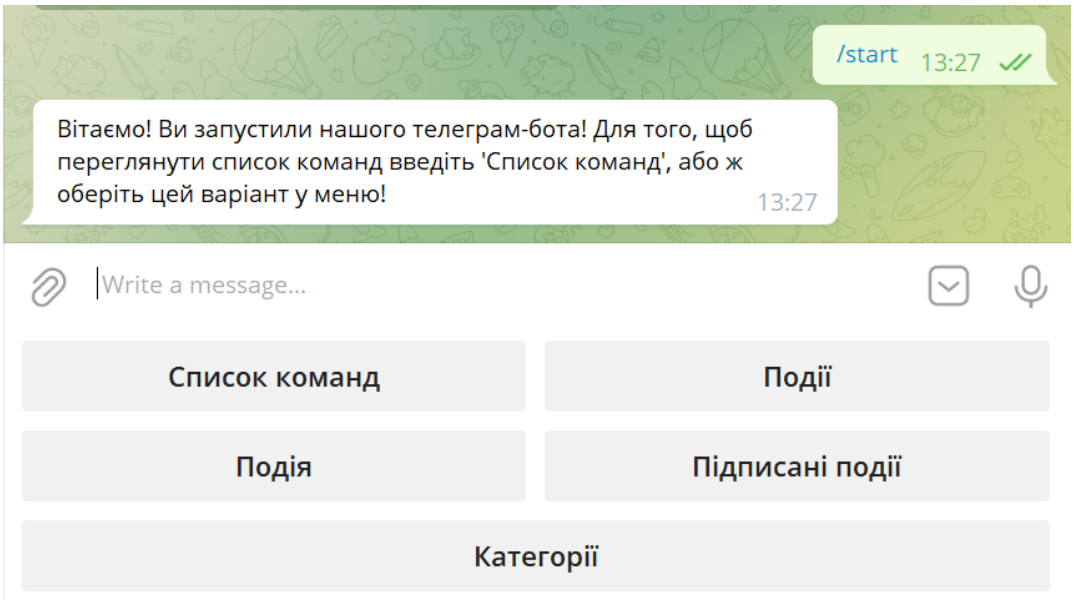


Рис. 8.9. Команда «/start»

І тим самим реєструється новий користувач у таблиці TelegramUsers нашої бази даних.

	Id	ChatId	Username	FirstName	LastName	CreatedAt
▶	3		DenVozniuk007	Den	Voznyuk	
*	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 8.10. Таблиця TelegramUsers

8.6.2. Команда «Список команд»

Для перевірки працездатності решти команд можна скористатися командою «Список команд», яка в свою чергу надсилає відповідь у вигляді повідомлення із списком всіх доступних користувачеві команд:

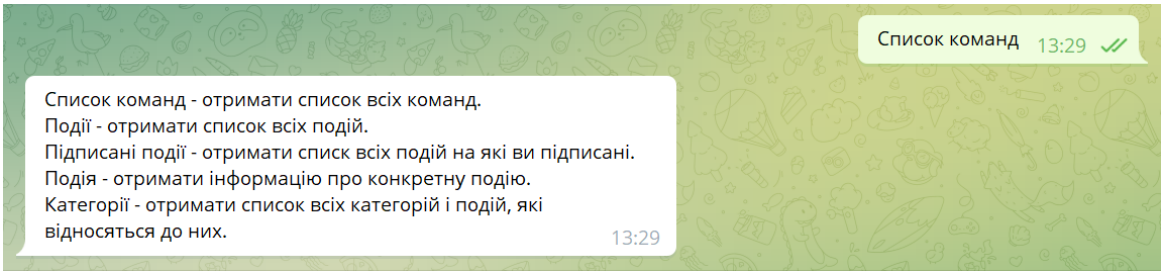


Рис. 8.11. Команда «Список команд»

8.6.3. Перевірка введення команд

Якщо ж ввести будь-який текст, що не відповідає назві будь-якої із команд перерахованих вище, то в такому разі бот не розпізнає цей запит і відправить для користувача наступне попередження:

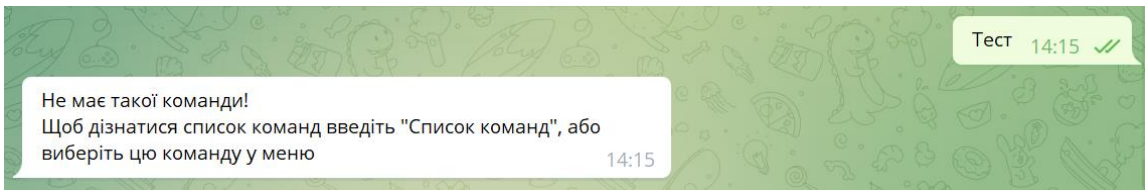


Рис. 8.12. Відповідь бота на неіснуючу команду

8.6.4. Команда «Події»

Команда «Події» виводить для користувача на екран список всіх подій:

Рис. 8.13. Команда «Події»

Оскільки подій може бути багато і щоб не виводити їх всі відразу, то для цього зроблений механізм пагінації. Його суть полягає у тому, що при натисканні на кнопку «Завантажити ще» бот відправить наступний список подій з бази даних. Кількість подій, що виводяться є фіксованою в системі.

Рис. 8.14. Кнопка «Завантажити ще»

Після натискання кнопки «Завантажити ще» бот надсилає користувачу наступний список подій.

Коли бот надсилає останній список подій із загального списку, то в такому разі кнопка «Завантажити ще» зникає, оскільки більше не має подій в загальному списку і залишається лише кнопка «Детальніше»:

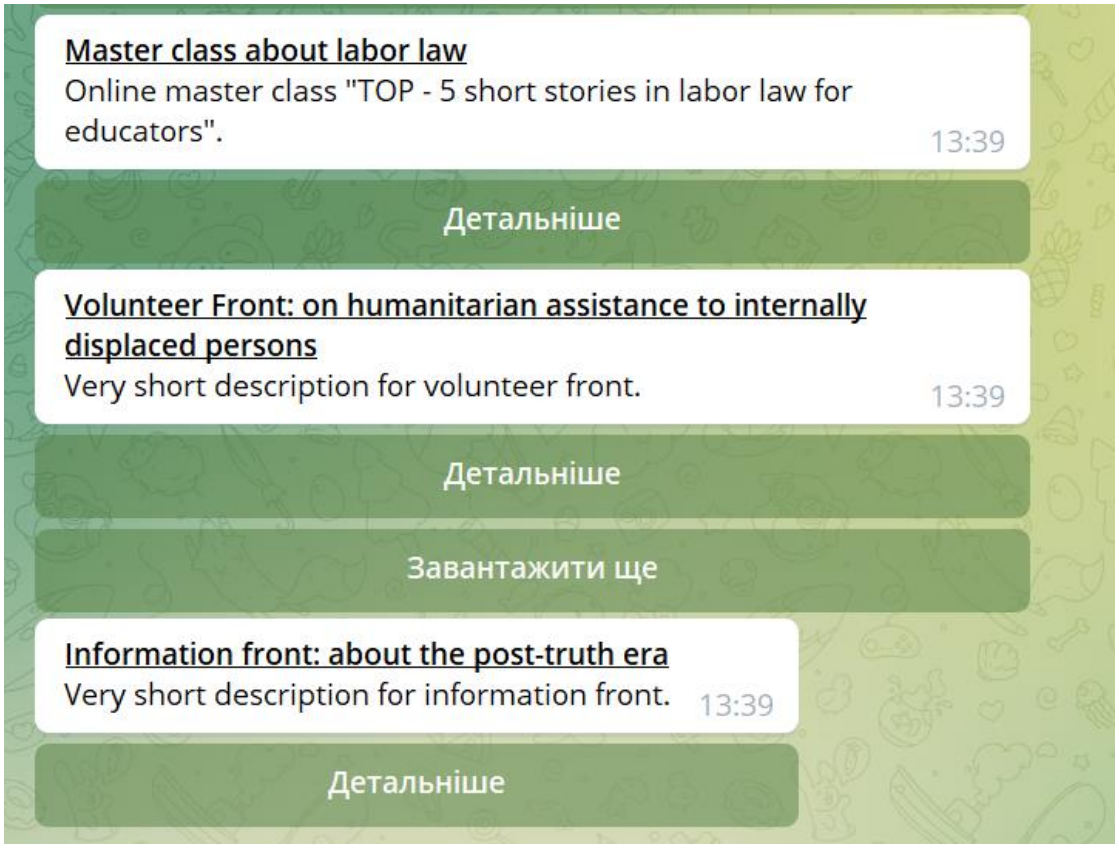


Рис. 8.15. Кінець списку подій

8.6.5. Команда «Підписані події»

Команда «Підписані події» виводить для користувача на екран список всіх подій на які підписаний користувач. У виведенні даного списку також передбачена пагінація, як і при виведенні списку всіх подій:

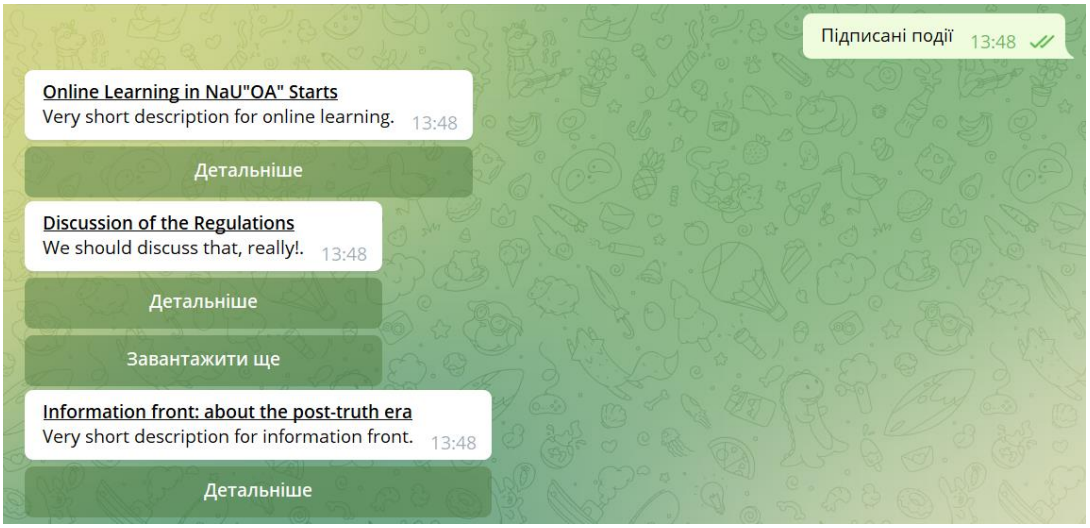


Рис. 8.16. Команда «Підписані події»

8.6.6. Команда «Подія»

Команда «Подія» виводить для користувача на екран повідомлення про вибір будь-якої події серед наявних для подальшого перегляду:

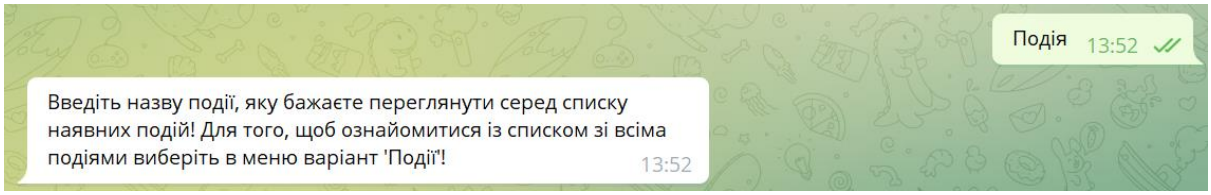


Рис. 8.17. Команда «Подія»

Після того як користувач ввів команду «Подія», телеграм-бот повідомляє його про те, що він може обрати будь-яку подію серед наявних. Для цього потрібно, щоб користувач ввів у поле текст та відправив для телеграм-бота.
Якщо користувач введе будь-який текст і жодна із подій не міститиме цього тексту у своєму описі, то телеграм-бот відправить у відповідь наступне повідомлення:

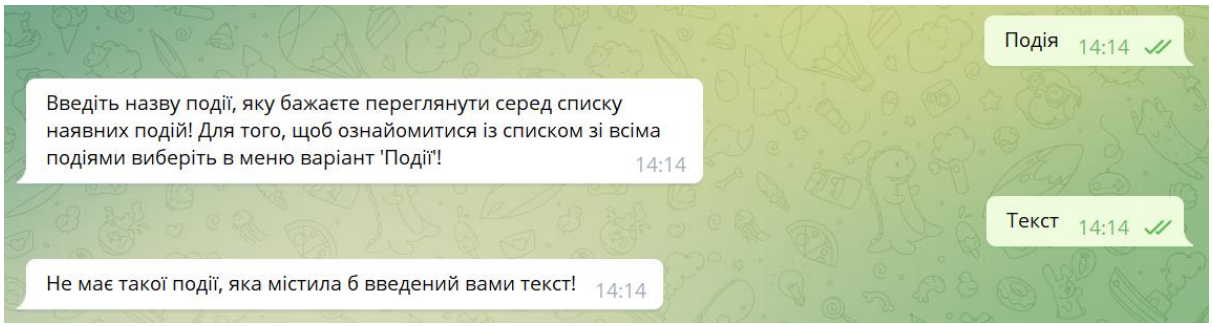


Рис. 8.18. Відповідь бота на текст, який не містить жодна подія

Якщо ж хоч одна із подій у своєму описі міститиме текст, який ввів користувач, то в такому разі бот відправить список подій, які міститимуть в собі цей текст з кнопкою «Підписатися» під кожною подією із цього списку, на які користувач ще не підписаний. Якщо користувач вже підписаний на будь-яку подію із цього списку, то в такому разі телеграм-бот відправить користувачеві кожну таку подію на екран із кнопкою «Відписатися».

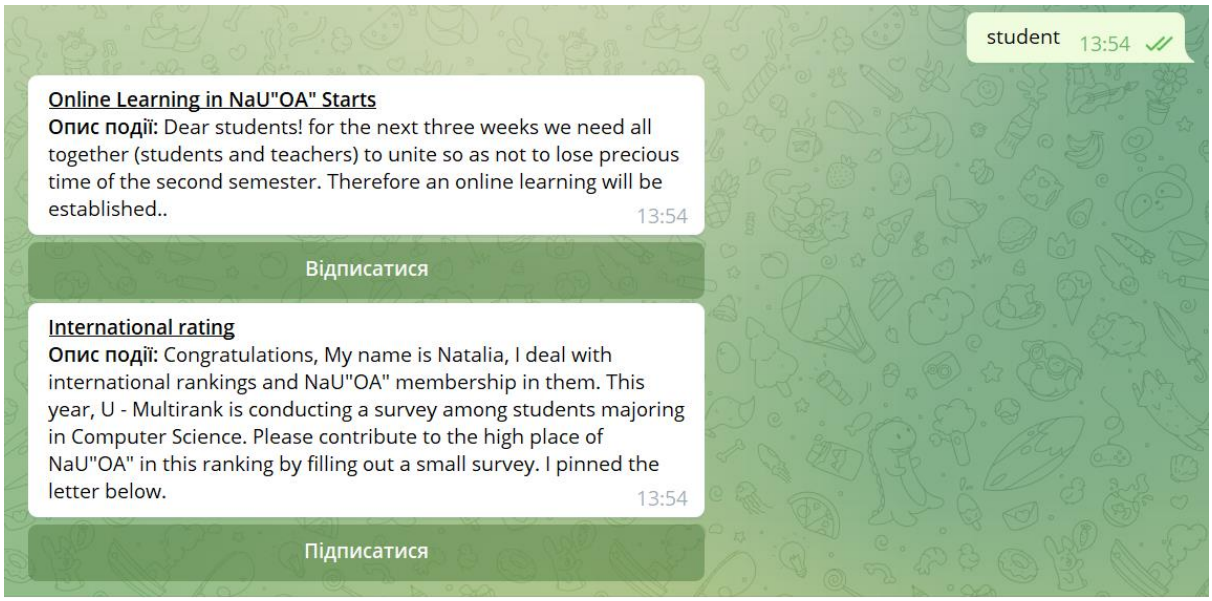


Рис. 8.19. Результат роботи команди «Подія»

8.6.7. Підписання користувача на подію та відписання

На подію «International rating» користувач ще не був підписаний, отже, при натисканні на кнопку «Підписатися» він отримає наступне повідомлення:

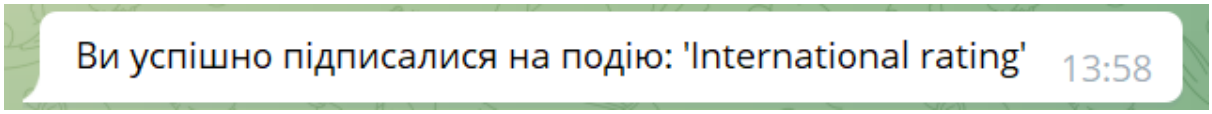


Рис. 8.20. Відповідь на кнопку «Підписатися»

Після отримання даного повідомлення користувач підписується на цю подію і вона автоматично добавляється в список подій на які підписаний користувач:

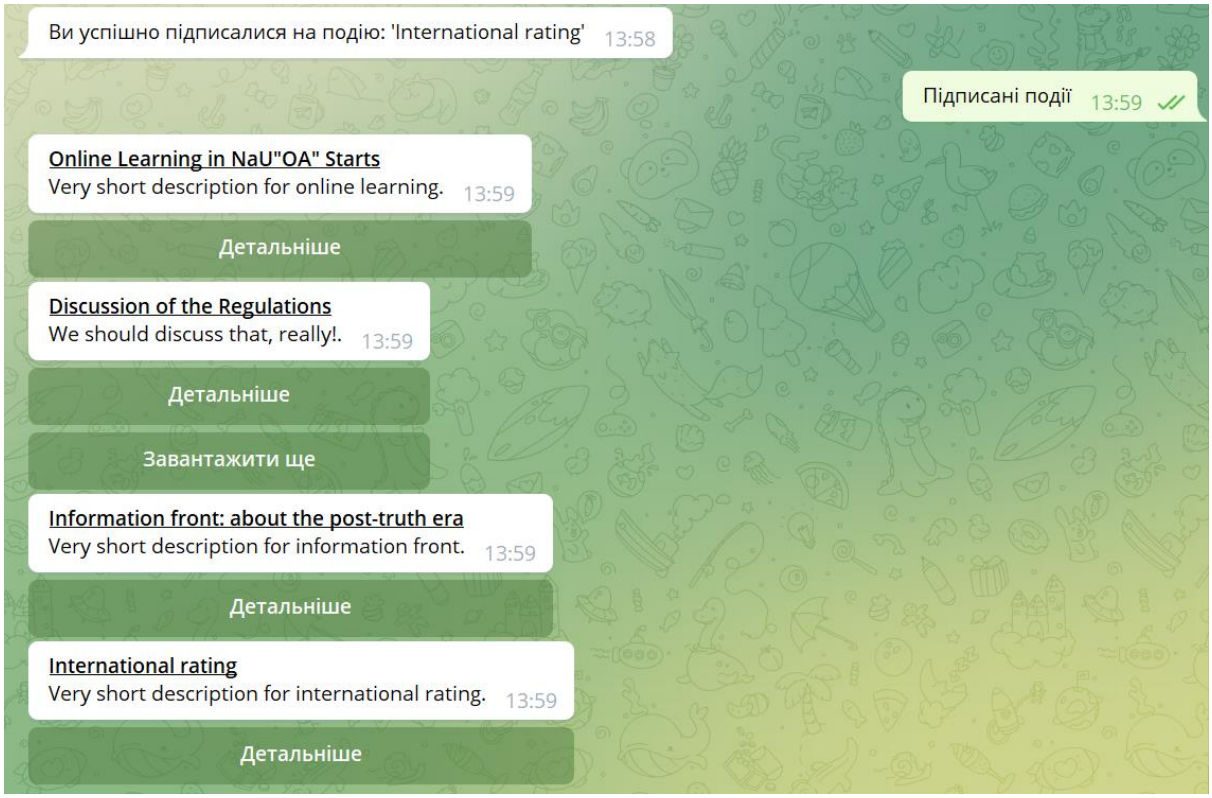


Рис. 8.21. Команда «Підписані події»

Також, щоб вивести конкретну подію, користувач може натиснути на кнопку «Детальніше» при виведенні списку всіх подій, або ж списку підписаних подій, і телеграм-бот відправить цю подію на екран під якою знову ж таки буде кнопка або «Підписатися», якщо користувач ще не підписаний на цю подію, або «Відписатися» - якщо підписаний.

Щоб перевірити процес реалізації скасування підписки необхідно вивести подію, яка є в списку подій на які підписаний користувач, та натиснути кнопку «Відписатися»:

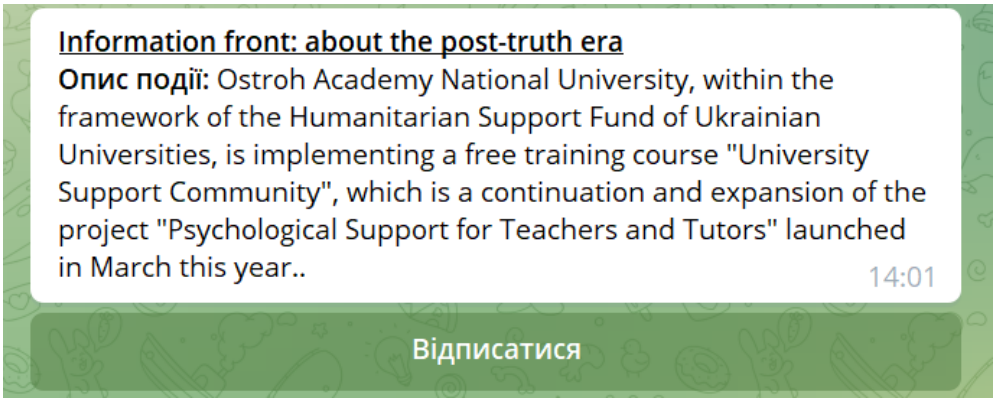


Рис. 8.22. Результат роботи кнопки «Детальніше»

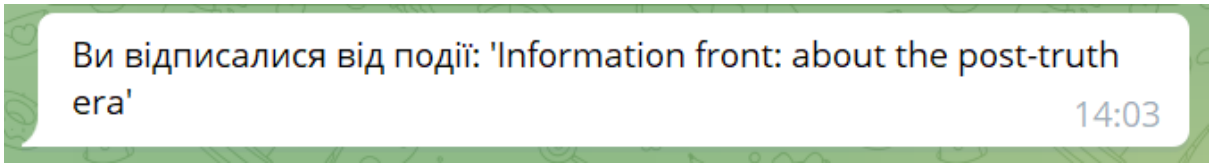


Рис. 8.23. Відповідь на кнопку «Відписатися»

Після отримання даного повідомлення користувач відписується від цієї події і вона автоматично видаляється із списку подій на які підписаний користувач:

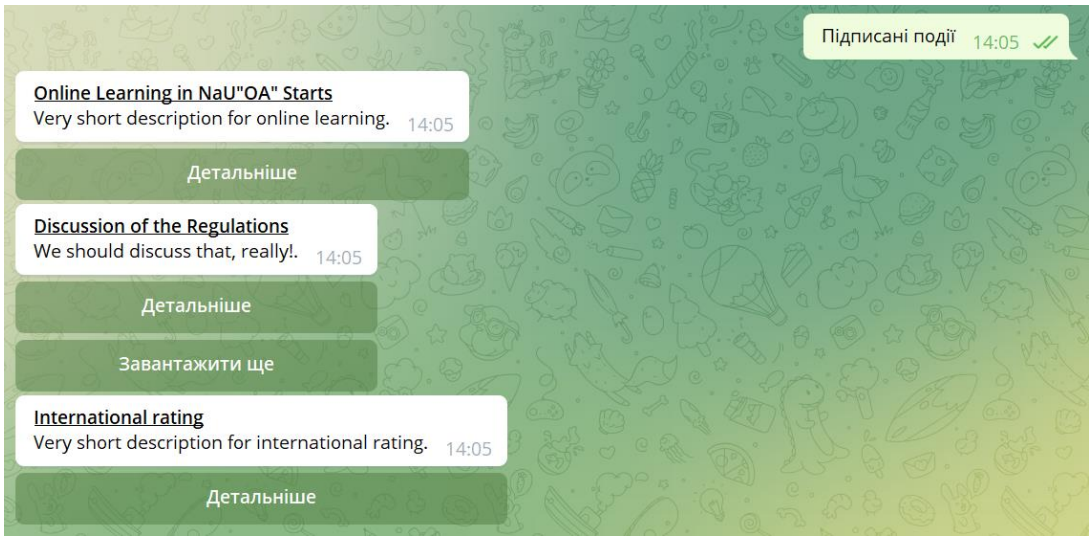


Рис. 8.24. Команда «Підписані події»

Тепер, якщо вивести цю саму подію на екран, то кнопка зміниться на «Підписатися» і, натиснувши її, знову буде реалізований процес підписки, так як даної події в списку подій на які підписаний користувач вже не має:

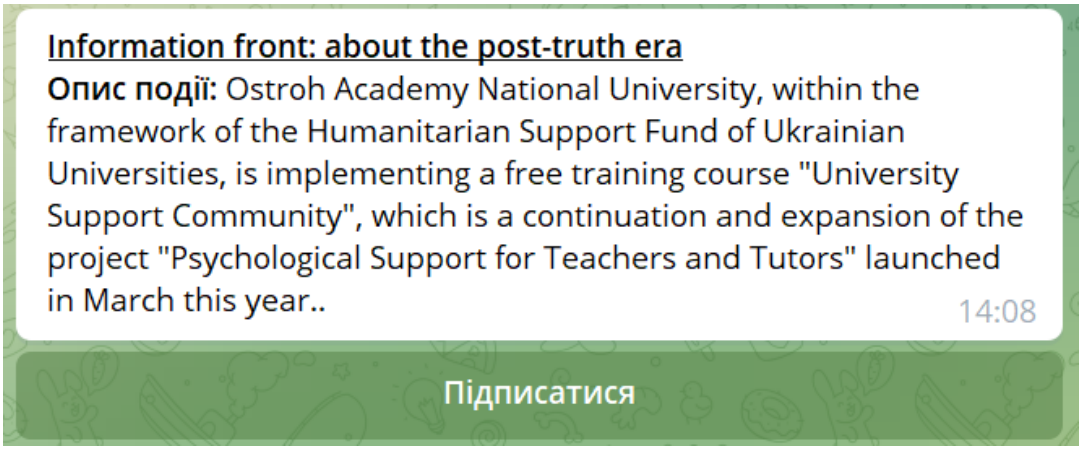


Рис. 8.25. Виведення події.Команда «Категорії»

Для того, щоб вивести список всіх доступних категорій, потрібно ввести, або обрати в меню кнопку «Категорії»:

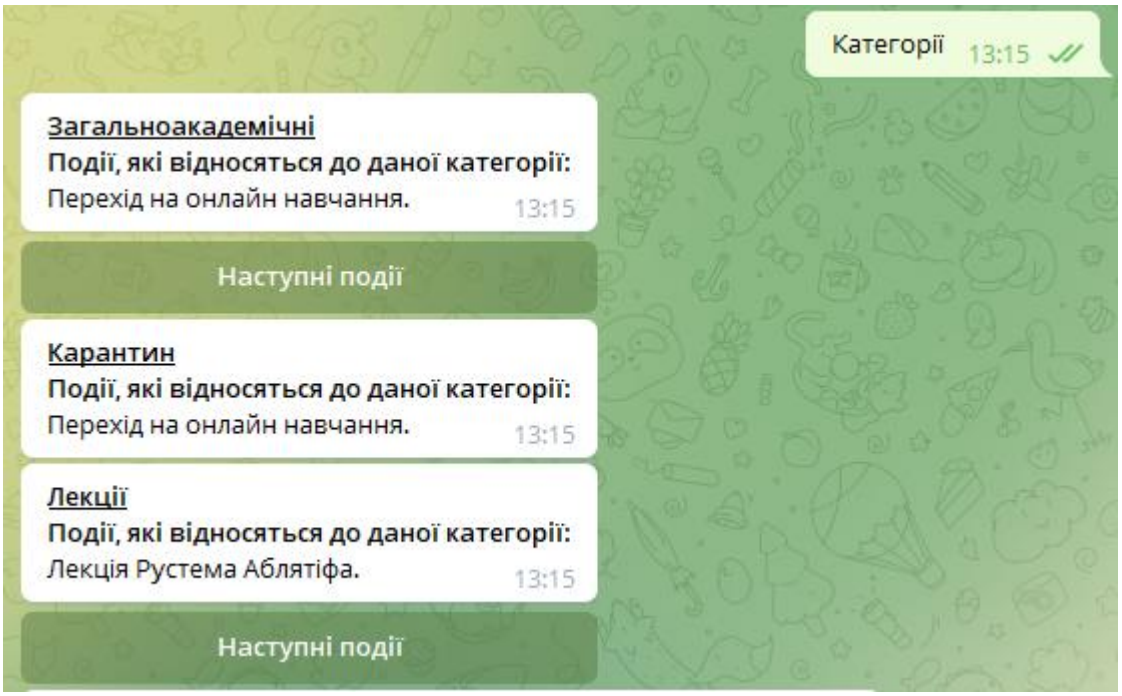


Рис. 8.26. Команда «Категорії»

Після цього бот надішле для користувача список всіх категорій та подіями, які відносяться до кожної із категорій. Оскільки подій може бути багато, то в даному випадку для їх виведення реалізований механізм пагінації за яким після натискання на кнопку «Наступні події» для користувача під даною категорією підвантажиться наступний список подій:

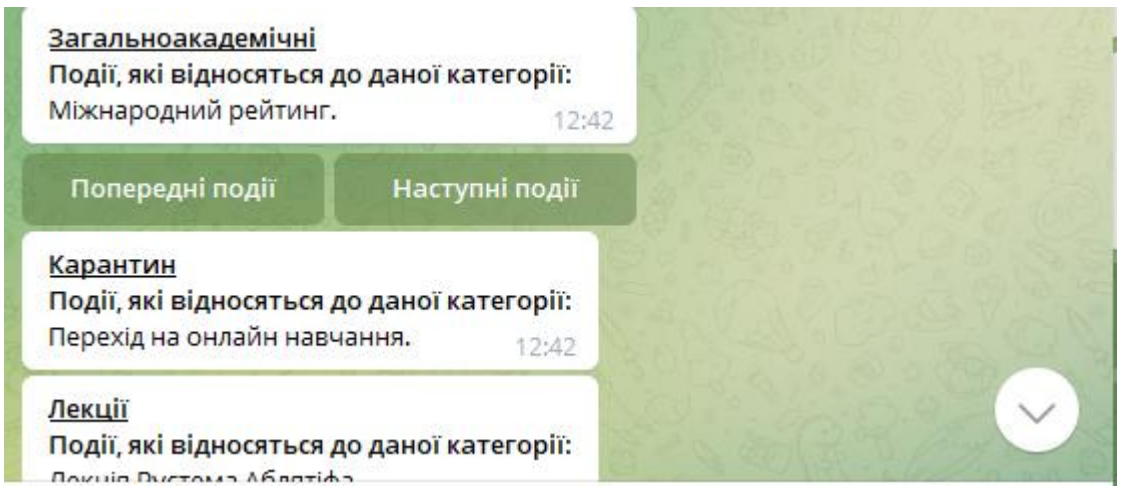


Рис. 8.27. Результат роботи кнопки «Наступні події»

Після цього, як підвантажився новий список подій, з’являється ще одна кнопка «Попередні події» за допомогою якої можна повернутися до попереднього списку подій:

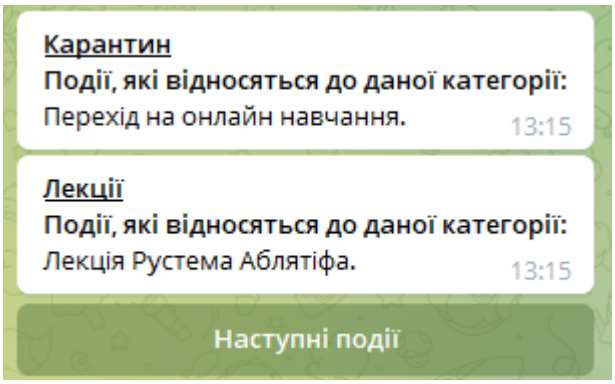


Рис. 8.28. Результат роботи кнопки «Попередні події»

Якщо ж при натисканні на кнопку «Наступні події» буде підвантажений останній список подій, що відносяться до категорії, то в такому випадку дана кнопка для завантаження наступних подій зникне і залишиться лише варіант завантаження попередніх подій:

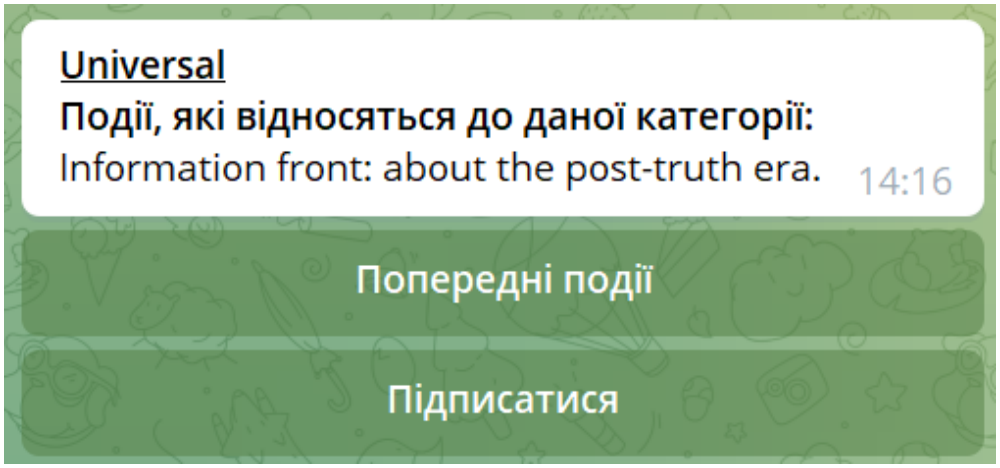


Рис. 8.29. Кінець списку подій даної категорії

Також використання телеграм-бота передбачає підписання на категорію. Після натискання на кнопку «Підписатися» під будь-якою категорією користувач підписується на всі події, які відносяться до даної категорії.

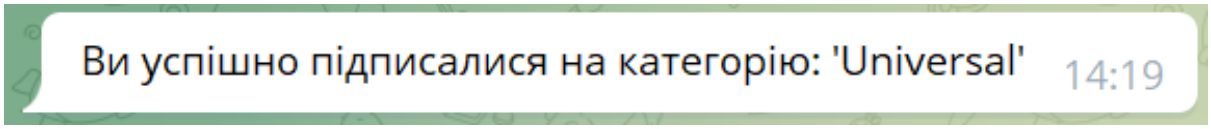


Рис. 8.30. Підписка на категорію

Після цього всі події з цієї категорії, на які ще не був підписаний користувач, добавляються в список підписаних подій. Таким чином, при повторному виведенні команди «Категорії» під даною категорією, на яку користувач підписувався, кнопка «Підписатися» зміниться на кнопку «Відписатися», так як користувач підписаний на всі події цієї категорії.

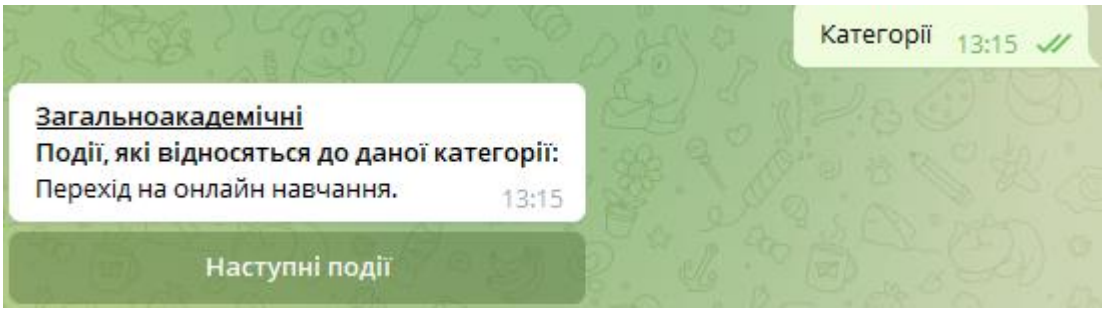


Рис. 8.31. Команда «Категорії»

8.7. Висновки

Інтерфейс веб застосунку та телеграм боту відіграє значну роль для користувача системи. Він визначає наскільки ефективно система взаємодіє з користувачем. Основними цілями роботи над інтрефейсом були логічність та простота, що надає користувачу чітке поняття як користуватися системою.

ВИСНОВОК

Результатом даної роботи є програмне забезпечення для керування системою анонсів та подій.

Використовуючи даний застосунок, користувач може відслідковувати події. Фільтрувати ці події, підписуватись на них та отримувати сповіщення про наближення дати проведення події. Також він має можливість переглядати категорії подій, а також підписуватись на категорію загалом.

Адміністратор цього програмного застосунку має змогу створювати категорії та події. Він може редагувати ці категорії та події, змінювати їх опис, дату проведення тощо. Мати змогу видаляти їх, здійснювати пошук та фільтрацію. Для вищевказаних дій, адміністратору системи потрібно авторизуватись в системі. Також він має функціонал для створення адміністраторів системи.

Для створення серверної частини цього рішення було використано ASP.NET Core платформу від Microsoft, систему керування базами даних MS SQL. Для застосунку також було використано Serilog – для логування подій системи, Swagger – для написання прикладного програмного інтерфейсу, Automapper – для організації потоку даних в системі на рівні шару доступу до даних, Entity Framework Core для організації шару доступу до даних.

Основою платформою для розробки клієнтської частини була бібліотека на основі мови програмування JavaScript – React JS. Основним критерієм вибору цієї бібліотеки слугувала швидкість та простота розробки і легка масштабованість проєкту. Основною ідеєю React JS є компоненти, які репрезентують блоки нашого проєкту. З метою керування та відстеження станів компонентів були використані хуки – спеціальні функції.

Для розробки інтерфейсу було використано два інструменти – графічний редактор Adobe XD, який дозволив створити попередні макети веб застосунку та MUI – бібліотека стилів, що надає користувачам елементи з попередніми стилями, які можна змінити. Основними цілями при розробці інтерфейсу були

простота та зрозумілість.

Організація системи сповіщень була здійснена за допомогою використання бота для месенджера Telegram та бібліотеки для планування завдань у фоновому режимі Hangfire. Для розробки бота було використано бібліотеку Telegram.Bot для платформи .Net. Отримавши посилання на бота та активувавши його, користувач має змогу переглядати події, шукати їх та підписуватись на них. Також він має змогу переглядати категорії та події, що входять до цієї категорії

Користувачами даної системи можуть бути студенти та викладачі, а також непов’язані із університетом користувачі, що можуть бути зацікавлені в проведенні різноманітних подій та інформуванні про них певної цільової аудиторії користувачів.

Для розробки даного програмного застосунку було здійснено порівняльний аналіз із існуючою системою.

Система має переваги у вигляді сучасного інтерфейсу, сповіщення про наближення події через соціальну мережу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visual Studio 2022 – продукт Microsoft [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/.
2. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://open.zeba.academy/pochemu-vscode-populyaren/.
3. Adobe XD [Електронний ресурс] – Режим доступу до ресурсу: https://softlist.com.ua/catalog/product-adobe-xd/.
4. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/use-postman-in-testing/.
5. Ngrok [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/ngrok-localtunnel/.
6. GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://blog.desdelinux.net/ru/github-vs-gitlab/.
7. What is Agile? What is Scrum [Електронний ресурс] // Cprime – Режим доступу до ресурсу: https://www.cprime.com/resources/what-is-agile-what-is-scrum/
8. What are sprints? [Електронний ресурс] // Atlassian – Режим доступу до ресурсу: https://www.atlassian.com/agile/scrum/sprints
9. What is Github? [Електронний ресурс] // Kinsta – Режим доступу до ресурсу: https://kinsta.com/knowledgebase/what-is-github/
10. What is Hangfire? [Електронний ресурс] // codewithmukesh – Режим доступу до ресурсу: https://codewithmukesh.com/blog/hangfire-in-aspnet-core-3-1/#What_is_Hangfire
11. What is database? [Електронний ресурс] // Oracle – Режим доступу до ресурсу: https://www.oracle.com/database/what-is-database/
12. ASP.NET Identity [Електронний ресурс] // Scott Brady 91 – Режим доступу до ресурсу: https://www.scottbrady91.com/aspnet-identity
13. Entity Framework Core [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/core/

14. Architectural Patterns, Pethuru Raj, Anupama Raman, Harihara Subramanian, 2017
15. What is N-Tier Architecture? How It Works, Examples, Tutorials, and More [Електронний ресурс] // Stackify – Режим доступу до ресурсу: https://stackify.com/n-tier-architecture
16. What is a multi layered software architecture? [Електронний ресурс] // Packtpub – Режим доступу до ресурсу: https://hub.packtpub.com/what-is-multi-layered-software-architecture
17. What is the difference between authentication and authorization? [Електронний ресурс] // Sailpoint – Режим доступу до ресурсу: https://www.sailpoint.com/identity-library/difference-between-authentication-and-authorization/
18. How to use axios with react? [Електронний ресурс] // Freecodemap – Режим доступу до ресурсу: https://www.freecodecamp.org/news/how-to-use-axios-with-react/#what-is-axios
19. Getting started with React Router [Електронний ресурс] // React router – Режим доступу до ресурсу: https://reactrouter.com/docs/en/v6/getting-started/tutorial

ДОДАТОК А

Система подій та анонсів

Лістінг програми

Аркушів 5

Острог 2022

IGenericRepository.cs:

```
using Microsoft.EntityFrameworkCore.Query;
using Notifications.DAL.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using X.PagedList;

namespace Notifications.BL.IRepository
{
    public interface IGenericRepository<T> where T : class
    {
        Task<IList<T>> GetAll(
            Expression<Func<T, bool>> expression = null,
            Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null,
            List<string> includes = null
        );
        Task<IPagedList<T>> GetAll(RequestParams requestParams, List<string> includes
= null);
        Task<T>  Get(Expression<Func<T,  bool>>  expression,  List<string>  includes  =
null);
        Task<T> GetFirstOrDefault(Expression<Func<T, bool>> selector,
                                            Expression<Func<T, bool>> predicate = null,
                                            Func<IQueryable<T>,   IOrderedQueryable<T>>
orderBy = null,
                                            Func<IQueryable<T>,
IIncludableQueryable<T, object>> include = null);

        public Task<IList<T>> GetAllHere(Expression<Func<T, bool>> selector = null,
                                            Expression<Func<T, bool>> predicate = null,
                                            Func<IQueryable<T>,   IOrderedQueryable<T>>
orderBy = null,
                                            Func<IQueryable<T>,
IIncludableQueryable<T, object>> include = null);
        bool Exists(object primaryKey);
        Task Insert(T entity);
        Task InsertRange(IEnumerable<T> entities);
        Task Delete(long id);
        void DeleteRange(IEnumerable<T> entities);
        void Update(T entity);
    }
}
```

IUnitOfWork.cs

```
using Notifications.DAL.Models;
using System;
using System.Threading.Tasks;

namespace Notifications.BL.IRepository
{
    public interface IUnitOfWork : IDisposable
    {
        IGenericRepository<Category> Categories { get; }
        IGenericRepository<Event> Events { get; }
        IGenericRepository<NotificationType> NotificationTypes { get; }
        IGenericRepository<EventCategory> EventCategories { get; }
        IGenericRepository<NotificationTypeSubscription>
NotificationTypeSubscriptions { get; }
```

```
        IGenericRepository<Subscription> Subscriptions { get; }
        IGenericRepository<SubscriptionEvent> SubscriptionEvents { get; }
        IGenericRepository<ApplicationUser> Users { get; }
        Task Save();
    }
}
```

GenericRepository.cs:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Query;
using Notifications.BL.IRepository;
using Notifications.DAL.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using X.PagedList;

namespace Notifications.BL.Repository
{
    public class GenericRepository<T> : IGenericRepository<T> where T : class
    {
        readonly NotificationsContext context;
        readonly DbSet<T> db;
        public GenericRepository(NotificationsContext context)
        {
            this.context = context;
            this.db = this.context.Set<T>();
        }
        public async Task Delete(long id)
        {
            var entity = await db.FindAsync(id);
            db.Remove(entity);
        }

        public void DeleteRange(IEnumerable<T> entities)
        {
            db.RemoveRange(entities);
        }

        public async Task<T> Get(Expression<Func<T, bool>> expression, List<string>
includes = null)
        {
            IQueryable<T> query = db;
            if (includes != null)
            {
                foreach (var includeProperty in includes)
                {
                    query = query.Include(includeProperty);
                }
            }

            return await query.AsNoTracking().FirstOrDefaultAsync(expression);
        }

        public async Task<T> GetFirstOrDefault(Expression<Func<T, bool>> selector,
                                               Expression<Func<T, bool>> predicate = null,
                                               Func<IQueryable<T>,   IOrderedQueryable<T>>
orderBy = null,
                                               Func<IQueryable<T>,
IIncludableQueryable<T, object>> include = null)
        {
```

```
            IQueryable<T> query = db;

            if (include != null)
            {
                query = include(query);
            }

            if (predicate != null)
            {
                query = query.Where(predicate);
            }

            if (orderBy != null)
            {
                query = orderBy(query);
            }

            return await query.AsNoTracking().FirstOrDefaultAsync(selector);
        }

        public async Task<IList<T>> GetAllHere(Expression<Func<T, bool>> selector,
                                               Expression<Func<T, bool>> predicate = null,
                                               Func<IQueryable<T>,   IOrderedQueryable<T>>
orderBy = null,
                                               Func<IQueryable<T>,
IIncludableQueryable<T, object>> include = null)
        {
            IQueryable<T> query = db;

            if (include != null)
            {
                query = include(query);
            }

            if (predicate != null)
            {
                query = query.Where(predicate);
            }

            if (orderBy != null)
            {
                query = orderBy(query);
            }

            if (selector != null)
            {
                return await query.AsNoTracking().Where(selector).ToListAsync();
            }

            return await query.AsNoTracking().ToListAsync();
        }

        public async Task<IList<T>> GetAll(Expression<Func<T, bool>> expression =
null, Func<IQueryable<T>, IOrderedQueryable<T>> orderBy = null, List<string> includes
= null)
        {
            IQueryable<T> query = db;

            if (expression != null)
            {
                query = query.Where(expression);
            }

            if (includes != null)
            {
                foreach (var includeProperty in includes)
                {
```

```
                    query = query.Include(includeProperty);
                }
            }

            if (orderBy != null)
            {
                query = orderBy(query);
            }

            return await query.AsNoTracking().ToListAsync();
        }

        public async Task<IPagedList<T>> GetAll(RequestParams requestParams,
List<string> includes = null)
        {
            IQueryable<T> query = db;

            if (includes != null)
            {
                foreach (var includeProperty in includes)
                {
                    query = query.Include(includeProperty);
                }
            }

            return                                                          await
query.AsNoTracking().ToPagedListAsync(requestParams.PageNumber,
requestParams.PageSize);
        }

        public bool Exists(object primaryKey)
        {
            return db.Find(primaryKey) == null ? false : true;
        }

        public async Task Insert(T entity)
        {
            await db.AddAsync(entity);
        }

        public async Task InsertRange(IEnumerable<T> entities)
        {
            await db.AddRangeAsync(entities);
        }

        public void Update(T entity)
        {
            db.Attach(entity);
            context.Entry(entity).State = EntityState.Modified;
        }
    }
}
```

UnitOfWork.cs:

```
using Notifications.BL.IRepository;
using Notifications.DAL.Models;
using System;
using System.Threading.Tasks;

namespace Notifications.BL.Repository
{
    public class UnitOfWork : IUnitOfWork
    {
        readonly NotificationsContext context;
```

```
        IGenericRepository<Category> categories;
        IGenericRepository<Event> events;
        IGenericRepository<EventCategory> eventCategories;
        IGenericRepository<NotificationType> notificationTypes;
        IGenericRepository<NotificationTypeSubscription>
notificationTypeSubscriptions;
        IGenericRepository<Subscription> subscriptions;
        IGenericRepository<SubscriptionEvent> subscriptionEvents;
        IGenericRepository<ApplicationUser> users;
        public UnitOfWork(NotificationsContext context)
        {
            this.context = context;
        }
        public  IGenericRepository<Category>  Categories  =>  categories  ??=  new
GenericRepository<Category>(context);

        public    IGenericRepository<Event>    Events    =>    events    ??=    new
GenericRepository<Event>(context);

        public IGenericRepository<EventCategory> EventCategories => eventCategories
??= new GenericRepository<EventCategory>(context);

        public     IGenericRepository<NotificationType>     NotificationTypes     =>
notificationTypes ??= new GenericRepository<NotificationType>(context);

        public                        IGenericRepository<NotificationTypeSubscription>
NotificationTypeSubscriptions    =>    notificationTypeSubscriptions    ??=    new
GenericRepository<NotificationTypeSubscription>(context);

        public IGenericRepository<Subscription> Subscriptions => subscriptions ??= new
GenericRepository<Subscription>(context);

        public    IGenericRepository<SubscriptionEvent>    SubscriptionEvents    =>
subscriptionEvents ??= new GenericRepository<SubscriptionEvent>(context);

        public  IGenericRepository<ApplicationUser>  Users  =>  users  ??=  new
GenericRepository<ApplicationUser>(context);

        public void Dispose()
        {
            context.Dispose();  // Dispose of all the memory context was using
            GC.SuppressFinalize(this);
        }

        public async Task Save()
        {
            await context.SaveChangesAsync();
        }
    }
}
```

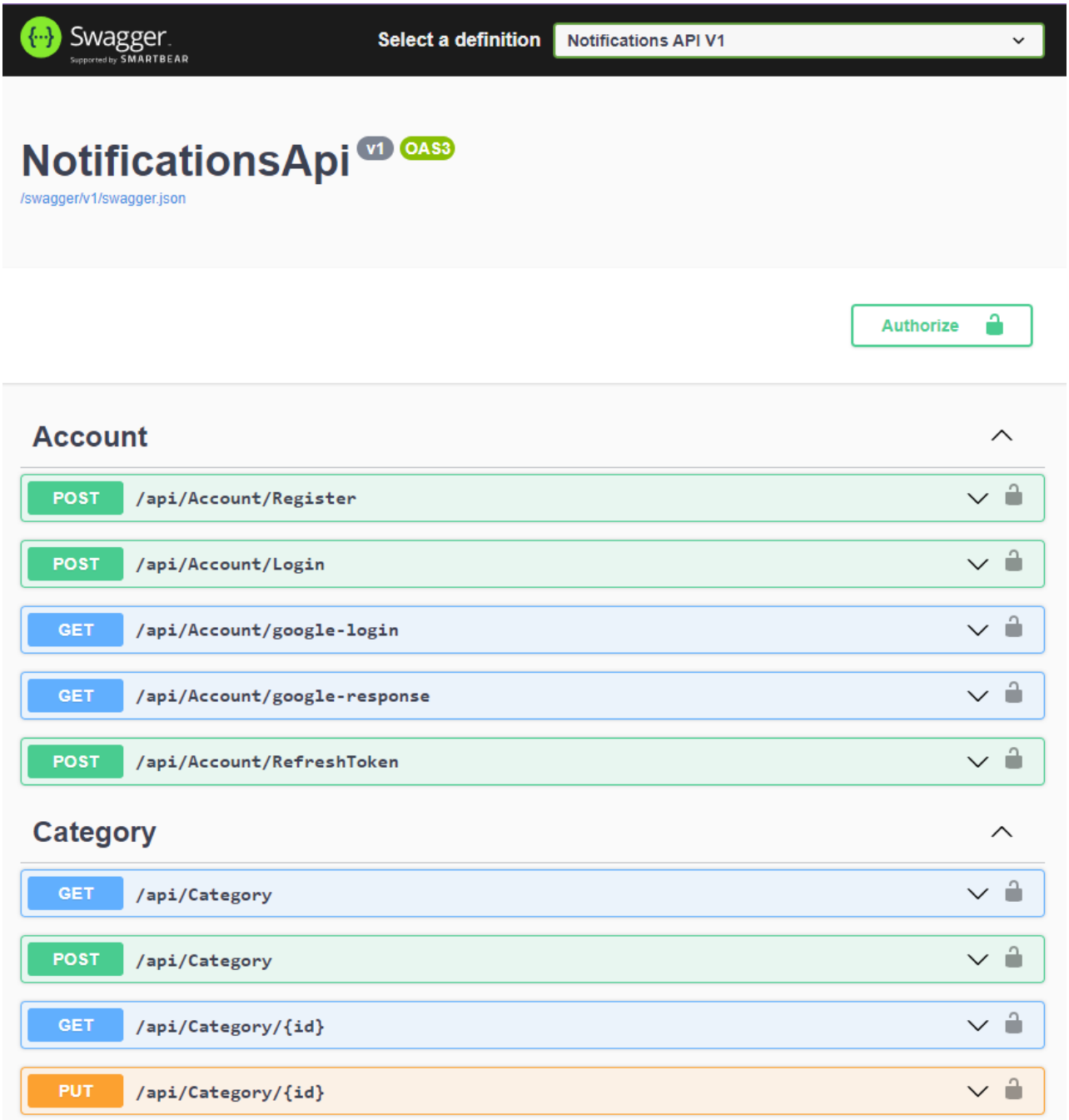
ДОДАТОК Б

Система подій та анонсів

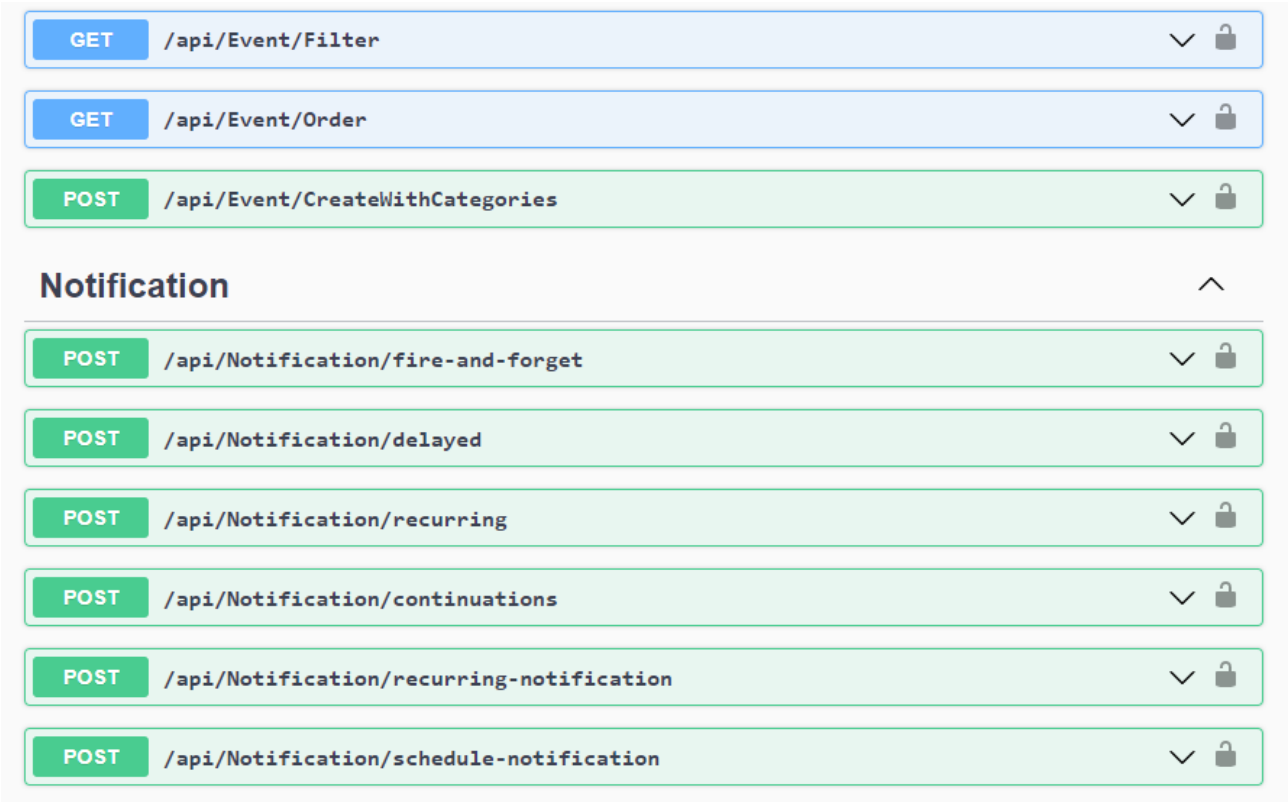
Набір інструментів Swagger

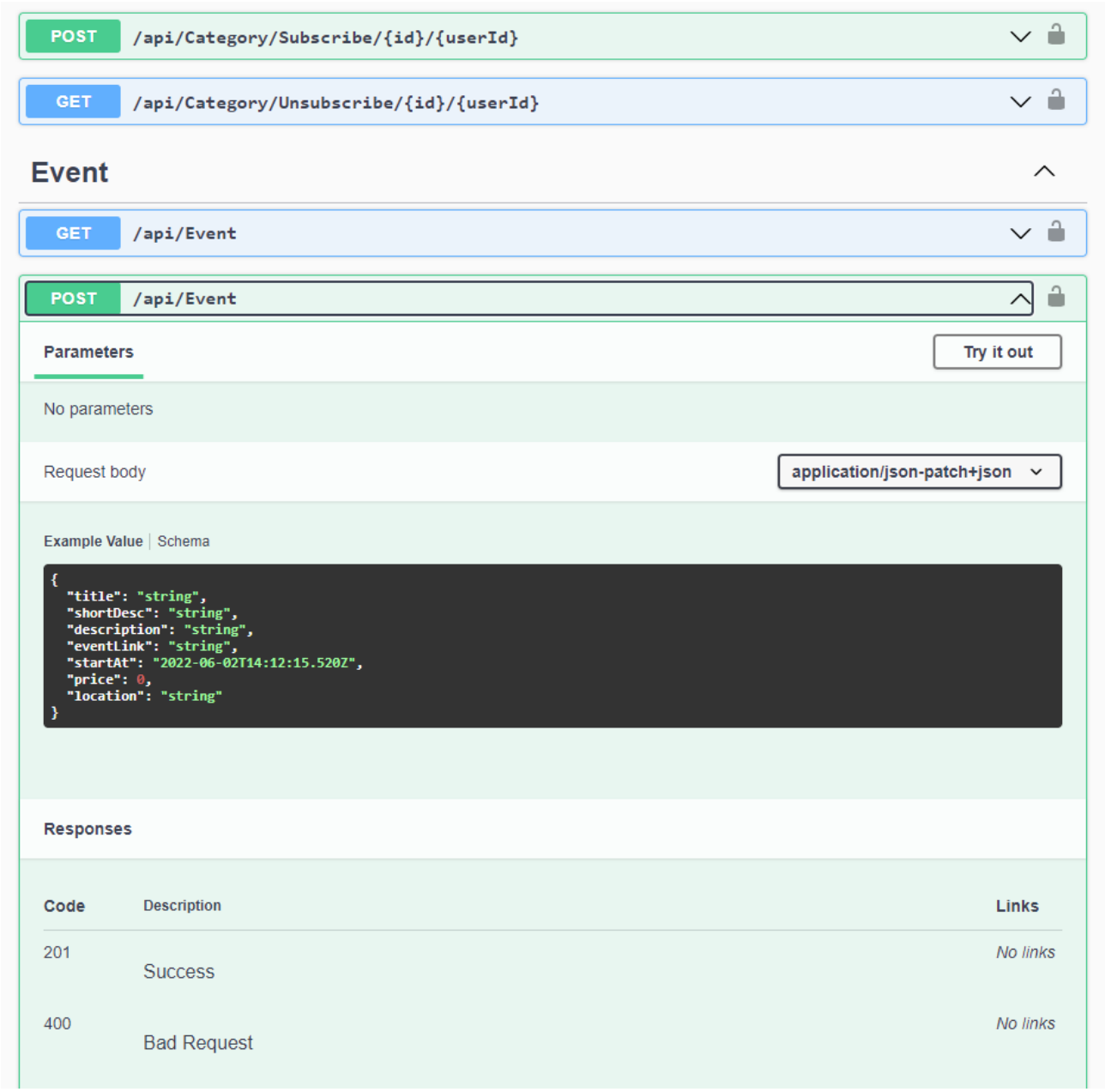
Аркушів 4

Острог 2022


Swagger
Supported by SMARTBEAR
Select a definition
Notifications API V1
NotificationsApi
v1
OAS3
/swagger/v1/swagger.json
Authorize
Account
POST /api/Account/Register
POST /api/Account/Login
GET /api/Account/google-login
GET /api/Account/google-response
POST /api/Account/RefreshToken
Category
GET /api/Category
POST /api/Category
GET /api/Category/{id}
PUT /api/Category/{id}


DELETE /api/Category/{id}
GET /api/Category/GetCategoryEvents/{id}
POST /api/Category/Subscribe/{id}/{userId}
GET /api/Category/Unsubscribe/{id}/{userId}
Event
GET /api/Event
POST /api/Event
GET /api/Event/{id}
DELETE /api/Event/{id}
PUT /api/Event/{EventId}
PUT /api/Event/{EventId}/{CategoryId}
GET /api/Event/GetUsers/{EventId}
PUT /api/Event/AddCategories/{EventId}
POST /api/Event/{EventId}/{TelegramId}
GET /api/Event/{EventId}/{TelegramId}
GET /api/Event/Telegram/{TelegramId}
GET /api/Event/Search


GET /api/Event/Filter
GET /api/Event/Order
POST /api/Event/CreateWithCategories
Notification
POST /api/Notification/fire-and-forget
POST /api/Notification/delayed
POST /api/Notification/recurring
POST /api/Notification/continuations
POST /api/Notification/recurring-notification
POST /api/Notification/schedule-notification


POST /api/Category/Subscribe/{id}/{userId}
GET /api/Category/Unsubscribe/{id}/{userId}
Event
GET /api/Event
POST /api/Event
Parameters
Try it out
No parameters
Request body
application/json-patch+json
Example Value | Schema
{
"title": "string",
"shortDesc": "string",
"description": "string",
"eventLink": "string",
"startAt": "2022-06-02T14:12:15.520Z",
"price": 0,
"location": "string"
}
Responses
Code
Description
Links
201
Success
No links
400
Bad Request
No links

ДОДАТОК В

Система подій та анонсів

Лістінг програми

Аркушів 3

Острог 2022

AccountController.cs:

```
using AutoMapper;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Authentication.Google;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Notifications.BL.Services;
using Notifications.DAL.Models;
using Notifications.DTO.DTOs;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace Notifications.Api.Controllers
{
    [AllowAnonymous]
    [Route("api/[controller]")]
    [ApiController]
    public class AccountController : ControllerBase
    {
        readonly UserManager<ApplicationUser> userManager;
        readonly ILogger<AccountController> logger;
        readonly IMapper mapper;
        readonly IAuthManager authManager;
        public AccountController(
            UserManager<ApplicationUser> userManager,
            ILogger<AccountController> logger,
            IMapper mapper, IAuthManager authManager)
        {
            this.userManager = userManager;
            this.logger = logger;
            this.mapper = mapper;
            this.authManager = authManager;
        }

        [HttpPost]
        [Route("Register")]
        [ProducesResponseType(StatusCodes.Status200OK)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public async Task<IActionResult> Register([FromBody] UserDTO userDTO)
        {
            logger.LogInformation($"Registration Attempt for {userDTO.Email}");
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            try
            {
                var user = mapper.Map<ApplicationUser>(userDTO);
                user.UserName = userDTO.Email;
                var result = await userManager.CreateAsync(user, userDTO.Password);

                if (!result.Succeeded)
                {
                    foreach (var error in result.Errors) // possibly sensitive error
 info
                    {
                        ModelState.AddModelError(error.Code, error.Description);
                    }
                    return BadRequest(ModelState);
```

```
                }

                if (userDTO.Email.Equals("mykola.kalinichenko@oa.edu.ua"))
                {
                    await userManager.AddToRoleAsync(user, "Admin" );
                    return Accepted();
                }

                await userManager.AddToRolesAsync(user, userDTO.Roles);
                return Accepted();
            }
            catch (Exception ex)
            {
                logger.LogError(ex, $"Something Went Wrong in the {nameof(Register)}
");
                return Problem($"Something Went Wrong in the {nameof(Register)}", st
atusCode: 500);
            }
        }

        [HttpPost]
        [Route("Login")]
        public async Task<IActionResult> Login([FromBody] LoginUserDTO userDTO)
        {
            logger.LogInformation($"Login Attempt for {userDTO.Email}");
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            try
            {
                if (!await authManager.ValidateUser(userDTO))
                {
                    return Unauthorized();
                }

                return Accepted(new { Token = await authManager.CreateToken() });
            }
            catch (Exception ex)
            {
                logger.LogError(ex, $"Something Went Wrong in the {nameof(Login)}");
                return Problem($"Something Went Wrong in the {nameof(Login)}", statu
sCode: 500);
            }
        }

        [HttpGet("google-login")]
        public IActionResult GoogleLogin()
        {
            var props = new AuthenticationProperties { RedirectUri = Url.Action("Goo
gleResponse", "Account") };

            return Challenge(props, GoogleDefaults.AuthenticationScheme);
        }

        [HttpGet("google-response")]
        public async Task<IActionResult> GoogleResponse()
        {
            var result = await HttpContext.AuthenticateAsync(CookieAuthenticationDef
aults.AuthenticationScheme);

            var claims = result.Principal.Identities.FirstOrDefault()
                .Claims.Select(claim => new
                {
                    claim.Issuer,
                    claim.OriginalIssuer,
```

```
                    claim.Type,
                    claim.Value,
                });

            return Accepted(claims);
        }
    }
}
```

ДОДАТОК Г

Система подій та анонсів

Лістінг програми

Аркушів 1

Острог 2022

MapperInitializer.cs:

```
using AutoMapper;
using Notifications.DAL.Models;
using Notifications.DTO.DTOs;

namespace Notifications.DTO.Configurations
{
    public class MapperInitializer : Profile
    {
        public MapperInitializer()
        {
            CreateMap<Category, CategoryDTO>().ReverseMap();
            CreateMap<Category, CreateCategoryDTO>().ReverseMap();
            CreateMap<Event, EventDTO>().ReverseMap();
            CreateMap<Event, CreateEventDTO>().ReverseMap();
            CreateMap<NotificationType, NotificationTypeDTO>().ReverseMap();
            CreateMap<NotificationType, CreateNotificationTypeDTO>().ReverseMap();
            CreateMap<ApplicationUser, UserDTO>().ReverseMap();
        }
    }
}
```