Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Котенко Олександри Павлівни*
(ПІБ)

академічної групи *122-20-1*
(шифр)

спеціальності *122 «Комп'ютерні науки»*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка інформаційної системи для волонтерських організацій «Шлях до перемоги» мовою програмування C#*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Гуліна І.Г.*			
розділів:				
спеціальний	*доц. Гуліна І.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц. Шедловський І.А.*			
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-1* *Котенко Олександри Павлівни*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка інформаційної системи для волонтерських організацій «Шлях до перемоги» мовою програмування C#*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* № *469-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів практик та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видав _____ *доц. Гуліна І.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Котенко О.П.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2024 р.*

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка: 93 с., 45 рис., 3 дод., 17 джерел.

Об'єкт розробки: інформаційна система для волонтерських організацій «Шлях до перемоги».

Мета кваліфікаційної роботи: розробка інформаційної системи для волонтерських організацій на основі технологічного стека ASP.NET Core/Vue.js/TelegramBot Api/MSSQL DB.

У вступі розглядається можливість активної взаємодії волонтерських організацій з суспільством, порушується проблема важливості наявності сервісу для пошуку благодійних заходів та зборів, а також переваги використання цієї інформаційної системи для волонтерських організацій.

У першому розділі проведено дослідження предметної області та аналіз наявних засобів для розв’язання поставленої проблеми, призначення розробки та постановка задачі - постановка технологій, що використовуються, і поверхневий опис можливостей різних типів користувачів. Проведено опитування користувачів.

Другий розділ надає детальний огляд призначення платформи, рівнів доступу користувачів, використаних архітектурних підходів і програмних шаблонів, порядку роботи алгоритмів на сервері, а також обґрунтування вибору технологій для розробки проєкту.

В економічному розділі визначають рівень складності проєкту, проводять розрахунок вартості розроблення інформаційної платформи залежно від різних параметрів (об’єму коду, ресурсів, електроенергії, Інтернету тощо), а також визначають запланований час на його створення.

Практичне значення роботи полягає у розробці інформаційної системи, на якій волонтерські організації зможуть публікувати благодійні заходи та збори, і зацікавлені люди та інші волонтери можуть знайти події та грошові збори для підтримки ЗСУ тощо.

Актуальність платформи обумовлюється потребою збройних сил України в додатковому екіпіруванні, а також потребами людей які витралили житло, тварин евакуйованих з окупованих територій.

Список ключових слів: ІНФОРМАЦІЙНА СИСТЕМА, БРАУЗЕР, ASP.NET, MSSQL, VUE.JS, TELEGRAM, ВОЛОНТЕР, ВОЛОНТЕРСЬКА ОРГАНІЗАЦІЯ, ЗБІР, БЛАГОДІЙНИЙ ЗАХІД.

ABSTRACT

Explanatory note: 93 p., 45 figs., 3 app., 17 sources.

Object of development: information system for volunteer organisations "The Way to Victory".

The purpose of the qualification work: development of an information system for volunteer organisations based on the ASP.NET/Vue.js/TelegramBot Api/MSSQL DB technology stack.

The introduction discusses the possibility of active interaction between volunteer organisations and society, raises the issue of the importance of having a service for searching for charity events and meetings, as well as the benefits of using this information system for volunteer organisations.

The first chapter contains a study of the subject area and an analysis of the available tools for solving the problem, the purpose of development and the task statement - the definition of the technologies used and a superficial description of the capabilities of different types of users. A user survey was conducted.

The second section provides a detailed overview of the purpose of the platform, user access levels, architectural approaches and software templates used, the way algorithms work on the server, and the rationale for choosing technologies for the project.

The economic section determines the level of complexity of the project, calculates the cost of developing the information platform depending on various parameters (code volume, resources, electricity, Internet, etc.), and determines the planned time for its creation.

The practical significance of the work is to develop an information system where volunteer organisations can publish charity events and fundraisers, and where interested people and other volunteers can find events and fundraisers to support the Armed Forces, etc.

The relevance of the platform is due to the need of the Ukrainian armed forces for additional equipment, as well as the needs of people who have lost their homes and animals evacuated from the occupied territories.

List of keywords: INFORMATION SYSTEM, BROWSER, ASP.NET, MSSQL, VUE.JS, TELEGRAM, VOLUNTEER, VOLUNTEER ORGANISATION, COLLECTION, CHARITY EVENT.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface;

HTML – HyperText Markup Language;

CSS – Cascading Stylesheets;

JS – JavaScript;

SQL – Structured Query Language;

MS SQL – Microsoft Structured Query Language;

JWT – JSON web token;

JSON – JavaScript Object Notation;

HTTP – Hypertext Transfer Protocol;

DTO – Data transfer object.

ВСТУП

В сучасних реаліях українського світу волонтерство набуває ключової ролі. Наразі наша країна стикається з низкою складних викликів, серед яких військовий конфлікт, що вимагає залучення значних ресурсів для захисту суверенітету та територіальної цілісності. Допомоги потребує велика кількість населення будь це військові на фронті, чи люди, що лишились без житла, діти без батьків. Це охоплює надання тимчасового притулку, медичної допомоги, їжі, одягу та інших соціальних послуг. Крім того, волонтери займаються зоозахистом, пропонуючи допомогу безпритульним тваринам.

Волонтерство багатогранне та не має меж. З кожним днем охочих допомогти більшає, що не може не радувати. Однак, існує значна проблема з доступністю інформації про благодійні заходи та збір коштів, як для волонтерських організацій, так і для людей, які бажають запропонувати свою допомогу.

Зростає потреба в універсальному ресурсі, який міг би поєднати організації, що потребують допомоги та людей, які прагнуть допомогти. Така платформа, спрямована на координацію та співпрацю між організаціями та волонтерами, є надзвичайно важливим та дієвим інструментом у сфері соціальної допомоги. Це може сприяти ефективній організації ресурсів, забезпечити відкритість і прозорість допомоги, а також заохотить більшу кількість громадян до участі у волонтерській діяльності.

Основні переваги створення та використання такої інформаційної системи полягають в:

– легкому доступі до інформації в одному місці
– різні види допомоги, кожен матиме змогу знайти саме те, в чому він може допомогти
– актуальність та сповіщення про нові потреби, можливості
– швидкий перехід до зборів коштів
– популяризація волонтерської діяльності

Наразі єдиним методом отримання інформації про потребу в допомозі є соціальні мережі. Організації створюють Instagram сторінки або Telegram канали де публікують збори. Тобто користувачам необхідно відстежувати велику кількість сторінок та каналів у соціальних мережах, що не є зручним.

Метою дипломного проєкту є створення інформаційної системи, яка дала б змогу користувачеві отримувати доступ до інформації, яка їх цікавить в одному місці. Та у додаток Telegram бот, який буде сповіщати користувачів про усі нові події, що відповідають заданим фільтрам пошуку. Легкий та зрозумілий інтерфейс у вигляді календаря подій також посилить ефективність допомоги.

Зваживши усі речі описані вище було прийнято рішення розробити дипломну роботу обравши тему: «Інформаційна система для волонтерських організацій «Шлях до перемоги»».

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

У сучасному суспільстві волонтерство стає все більш популярною та визнаною соціальною діяльністю. Ця сфера є, була та буде відігравати ключову роль в багатьох аспектах нашого життя. Поняття благодійності та волонтерства можна розглядати під різними кутами, вони зачіпають не маленьку кількість різних сфер діяльності:

– Соціальна допомога. Волонтери й організації працюють у сферах, пов’язаних з підтримкою вразливих груп населення. Надають психічну допомогу, тимчасовий притулок та речі першої потреби людям які цього потребують.

– Охорона довкілля. Волонтери й організації можуть брати участь в екологічних програмах за для очищення довкілля від сміття, висадження дерев тощо. Популярним є масові прибирання паркових та лісових зон щосуботи з усіма небайдужими мешканцями міст.

– Медична допомога. Робота в лікарнях, медичних центрах або різних місіях, де потрібна медична допомога. В багатьох людей недостатньо коштів для отримання якісної медичної допомоги або бракує знань в сфері здоров’я. Тому лікарі влаштовують освітні сесії та безкоштовні консультації.

– Освіта. Дуже часто сьогодні допомоги потребують школи та інші навчальні заклади, особливо ті, які приймають переселенців дітей, які втратили свої домівки. До того ж якісна освіта відіграє ключову роль у вихованні нових поколінь, для розвитку наукового потенціалу нації та її економічного зростання. Освітні платформи надають безплатний доступ до своїх ресурсів, влаштовують лекції та майстеркласи.

– Культура та мистецтво. Допомоги іноді потребують навіть культурні діячі, що популяризують українські витвори мистецтва, чи реставрують або повністю відновлюють пам’ятки культури.

На цей час не тільки в Україні, але і в усьому світі існує дуже багато організацій, які знаходять небайдужих волонтерів, та об’єднують їх сили за для досягнення більших результатів в перерахованих вище сферах діяльності. На жаль варто констатувати, що зі збільшенням кількості волонтерів, які працюють над спільними проєктами, пропорційно збільшується і кількість процесів, за якими важко встигати та якими дуже важко керувати волонтерським очільникам.

Звертаючись до прикладів волонтерства, за якими ми сьогодні можемо спостерігати та які ми можемо аналізувати, варто навести реальні приклади актуальної волонтерської діяльності та навести перелік тих проблем з якими сьогодні стикається людина, яка хоча долучитися до подібної громадської організації.

Припустимо, що на фронті військовій дивізії чи роті треба автомобіль для переміщення особистого складу в зоні військових дій. Деяка волонтерська організація починає збір коштів на цей автомобіль, одразу після того як до них звертаються з проханням про допомогу. Небайдужі громадяни поступово заповнюють цей збір своїми коштами. З цього моменту починаються проблеми, адже для пошуку автомобіля для військових в зібраному грошовому діапазоні треба спочатку знайти волонтера, який розбирається в автомобілях, після цього знайти людей, які погодяться поїхати за цим автомобілем, а в кінці цей автомобіль треба перегнати до військових, можливо навіть на передову.

Основна проблема, яку можна спостерігати в такій волонтерській діяльності, це збір самих волонтерів, комунікація з ними, та пильна увага до деталей, які виникають під час, описаних на прикладі вище, різних етапів волонтерської діяльності. Для полегшення власної роботи, дуже часто волонтери створюють групи в соціальних мережах, таких як наприклад Facebook, Twitter, Telegram тощо. Подібні рішення дещо покращують ситуацію, але дані

платформи все одно дещо не пристосовані до введення складної публічної громадської волонтерської діяльності.

Розглянемо приклади функціонуючих благодійних організацій України.

Одним з самих масштабних фондів, що займається зборами для військового фронту – є фонд Сергія Притули. В цьому фонді працює понад 200 волонтерів з різних куточків країни та сфер діяльності. Його було засновано у 2020 році в місті Київ і зараз фонд зростає з шаленими темпами[1]. Росте кількість волонтерів та партнері, що роблять великі внески у перемогу. Важливо зазначити, що бізнес гіганти теж не залишаються осторонь та вкладають мільйони гривень зі свого прибутку у збори.

Найпопулярнішим збором фонду Сергія Притули є проєкт «Народний Байрактар». Під час даного збору всього за три було зібрано 600 мільйонів гривень. На ці кошти було придбано супутник ICEYE, що підсилило ефективність роботи збройних сил України.

Принцип роботи фонду з запитами на допомогу від військових досить легкий через свою структурованість, але максимально надійний. Волонтери відповідальні за отримання запитів комунікують з військовими підрозділами про необхідну допомогу. Наступним кроком є верифікація та аналіз заявки. Під час верифікації визначається правдивість отриманої інформації, факт існування та потреб підрозділу що відправив запит. Відштовхуючись від кількості запитів та їх терміновості команда волонтерів визначає пріоритети та відправляє запит у роботу. Далі відбувається закупка та передала необхідних речей, отримання та публікація звітності, що є неодмінним та надзвичайно важливим кроком, аби люди знали про кінцеве призначення їх пожертвувань та їх цінність.

Головним ресурсом для залучення донатів є соціальні мережі та взаємодія з інфлюєнсерами. Фонд Сергія Притули має свій власний вебсайт та сторінки в багатьох соціальних мережах. Сайт містить зручний елемент для пожертвування

коштів на рахунки фонду. Також усі звіти про минулі та активні проєкти відображенні на сайті.

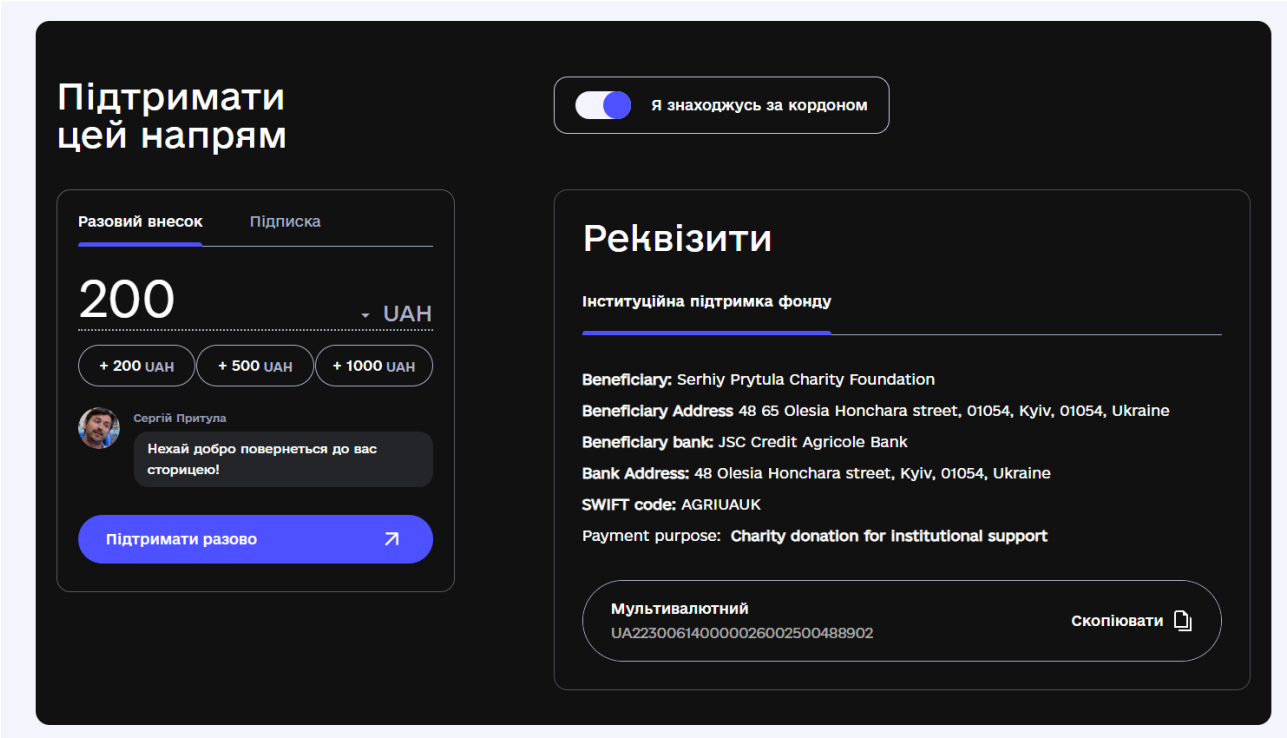

Рис. 1.1. Вебсайт фонду Сергія Притули

Ще одними цікавим проєктом є волонтерська ініціатива «rechi.support». Ця організація в переважній більшості надає допомогу родинам, які були вимушені переселитись в інші міста України через військові дії. Головним інструментом даної ініціативи є сторінка у соціальній мережі Instagram. Проєкт об’єднує людей яким потрібна допомога та тих хто має можливість її надати. Як можна зрозуміти з назви, основний ресурс – це речі [2].

Волонтери працюють за схожою схемою як і фонд Сергія Притули. Одним з найголовніших етапів опрацювання запитів після їх отримання є верифікація. Дані тимчасово переміщених родин ретельно перевіряються перш ніж перейти до наступного кроку. Після перевірки запит направляється до донора – людини, що має змогу та бажання його виконати. Донори самостійно надсилають речі до родин. Однією з останніх подій від «rechi.support» є ініціатива Нагодуй

пухнастих ВПО для підтримки рівня здорового харчування тваринок, які були вимушені переїхати. Волонтери використовують Google Forms для збору заявок від охочих надати допомогу та просувають свою ініціативу в соціальних мережах.

1.2. Призначення розробки та галузь застосування

Галузь застосування інформаційної системи для волонтерських організацій «Шлях до перемоги» досить широка та зачіпає усі верстви населення. Для розуміння потреб користувача та надання максимальної користі нашою інформаційною платформною було проведено опитування користувачів. В результаті опитування тридцяти респондентів було визначено найголовніші проблеми з якими стикаються люди під час своєї благодійної діяльності.

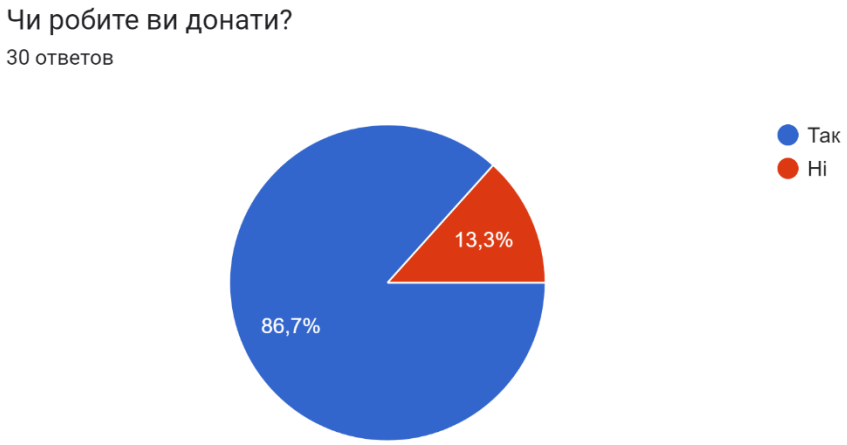

Рис. 1.2. Результат опитування

86,7% опитаних нами людей роблять донати та долучаються до зборів, що є показником актуальності наявності платформи, яка могли б полегшити цей процес. Однак користувачі стикаються з певними проблемами. Зокрема, 61,5% зазначили, що відчувають сумніви з приводу того чи дійдуть гроші до свого місця призначення та 46,2% часто не довіряють засновникам зборів.

Таким чином можна підсумувати, що існує нагальна потреба в створенні платформи, яка б забезпечувала прозорість і підвищувала довіру до процесу збору коштів. Це допоможе розвіяти сумніви та підвищити впевненість користувачів у тому, що їхні пожертви будуть використані за призначенням.

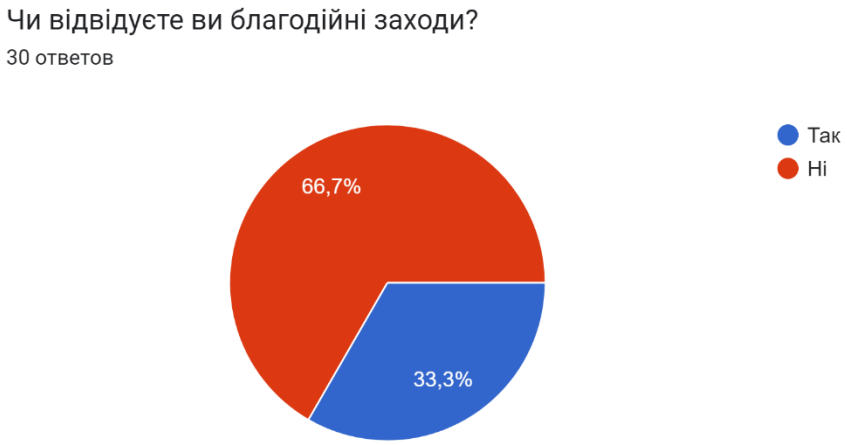

Рис. 1.3. Результат опитування

Благодійні заходи не користуються такою самою популярністю як збори, проте вони залишаються досить актуальними. 33,3% опитаних респондентів зазначили, що відвідують благодійні заходи, з них 70% роблять це 2-3 рази на рік. Це свідчить про те, що хоча благодійні заходи менш популярні, вони все ж мають значну аудиторію, яка регулярно бере в них участь.

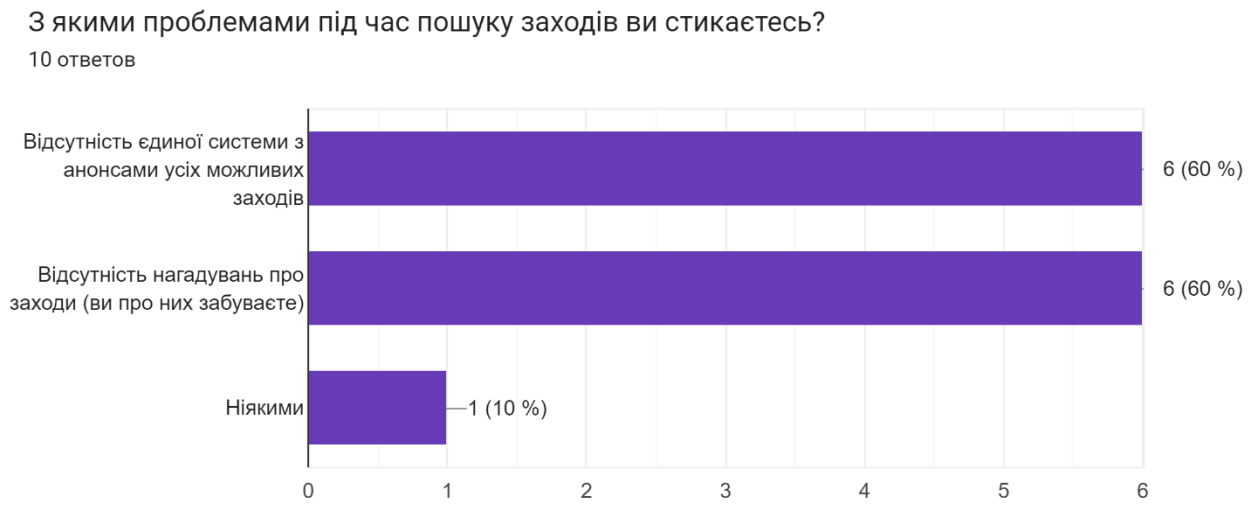

Рис. 1.4. Результат опитування

На питання про те, з якими проблемами стикаються люди під час пошуку благодійних заходів, було отримано цікаві результати. 60% поскаржились на відсутність єдиної системи з анонсами усіх можливих заходів. Це означає, що людям важко знайти інформацію про всі заходи, які відбуваються в їхньому місті, що ускладнює їх участь в них. Ще 60% відповіли, що забувають про заходи на які вони хотіли б піти. Ця проблема може бути пов'язана з відсутністю нагадувань або систем планування, які допомагають людям запам'ятовувати дати і час проведення благодійних заходів. Люди можуть записати інформацію про подію, але через брак нагадувань забувають про неї з наближенням дати.

На основі проведеного опитування користувачів було виявлено основні проблеми, з якими вони стикаються, а також їх потреби та очікування. Після чого було розроблено бізнес-модель для інформаційної системи.

Рис. 1.5. Бізнес-модель

Нижче буде наведено детальний опис деяких пунктів створеної бізнес-моделі. Визначені наступні ключові партнери:

1. Благодійні організації та фонди – це самий головний партнер для даної інформаційної системи, оскільки саме організації проводять різноманітні волонтерські активності та надають інформацію про них.

2. Урядові та громадські органи – влада теж має сильний вплив на благодійну діяльність, від урядових установ необхідно отримувати інформацію для перевірки тих чи інших організацій та волонтерів. Також громадські органи сприяють фондам та несуть свій вклад.

3. Компанії та корпорації – один з основних ресурсів залучення коштів на проєкти та збори.

Унікальна ціннісна пропозиція:

1. Легкий доступ до інформації, заходів та зборів. Як вже було визначено у опитуванні, користувачам не вистачає ресурсу в якому було б об’єднано всі можливі заходи та активності. Наявність такої інформаційної системи значно підвищить ефективність донесення інформації від її творців до споживачів.

2. Аналітика та звітність. Для підвищення довіри до зборів користувачам важливо бачити куди йдуть їх кошти. Організації матимуть змогу публікувати звітність про закупи спорядження, автомобілів, товарів тощо безпосередньо на інформаційній платформі та мати можливість змінювати статус зборів для їх відстежування.

3. Взаємодія та зворотний зв’язок. Користувачам буде надана підтримка у разі технічних чи будь-яких питань та проблем.

4. Швидкі донати. Для користувача важливо мати можливість зробити внесок без додаткових зусиль. За результатами опитування більшість використовує саме Монобанк для здійснення пожертвувань. Дана платформа планує впровадити інтеграцію з Монобанком для зручних переводів коштів на самій платформі.

Таким чином, інформаційна система «Шлях до перемоги» несе значну цінність для соціальної діяльності, пропонуючи універсальний інструмент для залучення громадян до благодійних та волонтерських ініціатив, підтримки благодійних організацій та розвитку активного громадянського суспільства.

1.3. Підстави для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

− освітня програма 122 Комп’ютерні науки;

− навчальний план та графік навчального процесу;

− наказ ректора Національного технічного університету «Дніпровська політехніка» № 469-с від 23.05.2024 р;

− завдання на кваліфікаційну роботу на тему «Розробка інформаційної системи для волонтерських організацій «Шлях до перемоги» мовою програмування C#».

1.4. Постановка завдання

Завдання: розробити інформаційну систему мовою програмування C# із використанням технології ASP.NET для API та HTML/CSS/JS/Vue.js для front-end частини. Розробити Telegram бот для оповіщення волонтерів про нові заходи.

У таблиці 1.1 наведено функціонал, який має реалізовувати інформаційна платформа для кожного типу користувачів

Таблиця 1.1

Можливості користувачів (об’єктів) при роботі з інформаційною системою

Об’єкт	Можливості
Волонтер	Реєстрація, авторизація, доєднання до наявних організацій, створення нових організацій, перегляд та фільтрація подій та зборів, отримання повідомлень про нові події у Telegram боті
Засновник організації	Реєстрація, авторизація, створення нових подій та зборів, зміна статусу та публікація підсумків подій.
Адміністратор	Вхід до системи, перевірка та підтвердження організацій та створених подій, зборів

Для створення інформаційної платформи потрібно:

- Проаналізувати предметну галузь на наявні рішення.
- Обрати оптимальне архітектурне рішення.
- Розробити API та спроектувати базу даних.
- Розробити front-end та інтерфейс.

1.5. Вимоги до програми або програмного виробу

1.5.1 Вимоги до функціональних характеристик

Інформаційна система для волонтерських організацій «Шлях до перемоги» має складатися з двох частин - серверної та клієнтської. Серверна частина повинна бути виконана мовою програмування C# з використанням платформи .NET у вигляді WEB API. Клієнтська частина має бути розроблена з допомогою VUE.JS та підтримувати доступ клієнтів з таких браузерів як Google, Firefox, Microsoft Edge та інші.

Для досягнення визначених цілей, розроблене програмне забезпечення має забезпечувати реалізацію наступних функцій:

- Інтуїтивно зрозумілий користувачу дизайн сайту.
- Швидка передача даних між сервером та клієнтською частиною.
- У сайту має бути адаптивний дизайн для мобільних пристроїв.

1.5.2 Вимоги до інформаційної безпеки

Важливим аспектом нашої платформи є інформаційна безпека. Наші користувачі повинні бути впевнені в тому, що їх дані надійно збережені та ніхто не зможе скористуватися ними. Для забезпечення інформаційної цілісності нашої системи будуть використані наступні підходи:

- Створення запитів з використанням протоколу HTTP між серверною та клієнтською частиною за для шифрування контенту запитів.
- Аутентифікація та авторизація. Цей підхід дасть змогу користуватись нашою платформою лише зареєстрованим користувачам.
- Збереження паролю користувачів у вигляді хешу. Оскільки жодна організація не застрахована від витоку даних, важливо не дати зловмисникам можливості скористатись даними користувачів для аутентифікації.

– Підтвердження пошти за якою реєструється користувач, за для захисту від атак ботів.

– Зберігання стану БД на кожному етапі проєктування системи.

1.5.3. Вимоги до складу та параметрів технічних засобів

Інформаційна система для волонтерських організацій «Шлях до перемоги» буде веб застосунком з клієнтською та серверною частиною. Платформа буде мати адаптивний дизайн для різних пристроїв, таких як телефон, комп'ютер, ноутбук та планшет. У ході виконання поставленого технічного завдання, досягнення адаптивності буде описано більш детально з урахуванням завдання та чітким описом використаного інструментарію для досягнення цієї мети.

Веб застосунок «Шлях до перемоги» буде доступний з усіх популярних браузерів, що існують сьогодні на різних платформах. Для конкретизації складу та параметрів технічних засобів, варто зауважити, що для браузерів буде використана їх остання актуальна версія, тому і користувачам рекомендується користуватися застосунком лише оновивши власне ПО.

Варто зауважити, що у ході використання платформи передбачаються деякі відмінності у користувацькому досвіді на різних операційних системах та у різних браузерах, але вони будуть не суттєвими і ключовим функціоналом розробленого проєкта можна буде скористатись будь де.

Важливо відмітити, що застосунок буде локалізований під українську мову, та у роботі буде розроблятися лише інтерфейс з її використанням.

В застосунок будуть інтегровані відповідні заходи безпеки для захисту особистих даних користувачів. Це включає в себе шифрування даних, застосування відповідних стандартів безпеки, таких як SSL, а також заходи проти несанкціонованого доступу та зламу. Платформа буде розроблена з урахуванням можливості легкої модифікації та розширення функціоналу в майбутньому. Це дозволить вільно адаптувати застосунок під конкретні потреби конкретної організації чи спільноти.

Основними базовими вимогами до складу та параметрів технічних засобів пристрою використання є:

– Графічна підтримка: Наявність відеокарти або вбудованого графічного процесора, який забезпечує відображення веб-сторінок та графічних інтерфейсів.

– Мережеві можливості: Підтримка мережевого підключення, що дозволяє використовувати Інтернет через Wi-Fi або Ethernet.

– Аудіо вивід: Звукова карта або вбудований аудіо процесор для відтворення звукових ефектів, музики та аудіо з веб-сайтів.

– USB та інші порти: Наявність портів USB для підключення зовнішніх пристроїв, таких як флешки, миші, клавіатури, тощо.

– Екран: Наявність дисплея (екрана) для відображення вмісту, може бути матрицям типу TFT, IPS, OLED тощо з підтримкою роздільної здатності, наприклад, HD, Full HD, або навіть 4K.

– Веб-камера і мікрофон: Наявність вбудованої веб-камери та мікрофона для відеозв'язку та запису аудіо та відео.

– Батарея: Якщо це ноутбук, то він може мати вбудовану батарею для живлення без підключення до мережі.

– Прилади для отримання інформації: тачпад або миша, та клавіатура механічна, або мембрана.

Що стосується браузерів, то вони можуть бути будь-якими головне, щоб вони мали останню актуальну версію оновлення.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для подальшого використання застосунку рекомендується використовувати програми з такими технічними характеристиками:

– Підтримка браузера Chrome версії 81.0.4044.20 або новішої для оптимальної продуктивності та сумісності.

– Сумісність з 32- або 64-розрядними процесорами та операційними системами для максимального охоплення користувачів.

– Операційні системи Windows 10 або 11 для стабільної роботи та уникнення проблем сумісності.

– Наявність клавіатури та миші/тачпаду для повноцінної взаємодії з програмами.

– Мінімум 4 ГБ оперативної пам'яті для ефективної роботи програм.

– Процесор із частотою 2 ГГц або вище для швидкої обробки даних.

– Достатній об'єм жорсткого диска (HDD або SSD) - мінімум 20 ГБ для зберігання тестових даних та зручного доступу до них.

Для використання програм у браузері на мобільних пристроях рекомендуються наступні характеристики:

– Підтримка Chrome для IOS версії 124.0.6367.111 або для Android версії 124.0.6367.113 для оптимальної продуктивності.

– Розмір екрана не менше 6,06 дюймів або 2436x1125 пікселів для зручного користування.

– Операційні системи IOS 15.4.1 або Android 13.

– Наявність сенсорного екрану для взаємодії з програмами на телефонах.

– Мінімум 4 ГБ оперативної пам'яті для плавної роботи програм.

– Процесор із частотою 2 ГГц або вище для швидкого виконання завдань.

– Жорсткий диск (HDD чи SSD) об'ємом не менше 16 ГБ для зберігання даних та програм.

Для використання програм у браузері на планшетах рекомендуються такі характеристики:

– Підтримка Chrome для IOS версії 135.0.6497.100 або для Android версії 135.0.6497.105.

– Розмір екрана не менше 10.5 дюймів або 2560x1600 пікселів.

– Операційні системи IOS 16.2 або Android 14.

– Наявність сенсорного екрану для взаємодії з програмами на планшетах.

– Мінімум 6 ГБ оперативної пам'яті для оптимальної продуктивності.

– Процесор із частотою 2.5 ГГц або вище для швидкої обробки даних.

– Жорсткий диск SSD об'ємом не менше 32 ГБ для ефективної роботи програм.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ
2.1. Функціональне призначення програми

Функціональне призначення інформаційної системи для волонтерських організацій «Шлях до перемоги» полягає в тому, щоб популяризувати волонтерську діяльність в Україні та за її межами та полегшити пошук різноманітних громадських ініціатив та заходів. Платформа має мати можливість виконувати такі функції:

- надання можливості збереження інформації про користувачів, організації та події у базі даних;
- можливість модифікувати та видаляти інформацію в системі відповідно до потреб користувачів;
- надання можливості отримування повідомлень про створення нових подій у Telegram;
- можливість зміни пароль користувача з надсиланням тимчасового коду на мобільний номер;
- забезпечення функціонала для пошуку подій та ініціатив за різними критеріями, такими як місце проведення, дата, теги заходу та організаціями;

Щоб забезпечити реалізацію вищезгаданих функцій, програмне забезпечення має підтримувати виконання наступних операцій:

- інтеграцію з Telegram API для функціонування боту;
- зберігання даних у системі бази даних з реляційною структурою;
- інтеграцію з сервісом для відправки повідомлень Twilio;
- підтримка збереження інформації в локальному сховищі браузера для авторизації;

2.2. Опис застосованих математичних методів

Під час розробки інформаційної системи для реалізації безпечного збереження паролів користувачів до бази даних було використано алгоритм хешування HMACSHA256.

Аутентифікація HMAC-SHA256 використовується для підтвердження автентичності та цілісності повідомлень шляхом використання секретного ключа разом із хеш-функцією. HMAC означає код аутентифікації повідомлень на основі хеш-функції, тоді як SHA256 є популярною хеш-функцією, яка створює результат довжиною 256 бітів. Математичні операції, які застосовуються в SHA-256, вимагають значних обчислювальних ресурсів і складності. Це робить процес зворотного обчислення хеш-функції та визначення початкових вхідних даних дуже складними завданнями.

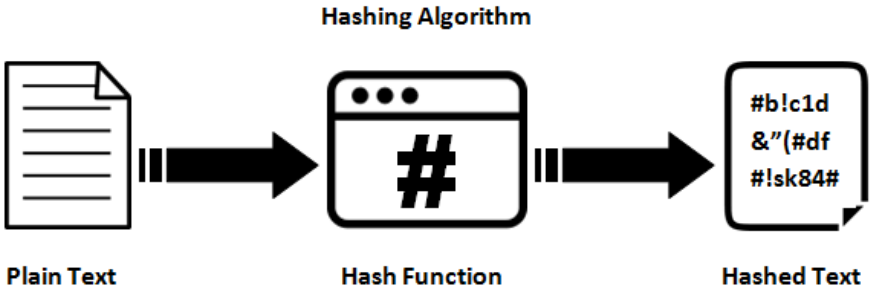

Рис. 2.1. Генерація хешу

Використання HMAC дозволяє обидвом сторонам, клієнту і серверу, спільно використовувати приватний ключ, який є відомим лише їм. Клієнт створює унікальний хеш (HMAC) для кожного запиту. Сервер, отримавши запит, хешує запитувані дані за допомогою приватного ключа клієнта і включає результат у відповідь. Хешування повідомлення і ключа відбувається на окремих етапах, що забезпечує безпеку процесу. Після отримання запиту сервер також генерує свій власний HMAC. Обидва HMAC порівнюються, і якщо вони збігаються, клієнт вважається автентичним.[3]

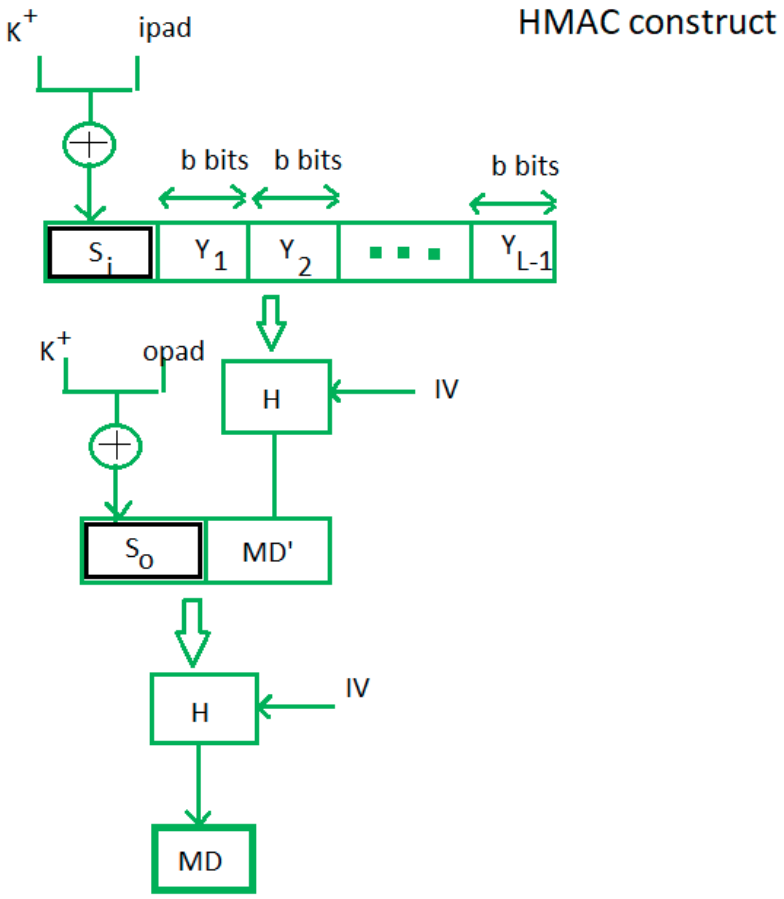

Рис. 2.2. Структура HMAC[4]

де H – позначає функцію хешування,

M – вихідне повідомлення,

Si та So – вхідні та вихідні сигнатури відповідно,

Yi – i-й блок у вихідному повідомленні M, де I лежить у діапазоні [1, L)

L – кількість блоків у M

K – секретний ключ, який використовується для хешування

IV – початковий вектор (деяка константа)

Таким чином алгоритм хешування HMAC-SHA256 надає можливість безпечно зберігати паролі користувачів у захешованому вигляді без можливості їх зворотного перетворення у початковий стан. Це дозволяє сказати, що навіть якщо дані бази даних потраплять до рук шахраїв, то це не дасть їм можливості використовувати паролі користувачів у своїх цілях.

2.3. Опис використаної архітектури та шаблонів проектування

Для розробки інформаційної платформи було використано архітектурний шаблон – Clean Architecture. Використання шаблонів в архітектурі коду дозволяє створити структуру коду, яка є чіткою і зрозумілою, спрощуючи підтримку і розширення системи. Це не аби як важливо для роботи над одним проєктом в команді, адже встановлює загальні правила для усіх розробників, що допомагає уникнути непорозумінь та підтримувати чистоту коду.

Основним правилом Clean Architecture є правило залежності. Залежності вихідного коду повинні вказувати тільки всередину, до політик вищого рівня. Жоден компонент внутрішнього кола не повинен мати інформації про будь-які деталі зовнішнього кола, включаючи назви функцій, класів, змінних або інших іменованих об'єктів програмного забезпечення, що оголошуються зовні. Формати даних, що використовуються у зовнішньому колі, не повинні бути прямо використані в внутрішньому колі, особливо якщо ці формати автоматично генеруються фреймворком у зовнішньому колі. Важливо забезпечити, щоб зміни у зовнішньому колі не мали впливу на внутрішні кола системи [5].

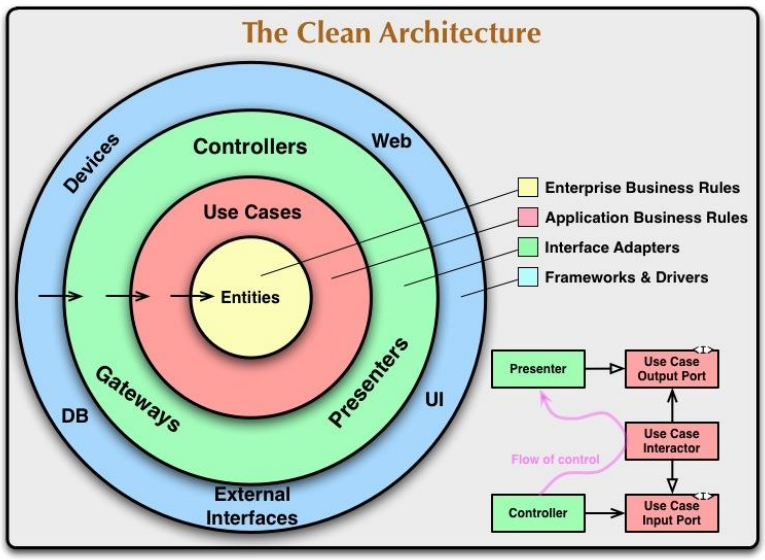

Рис. 2.3. Структура Clean Architecture

Також в ході розробки було використано поведінковий патерн проєктування Mediator. Використання даного патерну допомагає в зменшенні хаотичних зв’язків між об’єктами. Патерн обмежує прямі зв'язки між об'єктами та змушує їх співпрацювати лише через об'єкт-медіатор [6].

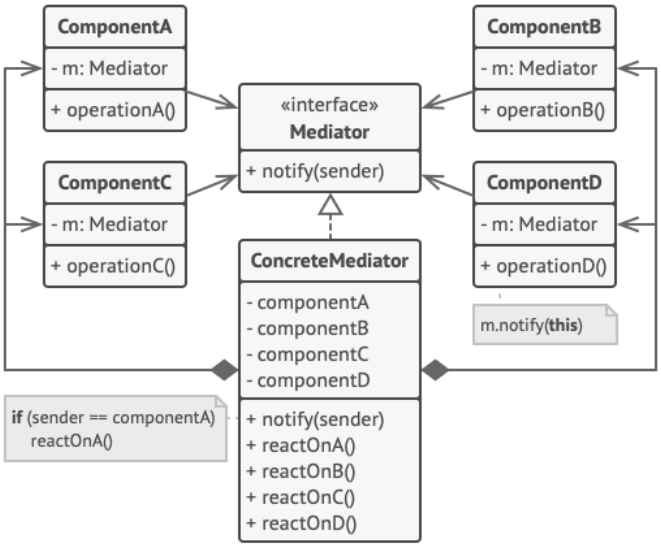

Рис. 2.4. Структура Mediator

Існує також інший підхід – сервіси, відомий як горизонтальна архітектура. Сервіси забезпечують зручну навігацію між методами та їх реалізацією, чого не можна сказати про підхід Mediator. Проте з погляду дотримання принципів SOLID, патерн Mediator є більш відповідним.

Одним з найголовніших принципів SOLID є принцип єдиної відповідальності. Який говорить про те, що кожен клас має відповідати лише за одну задачу. Це не означає, що класи повинні мати лише один метод або властивість. Вони можуть включати багато членів, якщо всі вони стосуються єдиної відповідальності. У випадку використання сервісів інколи буває складно дотримуватися такого принципу, адже з часом сервіс розширюється разом з його відповідальністю.

Існує ще один принцип SOLID який ускладнює роботу з сервісами – це принцип відкритості/закритості. Суть полягає в тому, що для спрощення внесення змін у програмні системи, вони повинні бути спроектовані так, щоб

дозволяти змінювати їх поведінку шляхом додавання нового коду, а не зміною існуючого [7]. Таким чином, щоб забезпечити виконання даного принципу сервісною архітектурою ми можемо використати наслідування, або патерн Декоратор. Це буде створювати додаткові інтерфейси та зв’язки між класами, яких ми маємо уникати. Що стосується обробників Mediator, оскільки всі вони реалізують однаковий інтерфейс, нам потрібно зосередитися лише на одному аспекті, не турбуючись про зміни в окремих обробниках.

Проаналізувавши два підходи до розробки, а саме використання сервісів і патерну Mediator, патерн Mediator був обраний як оптимальне рішення для розробки, враховуючи його переваги в спрощенні архітектури та підвищенні загальної якості програмного забезпечення.

Патерн CQRS теж було використано при розробці інформаційної системи «Шлях до перемоги». Основою задачею цього патерну є розділення операцій на дві групи: команди, що відповідають за модифікацію та змін стану програми та запити для отримання даних без зміни стану. Це робить додаток гнучкішим, дозволяє вносити зміни, не порушуючи роботу інших модулів, і усуває необхідність блокування таблиць або записів для оновлення, що може вплинути на продуктивність [8].

Патерн Репозиторій – механізм інкапсуляції поведінки зберігання, вилучення та пошуку, що імітує колекцію об'єктів [9]. Цей патерн також був використаний для розробки системи, адже дає можливість винести усю взаємодію з базою даних в окремі класи – репозиторії. Таким чином ми уникаємо ін'єкція контексту бази даних у сервіси чи обробники.

Репозиторій дозволяє нам додавати, оновлювати та видаляти об'єкти у колекції, а не турбуватися про те, як ці об'єкти потрапляють до бази даних та вилучаються з неї, або як надсилати та отримувати сутності з вебсервісу [10]. Завдяки цьому патерну, система стає більш гнучкою і готовою до змін, що є ключовим для довготривалої підтримки і розвитку програмного забезпечення.

2.4. Опис використаних технологій та мов програмування

Інформаційна система для волонтерських організацій була реалізована на мові програмування C# на базі ASP.NET Core з додатковим використанням фреймворку Vue.js для клієнтської частини застосунку та HTML з CSS. В якості СУБД використовувався MSSQL та Entity Framework для взаємодії з СУБД. Була виконана інтеграція з Telegram API для реалізації чат боту та Twilio для відправки повідомлень на мобільний телефон користувача.

Мова програмування C# – об'єктно-орієнтована та статично типізована мова програмування, що широко використовується в різних сферах від розробки комп'ютерних ігор, настільних додатків до вебзастосунків та серверів. Для розробки ігор мовою C# використовується багатоплатформний двигун Unity. Для настільних додатків існують такі графічні інтерфейси як Windows Forms або Windows Presentation Foundation. C# має високий рівень масштабування програм, що дозволяє створювати великі та довготривалі проєкти.

ASP.NET Core – це вебфреймворк збудований на основі платформи .Net. Це версія ASP.NET з відкритим вихідним кодом, яка працює на macOS, Linux та Windows [11].

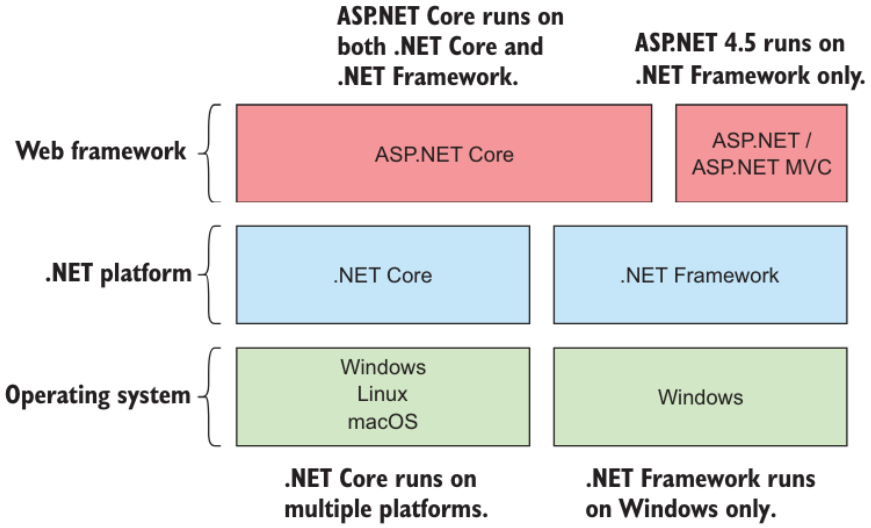

Рис. 2.5. Відносини між ASP.NET Core, ASP.NET, .NET Core і .NET Framework [12]

ASP.NET Core дотримується модульної архітектури, що додає фреймворку додаткову гнучкість. Це дозволяє будувати проєкт таким чином, щоб включати в нього тільки потрібні компоненти. Завдяки відкритому коду платформи, який опубліковано на GitHub можна більш детально поринути у принцип роботи та навіть зробити свій внесок у платформу.

MSSQL – Microsoft SQL Server програмне забезпечення для керування базами даних, що працює з реляційними базами даних. MSSQL заснована на мові структурованих запитів SQL. Тобто дані зберігаються у вигляді таблиці. Таблиці містять в собі колонки та рядки. Також є можливість створювати індекси для пришвидшення пошукових запитів до бази даних та ключів для забезпечення зав’язків між таблицями. MSSQL надає потужні інструменти для резервного копіювання та відновлення даних, що забезпечує надійність і безпеку зберігання інформації. Сервер підтримує транзакції, дозволяючи виконувати кілька операцій над даними як єдине ціле, що важливо для збереження цілісності бази даних.

Отже, MSSQL є всебічним рішенням для керування реляційними базами даних, яке гарантує високу продуктивність, надійність і безпеку, відповідаючи різноманітним потребам користувачів.

Entity Framework – це об'єктно-реляційний перетворювач. Entity Framework розвинулася з методології під назвою Entity Relationship Modeling (ERM). ERM визначає схему сутностей та їх взаємозв'язків. Сутності – це не те саме як об'єкти [13]. Існує два методи за якими можна працювати з Entity Framework code first та database first. Code first перетворює класи коду в об’єкти бази даних. Це дуже зручно для моделювання в процесі розробки, адже такий підхід забезпечує гнучке оновлення бази даних за допомогою міграцій. Міграції створюються автоматично за допомогою спеціальних команд, що прописуються у консолі. Також до готових міграцій можливо додавати частини SQL скриптів для більш специфічних задач. Database first використовується частіше за умови існування готової схеми даних перед початком розробки. Цей підхід дозволяє

розробникам працювати з встановленою і оптимізованою базою даних без необхідності створювати структуру з нуля.

Telegram API – дозволяє створювати свої власні клієнти Telegram. Цей API дозволяє створювати ботів. Telegram-боти – це спеціальні акаунти, для налаштування яких не потрібен додатковий номер телефону [14]. Такі боти мають великий перелік можливостей від простого спілкування, роботи з медіафайлами до виконання певних бізнес-задач. Telegram API підтримує RESTful архітектуру, що спрощує взаємодію з ним через HTTP запити. Також він підтримує асинхронні запити, що дозволяє ефективно обробляти багато запитів і подій одночасно.

Twilio – це хмарна комунікаційна платформа для обміну повідомленнями. Twilio API надає інтерфейси для інтеграції та виправляння текстових, голосових та відео повідомлень. Twilio API підтримує роботу з різними платформами, включаючи вебдодатки, мобільні додатки і системи IoT. Це дозволяє інтегрувати комунікаційні можливості Twilio в різноманітні середовища. Також надається широка документація з прикладами коду різними мовами програмування, що значно спрощує роботу розробникам.

Vue.js – це фреймворк, на основі мови програмування JS, що використовується для створення реактивних вебдодатків. Основною відмінністю Vue.js від інших фреймворків є те, що у віртуальному DOM дереві він використовує HTML шаблони. На цей час існує три версії даного фреймворку. Під час створення кваліфікаційної роботи використовувалася остання актуальна третя версія.

JS – це динамічна мова програмування, що була створена для безпосередньої роботи у середині браузера. JS базується на мові програмування ECMAScript. Під час виконання кваліфікаційної роботи, була використана остання актуальна версія JS. Варто зауважити, що за допомогою JS була реалізована взаємодія клієнтської та серверної сторін, також уся бізнес-логіка вебплатформи з клієнтської сторони була реалізована саме цією мовою.

HTML – це мова розмітки вебдокументів, яка використовується для структуризації елементів вебсторінок. Під час виконання кваліфікаційної роботи була використана п’ята версія HTML.

CSS – це мова опису стилю вебсторінок за допомогою HTML. У кваліфікаційній роботі використана актуальна третя версія CSS. Варто зауважити, що ні HTML, ні CSS не являються повноцінними мовами програмування, але грають визначну роль у створенні та стилізації вебсторінок та повноцінних сайтів. За допомогою CSS на сайті можуть використовуватися різні кольори, шрифти відступи та інша розмітка.

2.5. Опис структури програмного продукту та алгоритмів її функціонування

Відповідно до стандартів Clean Architecture програмний код було поділено на наступні модулі:

1. API Layer: це рівень, що відповідає за отримання запитів та відправки відповідей клієнту. Цей рівень не містить бізнес-логіки і спілкується з іншою частиною додатку через інші рівні. Тут знаходяться контроллери, що оброблюють вхідні HTTP-запити та повертають відповіді HTTP-відповідні [15]. Вони виконують дію посередника між клієнтом та серверною частиною. Також на цьому рівні містяться розширення, конфігурація застосунку та проміжне програмне забезпечення для коректного відображення виключень та помилок. Клас Program є вхідною точною в програму в якому налаштовуються усі сервіси та служби. Файл appsettings.json відповідає за конфігурацію, в ньому зберігаються необхідні значення для налаштування логів, рядок підключення до бази даних, конфігурація для відправки текстових повідомлень та секрет для хешування паролів.

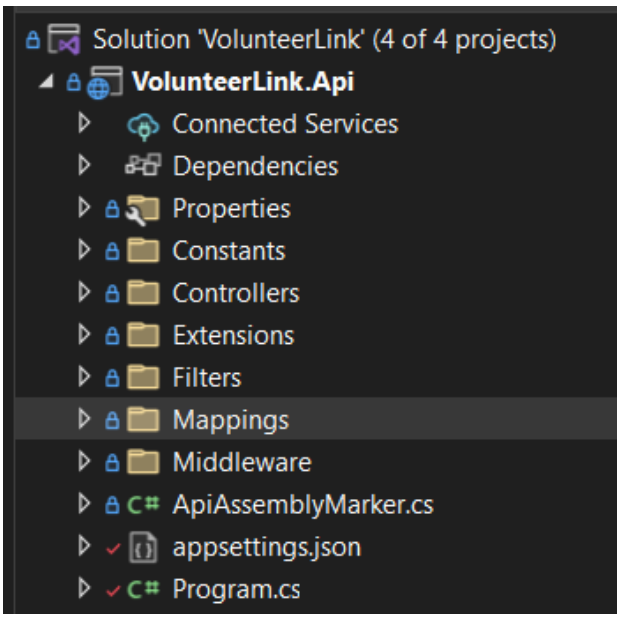

Рис. 2.6. Структура API рівня

2. Core Layer: це основний рівень архітектури який зберігає в собі бізнес-логіку. Цей рівень не має залежати від інших рівнів таких як API чи Infrastructure і взаємодіє з ними тільки за допомогою абстракцій. Він містить такі компоненти як функції, сервіси, доменні моделі та інтерфейси репозиторіїв. Папка Features містить в собі логіку для роботи з різними сутностями. Для кожної з сутностей існує окрема папка для поділу на команди чи це створення організації, редагування волонтера, отримання списку усіх заходів чи донаті тощо. У папці сервісів знаходяться сервіси для роботи з паролями, адаптери для відправки текстових повідомлень, а також сервіс Telegram боту разом з фоновим сервісом, що робить запит до бази даних кожні 5 хвилин та перевіряє наявність нових заходів, щоб відправляти про це повідомлення до користувачів у Telegram. Також на цьому рівні містяться DTO та правила для мапінгу сутностей з команд чи то запитів у DTO, чи доменні моделі. Інтерфейс ICoreAssemblyMarker в Core Layer використовується для полегшення налаштування залежностей та автоматичного виявлення збірки при конфігуруванні залежностей у додатку.

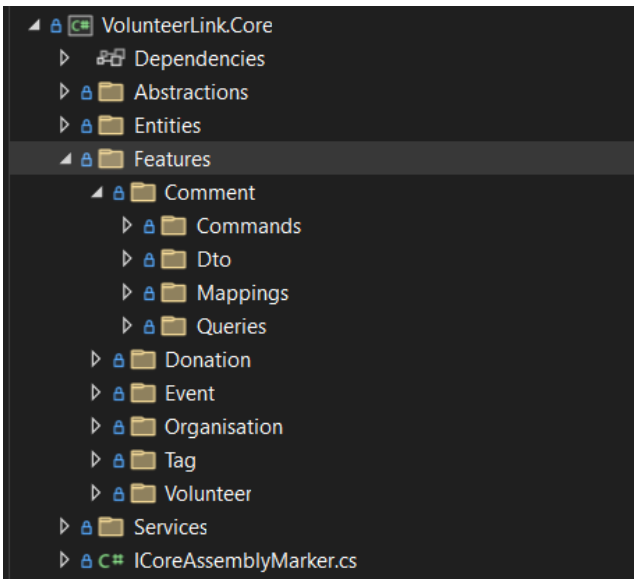

Рис. 2.7. Структура Core рівня

3. Infrastructure Layer: це рівень взаємодії з базою даних. На ньому зберігаються репозиторії, міграції та контекст бази даних. Context має у собі контекст класу Entity Framework і містить набір DbSet<>, що представляє колекції сутностей, які відображаються в таблиці бази даних.

4. Contracts Layer: це рівень, який визначає контракти між різними частинами системи. Контракти забезпечують чітку комунікацію і взаємодію між різними рівнями архітектури, такими як Core та API. Вони допомагають підтримувати слабке зв'язування і сприяють легкому тестуванню та модульності системи.

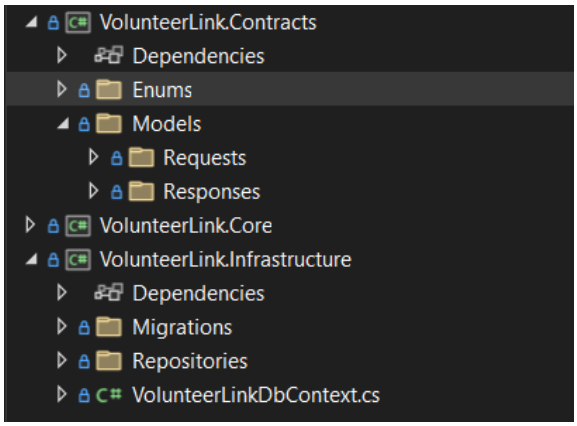

Рис. 2.8. Структура Infrastructure та Contracts рівня

Робота Telegram боту поділяється на дві частини. Перша частина це підписка користувача на бажані теги. Коли користувач в чаті пише команду /start то бот починає свою роботу і сповіщає сервер про те, що йому треба отримати список тегів з системи для відображення користувачу. Після чого бот формує повідомлення у вигляді меню з тегами. Коли користувач отримує це повідомлення, він може обрати за якими тегами він хотів би отримувати сповіщення про нові події. Після вибору тегів сервер зберігає дані про id чату та обрані теги в базу даних.

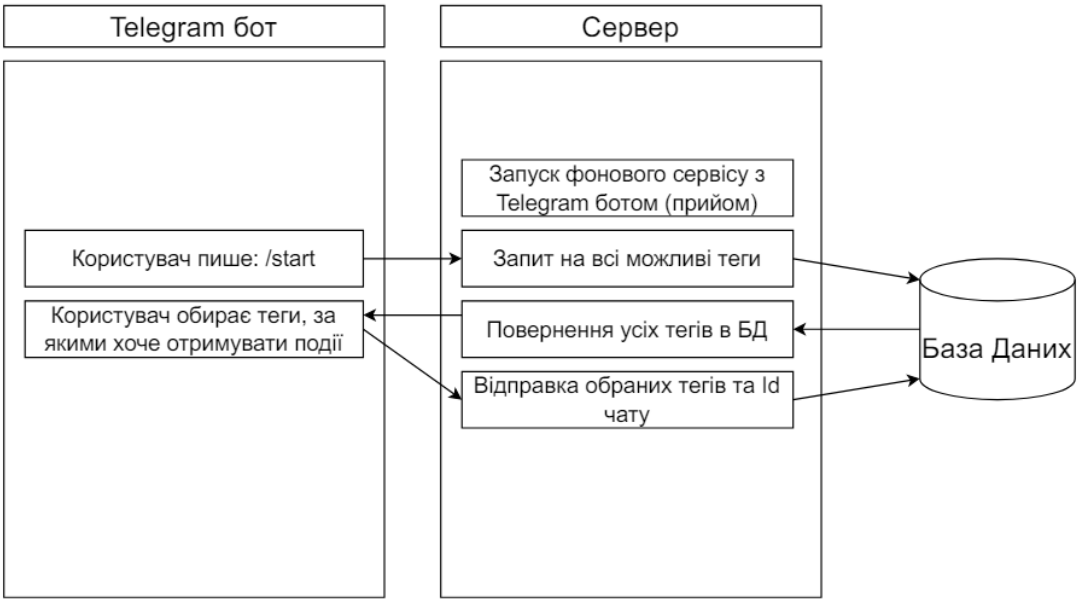

Рис. 2.9. Принцип роботи Telegram боту з обранням тегів

Друга частина являє собою фоновий сервіс, який працює кожні 5 хвилин на перевіряє базу даних на наявність новий подій з відповідними тегам. Коли нові події створюються – Telegram бот відправляє повідомлення користувачу про подію.

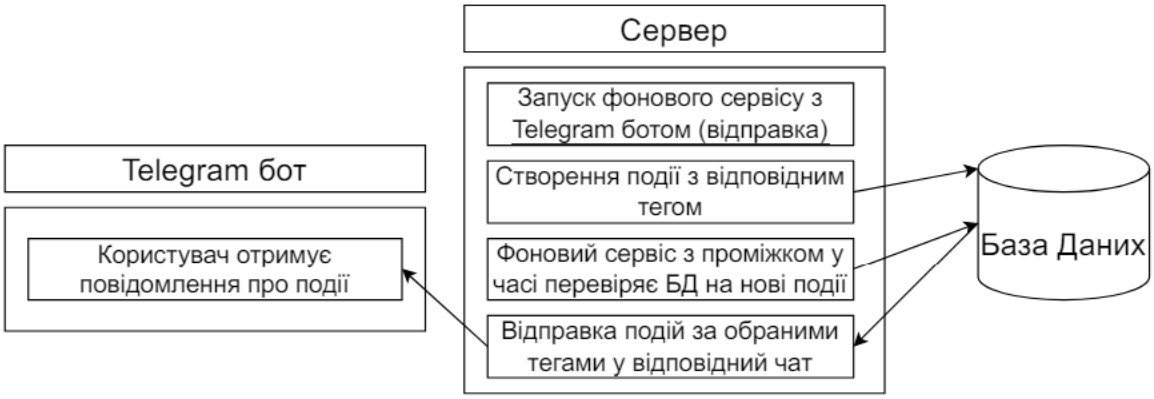

Рис. 2.10. Принцип роботи Telegram боту з відправкою повідомлень

Клієнтська частина інформаційної системи побудована на базі фреймворку Vue.js 3 та використовує безплатну бібліотеку стилів та макетів «Sakai». Архітектура клієнтської частини у своїй основі базується на менеджері пакетів npm, який відповідає за збереження та співпрацю різних пакетів та бібліотек у нашому застосунку.

У Vue.js кожна сторінка – це окремий компонент, який зберігається як окремий файл з розширенням vue. На рисунку показані всі компоненти, які використовуються в застосунку.

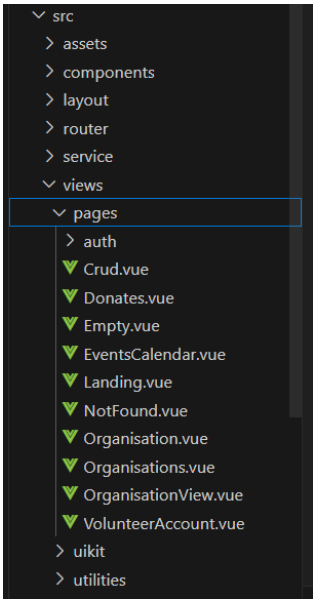

Рис. 2.9. Всі сторінки застосунку

Файл index.js, описує взаємозв’язок між компонентами та HTTP шляхом у веббраузері. Кожен файл, який описує вебсторінку, містить в собі три частини. Перша частина екранована тегом <script></script>, тут знаходиться JS код, функціональний код, який відповідає за бізнес логіку. У тезі <template></template> розташован HTML код, який відповідає за позиціювання елементів на сторінці, а в тезі <style></style> розташований CSS для стилізації.

Рис. 2.11. Архітектура файла, що описує вебсторінку

Важливим аспектом клієнтської роботи веб платформи є взаємодія с серверною частиною. Для того, щоб створити подібний зв’язок, була використана бібліотека axios, усі методи для безпосереднього з’єднання клієнтської частини та серверної розміщені у файлі Repository.js. Таким чином, безпосередньо axios не використовується у компонентах веб сторінок, всі сторінки використовують методи з репозиторію.

Для того, щоб зберігати JWT, який видає нам сервер буде використано VUEX та local storage веб браузера.

База даних містить в собі багато таблиць. Було застосовано два види зв’язків один до багатьох та багато до багатьох. Зв’язки один до багатьох широко застосовуються в проєктуванні баз даних. Наприклад, одна організація може створювати багато зборів, що є прикладом реалізації такого типу зв’язків.

Зв’язок багато до багатьох потребує створення додаткової таблиці, яка буде зберігати в собі Id. Прикладом реалізації такого типу зв’язків є таблиця EventTag. Багато тегів можуть бути пов’язані з багатьма заходами та навпаки.

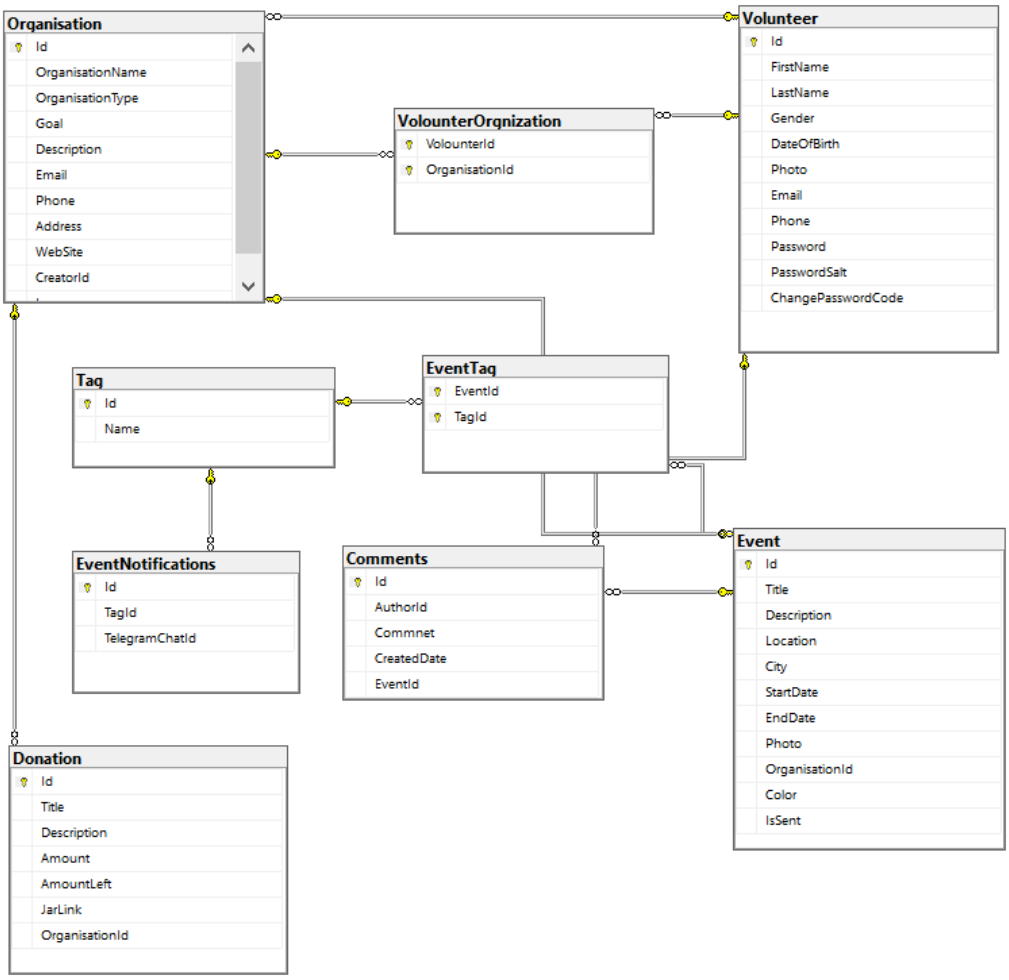

Рис. 2.11. UML-діаграма структури БД

Так як для розробки інформаційної системи було використано Entity Framework то в проекті присутні міграції. Міграції допомагають вносити зміни в базу даних, дозволяють автоматично відстежувати зміни в моделях даних і створювати скрипти для оновлення схеми бази даних відповідно до цих змін.

Це включає додавання нових таблиць, зміну існуючих, додавання або видалення стовпців, налаштування індексів і встановлення зв'язків між таблицями.

```
public partial class Addcomment : Migration
{
    /// <inheritdoc />
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Comments",
            columns: table => new
            {
                Id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                AuthorId = table.Column<int>(type: "int", nullable: false),
                Commnet = table.Column<string>(type: "nvarchar(max)", nullable: false),
                CreatedDate = table.Column<DateTime>(type: "datetime2", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Comments", x => x.Id);
                table.ForeignKey(
                    name: "FK_Comments_Volunteer_AuthorId",
                    column: x => x.AuthorId,
                    principalTable: "Volunteer",
                    principalColumn: "Id",
                    onDelete: ReferentialAction.Cascade);
            });

        migrationBuilder.CreateIndex(
            name: "IX_Comments_AuthorId",
            table: "Comments",
            column: "AuthorId");
    }
```

Рис. 2.12. Приклад міграції для давання сутності коментарів

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Інформаційна платформа отримує вхідні дані з клієнтської частини та зберігає їх до бази даних. Обґрунтування вхідних даних:

– користувач вводити свої дані для реєстрації або входу на платформі;

– користувач може редагувати свої дані, вводи та редагувати дані організацій, заходів, тегів та донатів.

Організація вхідних даних:

– вхідні дані отримуються через клієнтську частину та передаються у json форматі на сервер;

– вхідна інформація спрямовується до певних методів і функцій для її обробки, зберігання до бази даних та маніпуляцій.

Обґрунтування вихідних даних:

– система відображає дані користувачів, організацій, заходів, тегів та донатів;

– після маніпуляцій з даними вони автоматично оновлюються;

– при перегляді профіля користувача система відображає детальну інформацію;

– при перегляді профіля організації система відображає детальну інформацію та при перегляді заходу.

Організація вихідних даних:

– вихідні дані передаються з серверної частини на клієнтську частину у форматі json.

Таким чином забезпечується безпечний обмін інформацією між клієнтською та серверною частиною.

2.7. Опис роботи розробленого програмного продукту

2.7.1. Використані технічні засоби

Для роботи інформаційної системи для волонтерів «Шлях до перемоги», рекомендуються наступні технічних характеристики засобів:

- процесор класу AMD Ryzen™ 7 4800H Mobile Processor (8-core/16-thread, 12MB Cache, 4.2 GHz max boost);
- модулі пам’яті 16GB DDR4-3200 SO-DIMM x 2;
- SSD диск 1TB PCIe® 3.0 NVMe™ M.2;
- відеокарта NVIDIA® GeForce RTX™ 3050 Ti;
- 4 GB відеопам’яті;
- мережа Internet зі швидкістю не менше 10 мб/с;
- Windows 10;
- клавіатура;
- комп’ютерна миша.

Рекомендовані технічні характеристики, які були описані вище, визначають мінімальні вимоги до надійності, швидкості обробки даних і безпеки для забезпечення відповідності вимогам.

2.7.2. Використані програмні засоби

Під час розробки були використані наступні програмні засоби для сервісної частини з виростанням ASP.NET на мові програмування C#:

- Visual Studio 2022 Community;
- Microsoft SQL Server Management Studio 18.

Для розробки клієнтської частини з використанням Vue.js та бібліотека стилів Sakai:

- Visual Studio Code;
- веббраузер Google Chrome.

2.7.3. Виклик та завантаження програми

Локальний запуск системи складається з двох етапів. По-перше, треба запустити серверну частину. Оскільки сервісна частина побудована на .NET, то треба впевнитись, що він встановлений на вашій машині, для цього треба перейти на офіційний сайт Microsoft та завантажити .NET 7 [16]. Для зручного запуску проєкта можна скористатись IDE на ваш вибір, у якості прикладу буде використана Visual Studio 2022. Далі треба скопіювати з диску, який додається до кваліфікаційної роботи теку з серверною частиною, та знайти в ній файл з розширенням .sln. Натиснувши два рази на цей файл, відкриється Visual Studio з повністю підвантажений проєктом. Для того, щоб сервер працював коректно, варто створити базу даних та накотити збережені у проєкті міграції. У якості інструменту для взаємодії з базою даних рекомендується використовувати MS SQL [17]. У MS SQL варто створити базу з назвою «VolunteerLink». Далі у файлі appsettings замінити рядок з адресою підключення до бази даних. Після всього

цього, в менеджері Nuget пакетів, треба виконати команду «Update-Database» і дочекатися повідомлення про успішне виконання всіх міграцій. Тепер серверна частина працює і її можна запускати натиснувши на кнопку запуску.

Для того, щоб налаштувати другу частину системи, треба скопіювати теку з диска, яка містить в собі клієнтський код. Для того, щоб працювати з цим кодом було легше, пропонується завантажити на свою машину Visual Studio Code. Відкривши каталог з клієнтським рішенням у Visual Studio треба прописати в терміналі «npm install». Ця команда дозволить встановити всі залежності пакетів, які потрібні додатку для функціонування. Після чого ви можете прописати в терміналі ще одну команду «npm run dev», після чого відкриється клієнтська частина у браузері, який встановлений за замовчуванням. Для коректної роботи клієнтської частини варто, щоб вона працювала разом із серверною.

Варто зауважити, що для коректної роботи Telegram бота, та розсилки повідомлень на телефон, треба занести в файл appsettings відповідні ключі доступу. Для загального тестування системи цього можна не робити.

2.7.4. Опис інтерфейсу користувача

Платформа «Шлях до перемоги» має декілька веб сторінок, які описують основний функціонал доступний користувачам. Головною сторінкою за замовчуванням є сторінка, де відображаються усі організації, до яких може доєднатися користувач у разі, якщо він авторизований. На наданому рисунку, дві тестові організації на прикладі яких буде продемонстрована робота системи.

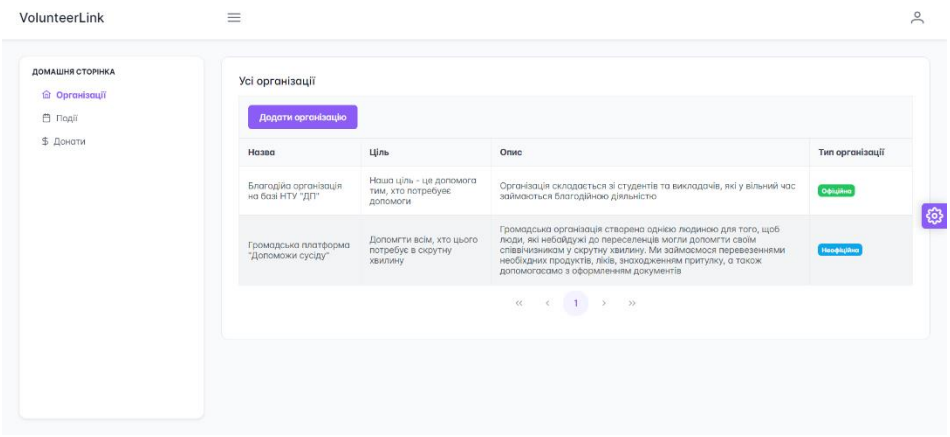

Рис. 2.13. Головна сторінка організацій

У майже будь-якому випадку, користувач завжди буде бачити верхнє меню та меню сторінок зліва. Верхнє меню містить:

– назву платформи;

– кнопку, яка ховає меню сторінок;

– іконку користувача для взаємодії з профілем користувача.

Якщо натиснути на іконку користувача, то можна буде побачити список можливих дій, який залежить від того авторизований користувач чи ні.

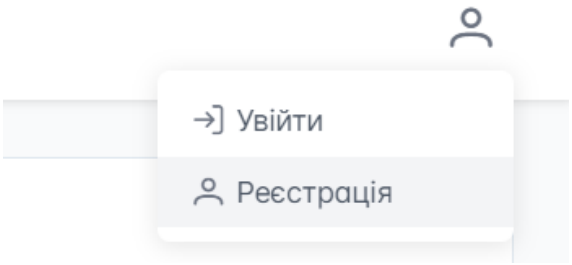

Рис. 2.14. Меню дій користувача

Якщо спробувати зареєструватися, натиснувши відповідну кнопку, то відкриється модальне вікно, де ми зможемо заповнити інформацію про себе, та стати повноцінним зареєстрованим користувачем платформи.

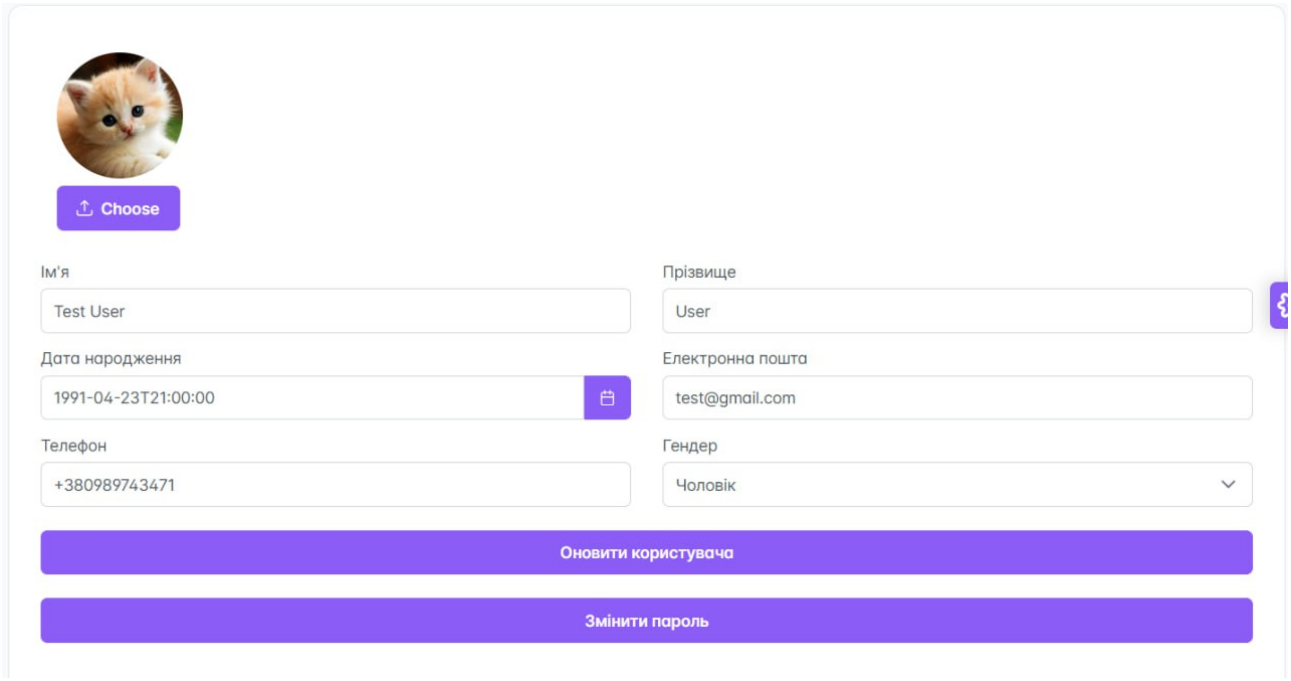

Рис. 2.15. Модальне вікно для реєстрації

Після натискання кнопки «Створити користувача» ми отримаємо повідомлення про те, що користувач успішно доданий до нашої системи.

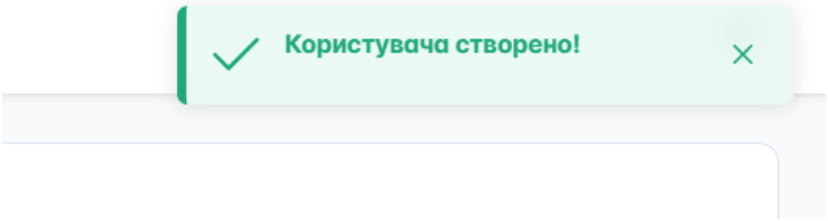

Рис. 2.16. Приклад повідомлення про успішне створення користувача

Після того, як ми створили нашого тестового користувача, ми маємо увійти в систему, використовуючи його дані, для цього треба знов натиснути на іконку користувача в верхньому меню і вже після цього натиснути на кнопку «Увійти». Після цього відкриється модальне вікно для підтвердження особистості, куди ми вводимо наші дані.

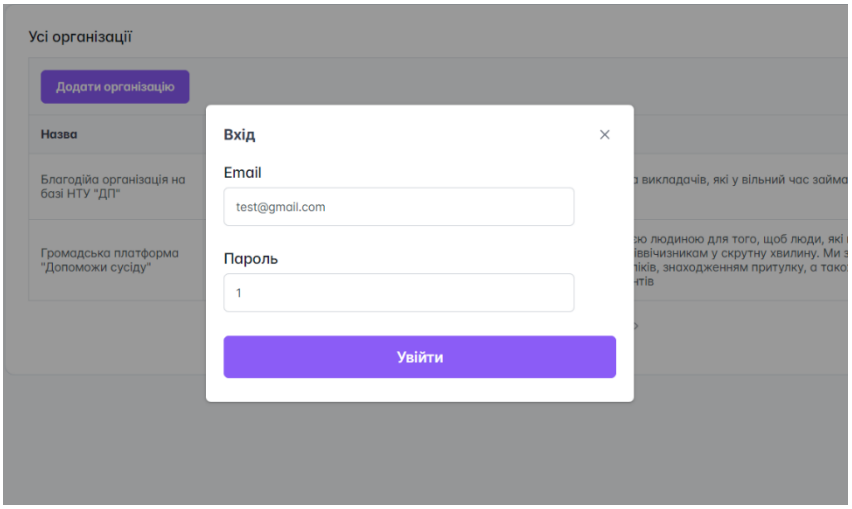

Рис. 2.17. Форма входу

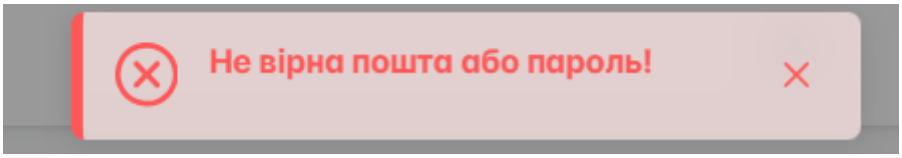

Рис. 2.18. Повідомлення буде показане користувачу, якщо він введе не вірні дані

У випадку, якщо користувач ввів свої дані правильно, то він отримає повідомлення, про успішний вхід. Тепер при натисканні на іконку користувача, список дій зміниться. Можна буде або вийти, або передивитися свій профіль.

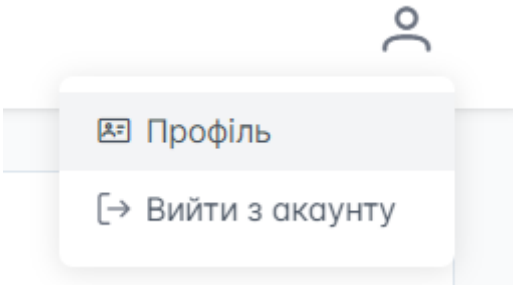

Рис. 2.19. Список можливих дій користувача, який ввійшов

Якщо натиснути на кнопку «Профіль», тоді ми перейдемо до редагування та перегляду власного акаунту, де можемо побачити основні дані, які ми заповнювали на минулому кроці.

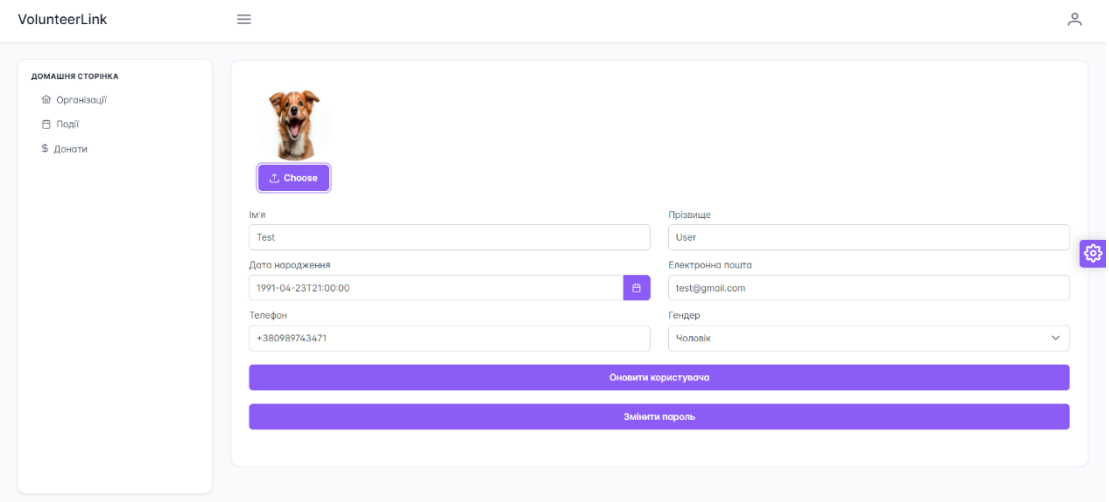

Рис. 2.20. Профіль користувача

Спробуємо змінити ім’я та фотографію профіля.

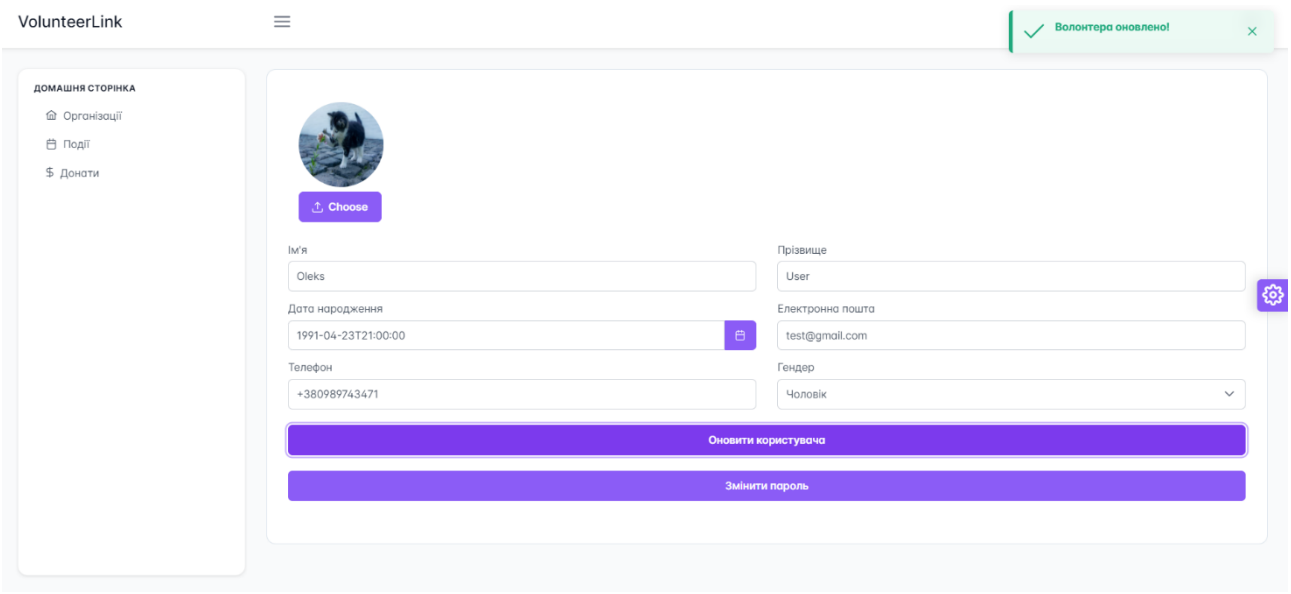

Рис. 2.21. Профіль користувача після змін та оновлення сторінки

На цій сторінці також можна змінити пароль. Щоб розпочати процес зміни пароля, треба натиснути на відповідну кнопку на рисунку, після чого відкриється спеціальне модальне вікно, де треба ввести код, який прийде на вказаний мобільний телефон користувача.

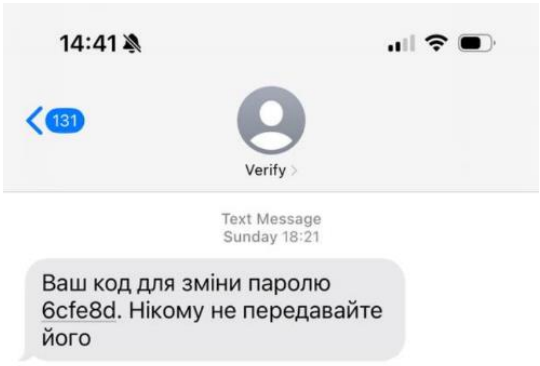

Рис. 2.22. Код для зміни пароля

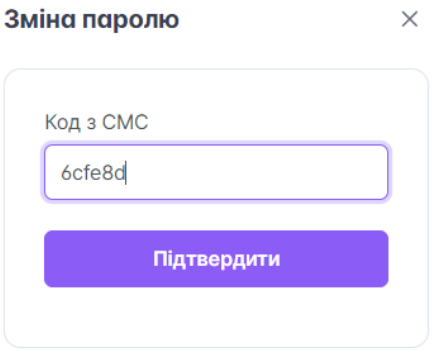

Рис. 2.23. Модальне вікно для введення коду

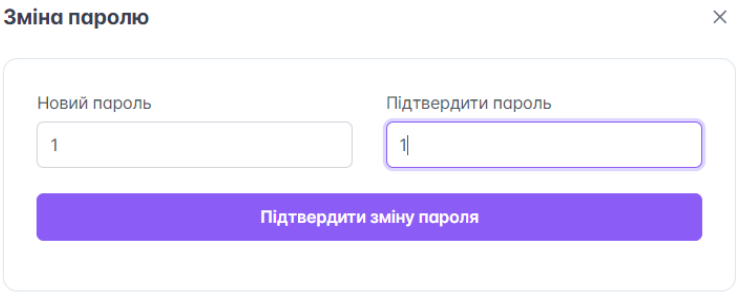

Рис. 2.24. Модальне вікно для зміни пароля

Повернувшись на сторінку організацій, та натиснувши кнопку «Додати організацію» відкриється модальне вікно для створення організації де, кожен бажаючий волонтер, може створити свою власну організацію та почати допомагати людям, для цього треба заповнити основну інформацію про організацію, та натиснути кнопку «Створити організацію.

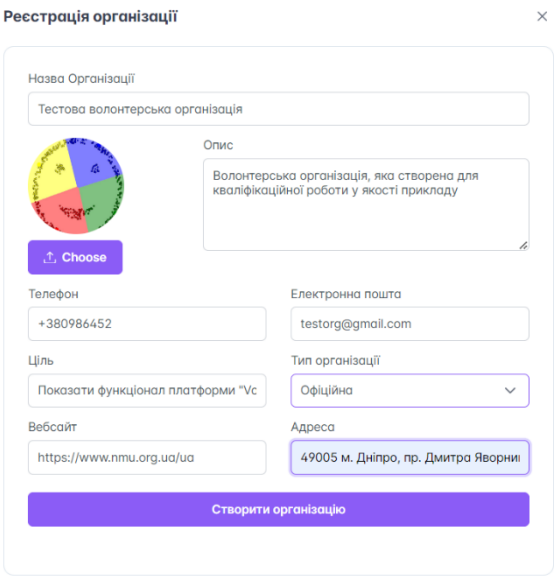

Рис. 2.25. Модальне вікно для реєстрації організації

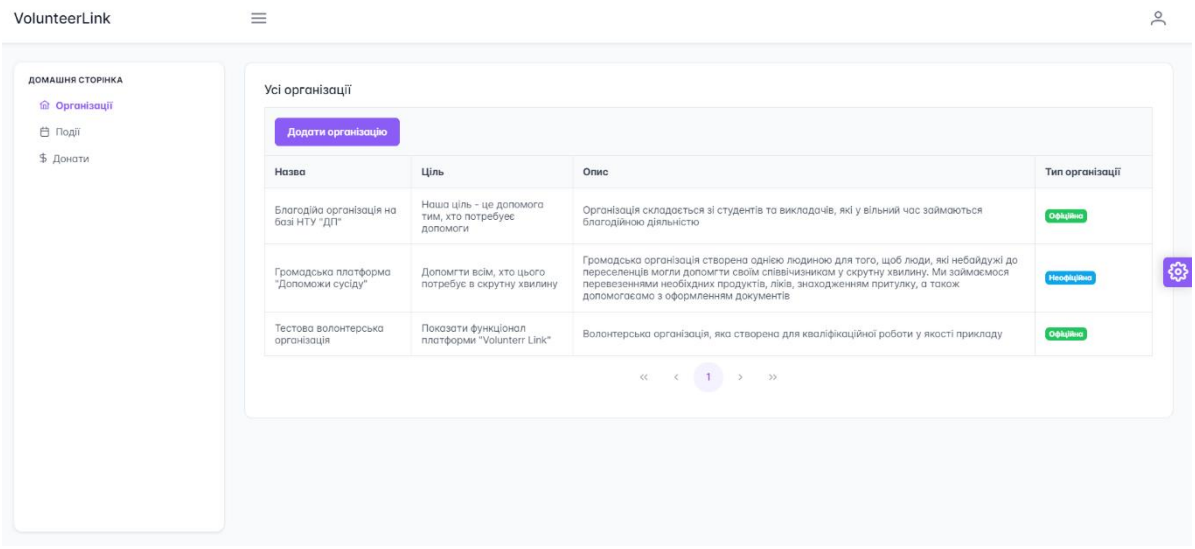

Рис. 2.26. Можемо побачити відповідні зміни в таблиці усіх організацій

Тепер користувач має змогу перейти на сторінку «Події» та передивитися в календарі список усіх подій, які вже були створені іншими організаціями.

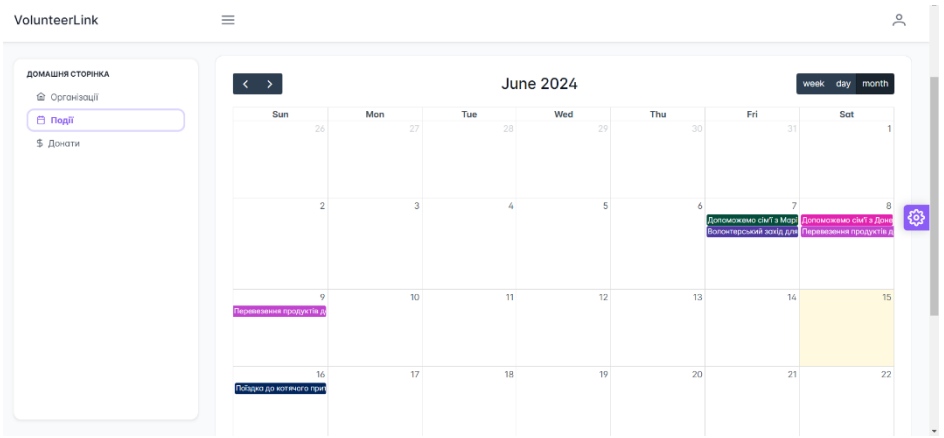

Рис. 2.27. Календар з усіма подіями, всіх організацій на нашій платформі

Зверху над календар є система фільтрів, де користувач може встановити ті параметри фільтрації для подій, які цікавлять саме його. Наприклад на рисунку встановлені фільтри для відображення подій, які належать нашій організації, також події мають бути відмічені тегами «Волонтерство» та «Будь-яка допомога», а також відбуватися вони повинні вже в наступному місяці липні у місті Кривий Ріг. Зараз таких подій не існує.

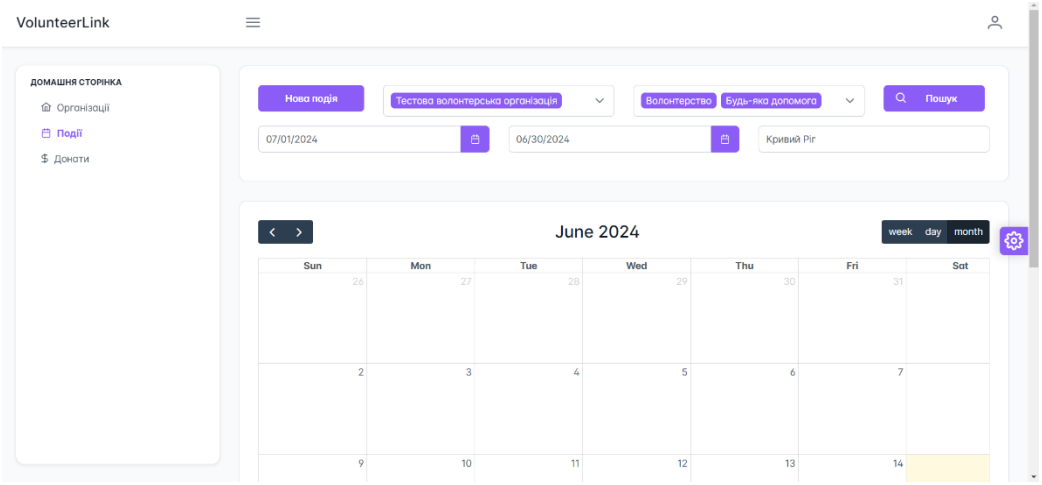

Рис. 2.28. Календар з фільтрацією

Створимо подію, яка буде відповідати вимогам фільтрації, та перевіримо результат. Для створення події треба натиснути на кнопку «Нова подія», яка відкриє модальне вікно для створення. В модальному вікні ми можемо обрати колір відображення події; організацію, до якої відноситься подія; теги за якими можна бути знайти цю подію, або отримати сповіщення про її створення у телеграм боті.

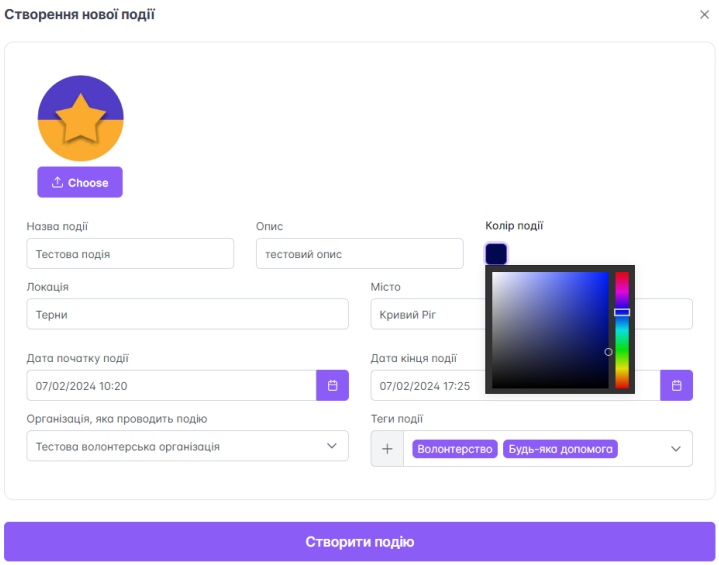

Рис. 2.29. Створення тестової події, що відповідає вимогам фільтрації

Якщо користувач бажає додати новий тег, якого ще не існує для його події, то він може натиснути на іконку плюса, після чого відкриється модальне вікно для додавання нового тегу.

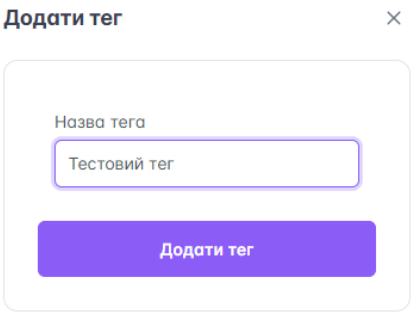

Рис. 2.30. Створення нового тега

Натиснувши кнопку «Створити подію», користувач збереже подію в нашій системі, та зможе знайти її в календарі, перемкнувшись на місяць липень та натиснувши на кнопку «Пошук».

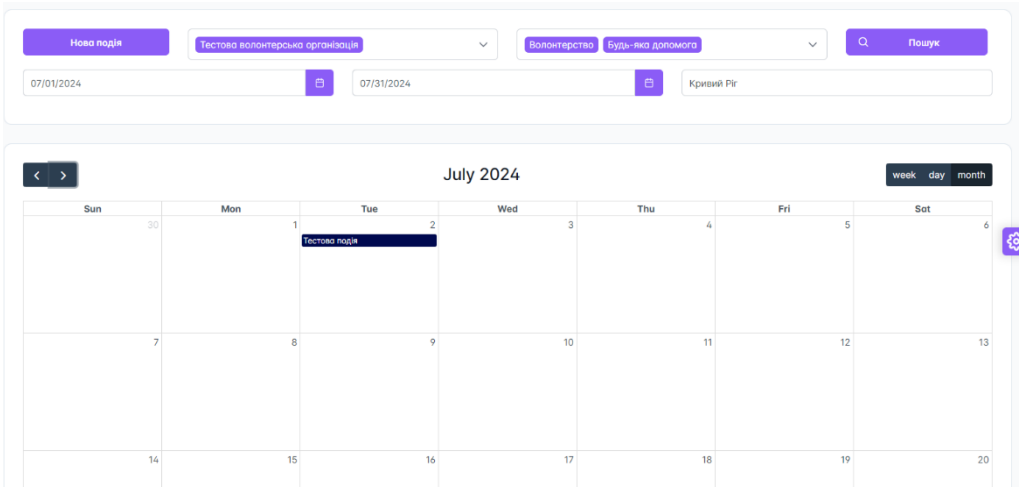

Рис. 2.31. Створена тестова подія у календарі, що відповідає фільтрам

Кожен користувач має змогу переглянути подібні події, та залишити коментар, за бажанням. При натисканні на подію в календарі, відкривається модальне вікно, де можна побачити головну інформацію про подію, а саме її назву, час протягом якого вона буде проходити, опис, коментарі, картинку та теги. Щоб додати новий тег, варто натиснути на кнопку «Додати коментар» та у відкритому модальному вікні написати бажаний коментар.

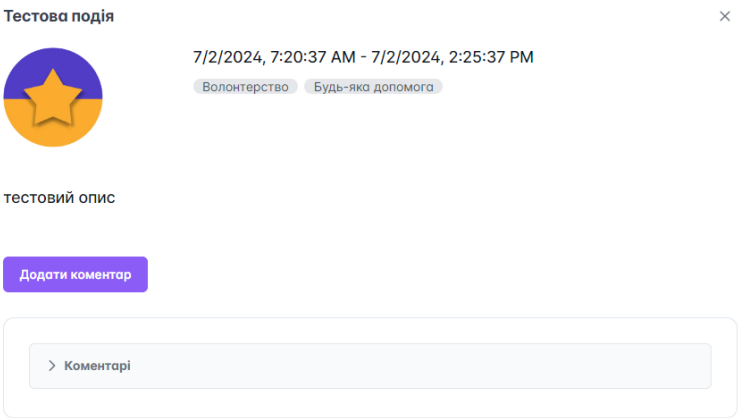

Рис. 2.32. Модальне вікно перегляду події

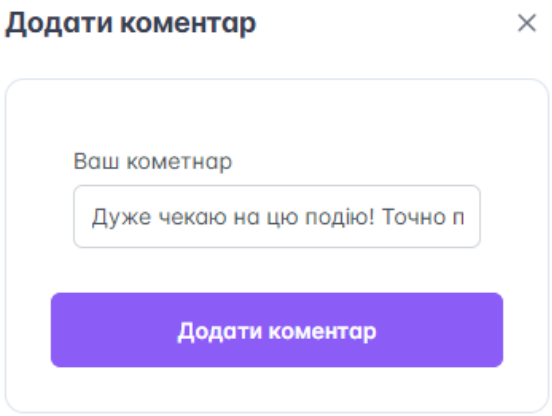

Рис. 2.33. Модальне вікно для додавання коментарів

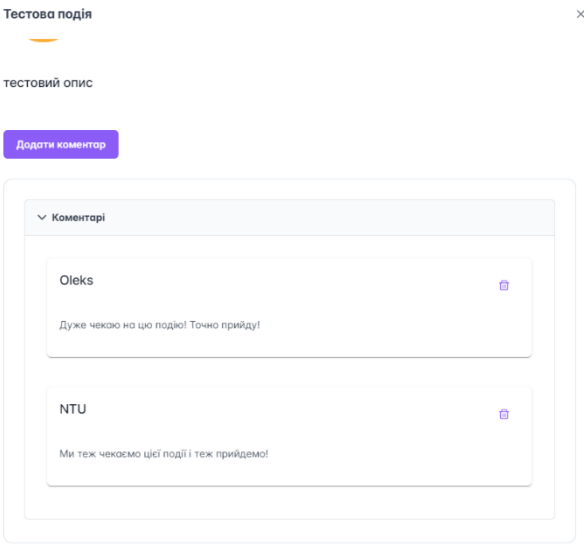

Рис. 2.34. Вигляд коментарів у перегляді події

Останньою сторінкою, яку може відвідати користувач є сторінка донатів. На сторінці донатів можна знайти пост, з такою інформацією про збір як ціль збору; організацію, яка його проводить збір; опис збору; яку суму треба зібрати і скільки вже зібрано. Також користувач може скористатися полем пошуку та знайти збір, який може його зацікавити.

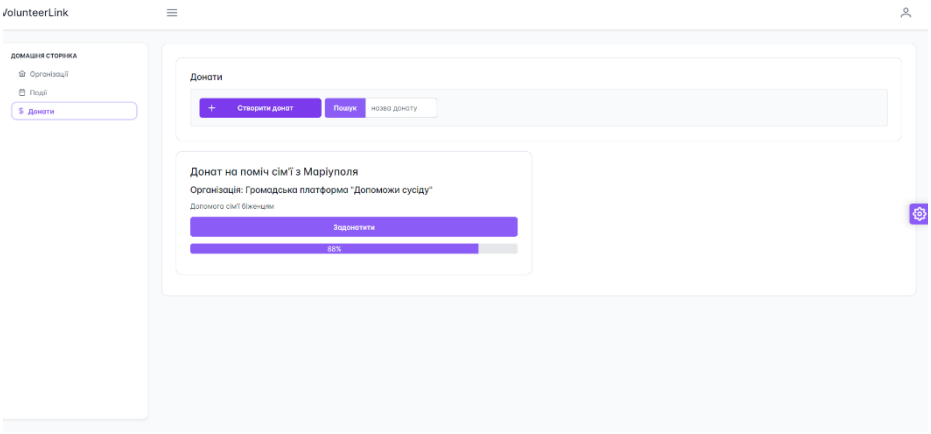

Рис. 2.35. Вигляд сторінки донатів

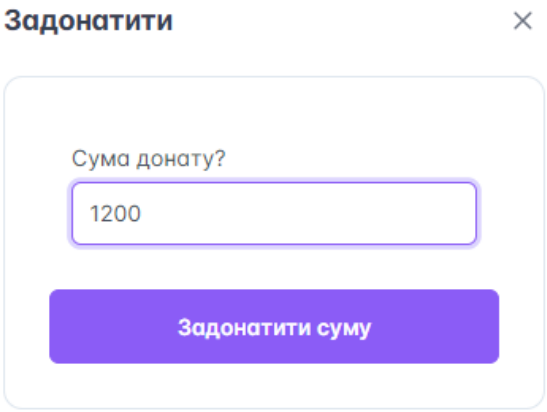

Рис. 2.36. Модальне вікно донату

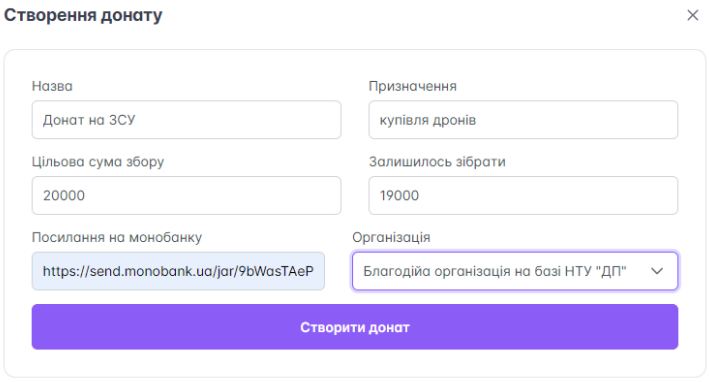

Рис. 2.37. Модальне вікно створення донату

Опис функцій Telegram боту. Перейшовши в бот перший раз користувач отримує загальне повідомлення та єдину кнопку – Start.

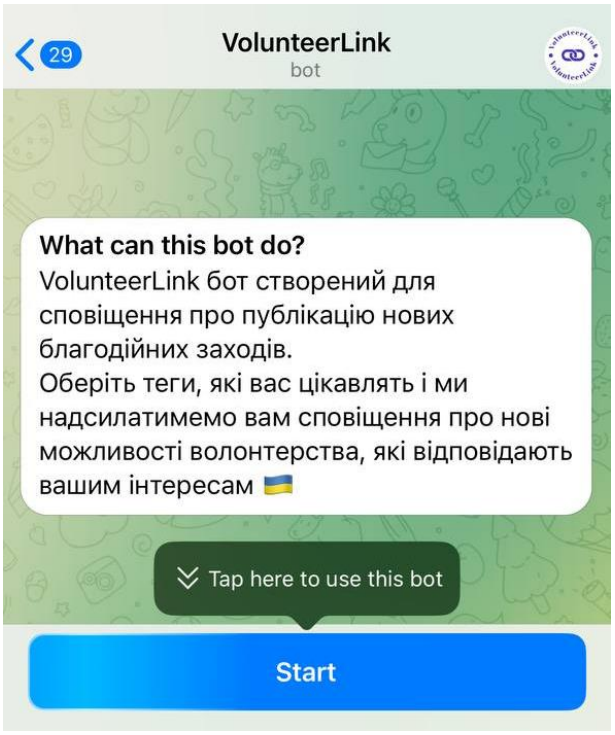

Рис. 2.38. Старт роботи з ботом

Після початку роботи в користувача з’являється можливість обрати ті теги, за якими він хоче отримувати сповіщення про нові заходи, що відбудуться.

Рис. 2.39. Вибір тегів

Після цього все готово і треба просто чекати на повідомлення про нові події. У разі появи таких, користувач буде отримувати повідомлення як на рис. 2.40.

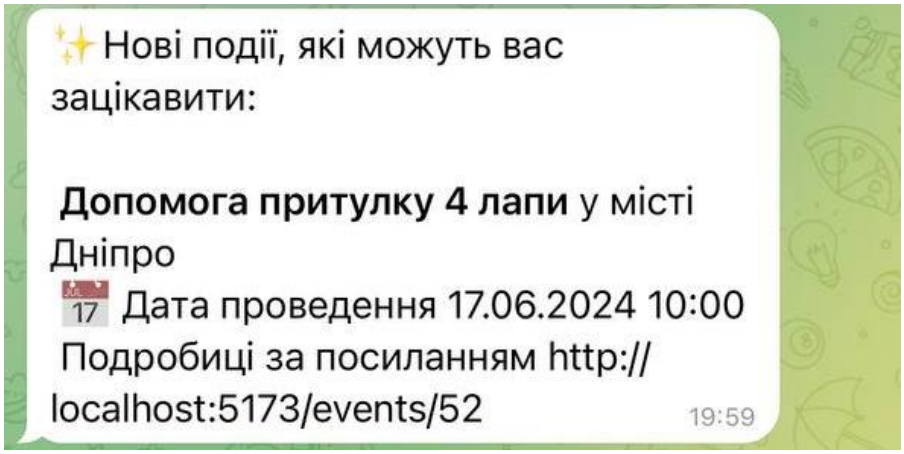

Рис. 2.40. Повідомлення про нові події

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Визначення трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми – 2150;
2. коефіцієнт складності програми – 1,4;
3. коефіцієнт корекції програми в ході її розробки – 0,2;
4. годинна заробітна плата програміста – 425 грн/год;

За останніми даними взятими з популярної української платформи «Work.ua» https://www.work.ua/salary-.Net-програміст/, середня заробітна плата .NET розробника становить 70 тисяч гривень. У програміста зазвичай робочий графік, що складається з 21 робочого дня по 8 годин на день. Отже, погодинна ставка становитиме (70 тисяч гривень / 8 годин) / 21 робочий день = 425 гривень на годину.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,1;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,4;
7. вартість машино-години ЕОМ – 0,57 грн/год.

У ході написання кваліфікаційної роботи єдиними витратами, які вплинули на підрахунок є витрати на інтернет з’єднання та витрати на електроенергію під час роботи за ноутбуком. Згідно за офіційними даними України на момент написання роботи, вартість 1 кВт/год становила 2,64 гривень [https://yasno.com.ua/b2c-tariffs]. Домашній інтернет коштував 295 гривень (за стабільне з’єднання 100 Мбіт/сек) на місяць [http://www.multitest.ua/internet-providers/fregat/about/]. Таким чином вартість інтернету в годину 295 / (30 * 24) = 0,41 грн/год. Ноутбук споживає 60 Вт, тож вартість електроенергії за годину використання становила 0,06 * 2,64 = 0,16 гривень. Таким чином, вартість машино-години комп'ютера становила 0,16 + 0,41 = 0, 57 гривень на годину.

Трудомісткість розроблення ПЗ може бути розрахована з використанням системи моделей, що пропонують різні рівні точності оцінки. Трудомісткість розроблення програмного забезпечення може бути визначена за такою формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин, } (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (2850);

c - коефіцієнт складності програми (1,4);

p - коефіцієнт корекції програми в ході її розробки (0,2).

Після підставлення значень умовне число операторів дорівнює:

$$Q = 2150 \cdot 1{,}4 \cdot (1 + 0{,}2) = 3612$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин,}$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 3 до 5 років він складає 1,2.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% (В = 1.2). Після підставлення значень маємо:

$$t_u = (3612 \cdot 1{,}2) / (75 \cdot 1{,}2) = 48{,}16 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20..25) \cdot k}, \text{ людино-годин, } (3.2)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.2), отримаємо:

$$t_a = 3612 / (20 \cdot 1{,}2) = 150{,}5 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25) \cdot k}, \text{ людино-годин.}$$

$$t_n = 3612 / (25 \cdot 1{,}2) = 120{,}4 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл}=\frac{Q}{(4..5)\cdot k}, \text{ людино-годин.}$$

Підставивши значення:

$$t_{отл} = 3612/ (5 \cdot 1,2) = 602 \text{ людино-годин.}$$

- за умови комплексного налагодження завдання:

$$t^{k}_{отл} = 1,5 \cdot t_{отл}, \text{ людино-годин.}$$

$$t^{k}_{отл} = 1,5 \cdot 602 = 903 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{д} = t_{дp} + t_{до}, \text{ людино-годин,}$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{др}=\frac{Q}{(15\text{-}20)\cdot k}, \text{ людино-годин.}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до}=0,75 \cdot t_{др}, \text{ людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$t_{др}$ = 3612/ (19 · 1,2) = 158,4 людино-годин.

$t_{до}$ = 0,75 · 158,4= 118,8 людино-годин.

$t_{д}$ = 158,4 + 118,8 = 277,2 людино-годин.

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t = 50 + 48,16 + 150,5 + 120,4 + 602 + 277,2 = 1248,26 людино-годин.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,}$$

де: t - загальна трудомісткість, людино-годин;

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 425 грн / год, отримуємо:

$З_{ЗП}$ = 1248,26 · 425 = 530 510,5 грн.

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot C_{мч}, \text{ грн}, \quad (3.3)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (0,57 грн/год).

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 602 \cdot 0{,}57 = 343{,}12 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 530\ 510{,}5 + 343{,}12 = 530\ 853{,}62 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс,}$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин);

t – загальна трудомісткість, людино годин()

Звідси витрати на створення програмного продукту:

$$T = 1248{,}26 / 1 \cdot 176 \approx 7{,}1 \text{ міс.}$$

Висновок: остаточні витрати на створення інформаційної системи для волонтерських організацій «Шлях до перемоги» становлять 530 853,62 гривень. Розрахунки показують, що термін створення програмного продукту буде становити 7,1 місяців. У цей період включено час на виправлення помилок та уточнення завдань у зв'язку з їхньою неточністю.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено інформаційну систему для волонтерської діяльності, що складається з серверної та клієнтської частини. Головним функціоналом створеної системи є створення та збереження, користувачів, організацій, подій та тегів, за якими ці події можна знайти. Серверна частина побудована за допомогою ASP.NET Core, а клієнтська частина базується на Vue.js. Важливо відмітити, що платформа інтегрована з Telegram, та має власного бота, який вміє сповіщати користувачів про нові події.

Під час виконання кваліфікаційної роботи, було дотримано усіх вимог, що були зазначені в розділі постановки задачі. Завдяки обраним архітектурним рішенням, система повноцінно функціонує, там має зрозумілу структуру, яка дозволила б у майбутньому розширити функціонал та додати нові сутності з якими могли б працювати волонтери.

Для користувачів нашої платформи відкривається велика кількість можливостей для розвитку своєї волонтерської діяльності. Кожен охочий може безкоштовно залучити інших волонтерів до своїх ініціатив та ідей. Користувачі також мають змогу створювати події, на які можна запросити інших небайдужих волонтерів, які хочуть внести свій вклад у будь-яку спільну справу. Загалом, створена нами система може дозволити будь-кому доєднатися до спільної справи та допомогти зробити цей світ кращим.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Благодійний фонд Сергія Притули | URL: https://prytulafoundation.org/about#support-direction (дата звернення: 20.05.23)
2. Rechi Support | URL: https://rechisupport.pp.ua/ (дата звернення: 20.05.23)
3. What is HMAC | URL: https://www.geeksforgeeks.org/what-is-hmachash-based-message-authentication-code/ (дата звернення: 09.06.24)
4. HMAC Algorithm in Computer Network | URL: https://www.geeksforgeeks.org/hmac-algorithm-in-computer-network/ (дата звернення: 09.06.24)
5. Robert C. Martin: Clean Architecture: A Craftsman's Guide to Software Structure and Design 1st Edition 2017, 205 p.
6. Alexander Shvets: Dive Into DESIGN PATTERNS 2018, 305 p.
7. Robert C. Martin: Clean Architecture: A Craftsman's Guide to Software Structure and Design 1st Edition 2017, 79 p.
8. Mastering CQRS Design Pattern with MediatR in C# .NET | URL: https://dev.to/dianaiminza/mastering-cqrs-design-pattern-with-mediatr-in-c-net-khk (дата звернення: 11.06.24)
9. Jimmy Nilsson: Applying Domain-Driven Design And Patterns: With Examples in C# and .Net 1st Edition 2006, 108 p.
10. Julia Lerman: Programming Entity Framework 2009, 695 p.
11. What is ASP.NET Core? | URL: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core (дата звернення: 12.06.24)
12. Third Edition: ASP.NET Core in Action 3rd Edition 2023, 23 p.
13. Julia Lerman: Programming Entity Framework 2009, 2 p.
14. Telegram APIs | URL: https://core.telegram.org/ (дата звернення: 15.06.24)
15. Clean Architecture in ASP .NET Core Web API | URL: https://juldhais.net/clean-architecture-in-asp-net-core-web-api-4e5ef0b96f99 (дата

звернення: 16.06.24)

16. Visual Studio 2022 | URL: https://visualstudio.microsoft.com/vs/ (дата звернення: 16.06.24)

17. Install SQL Server 2022 | URL: https://www.microsoft.com/en-us/sql-server/sql-server-downloads (дата звернення: 16.06.24)

ДОДАТОК А

КОД програми

Program.cs

```
using VolunteerLink.Api.Extensions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using VolunteerLink.Core.Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Configure custom services.
builder.ConfigureDatabase();
builder.ConfigureApplication();

// Configure CORS.
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", builder =>
    {
        builder.AllowAnyOrigin()
               .AllowAnyMethod()
               .AllowAnyHeader();
    });
});

// Register the Telegram bot background service.
builder.Services.AddHostedService<TelegramBotHostedService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
app.UseSwagger();
app.UseSwaggerUI();

app.UseRouting();
app.UseHttpsRedirection();
app.UseCors("AllowAll");
app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();

app.Run();
```

WebApplicationExtensions.cs

```
using Microsoft.AspNetCore.Mvc.ApiExplorer;
using Swashbuckle.AspNetCore.SwaggerUI;

namespace VolunteerLink.Api.Extensions;
```

```
public static class WebApplicationExtensions
{
    public static IApplicationBuilder UseApplicationSwagger(this IApplicationBuilder
app, IWebHostEnvironment webHostEnvironment)
    {
        if (webHostEnvironment.EnvironmentName.Equals(Environments.Production,
StringComparison.OrdinalIgnoreCase))
        {
            return app;
        }

        app.UseSwagger();

        var apiVersionDescriptionProvider =

app.ApplicationServices.GetRequiredService<IApiVersionDescriptionProvider>();
        app.UseSwaggerUI(options =>
        {
            foreach (var description in
apiVersionDescriptionProvider.ApiVersionDescriptions)
            {
                options.SwaggerEndpoint(
                    $"/swagger/{description.GroupName}/swagger.json",
                    description.GroupName.ToUpperInvariant());
            }

            options.DocExpansion(DocExpansion.None);
            options.OAuthClientId("swaggerui");
            options.OAuthAppName("Swagger UI");
        });

        return app;
    }
}
```

SwaggerExtensions.cs

```
using Microsoft.OpenApi.Models;

namespace VolunteerLink.Api.Extensions;

public static class SwaggerExtensions
{
    public static void ConfigureSwagger(this IServiceCollection services)
    {
        var info = new OpenApiInfo()
        {
            Title = "VolunteerLink API",
            Version = "v1"
        };

        var securityScheme = new OpenApiSecurityScheme()
        {
            Name = "Authorization",
            Type = SecuritySchemeType.ApiKey,
            Scheme = "Bearer",
            BearerFormat = "JWT",
            In = ParameterLocation.Header,
            Description = "JSON Web Token based security",
        };
```

```
        var securityReq = new OpenApiSecurityRequirement()
    {
       {
           new OpenApiSecurityScheme
           {
               Reference = new OpenApiReference
               {
                    Type = ReferenceType.SecurityScheme,
                    Id = "Bearer"
               }
           },
           new string[] {}
       }
    };

        services.AddSwaggerGen(o =>
        {
            o.SwaggerDoc("v1", info);
            o.AddSecurityDefinition("Bearer", securityScheme);
            o.AddSecurityRequirement(securityReq);
        });
    }
}
```

ServiceCollectionExtensions.cs

```
using VolunteerLink.Core.Abstractions;
using VolunteerLink.Infrastructure;
using Microsoft.EntityFrameworkCore;
using VolunteerLink.Core.Abstractions.Repositories;
using VolunteerLink.Infrastructure.Repositories;
using VolunteerLink.Core;
using Microsoft.AspNetCore.Mvc;
using Serilog.Filters;
using Serilog;
using Serilog.Sinks.SystemConsole.Themes;
using JWT.Extensions.AspNetCore;

namespace VolunteerLink.Api.Extensions
{
    public static class ServiceCollectionExtensions
    {
        public static WebApplicationBuilder ConfigureApplication(this
WebApplicationBuilder builder)
        {

            builder.AddRepositories();
            builder.AddAutomapper();
            builder.AddVersioning();
            builder.AddSerilog();
            builder.Services.AddServices();
            builder.Services.ConfigureSwagger();
            builder.Services.AddApplicationMediatR();
            builder.Services.AddJwt(builder.Configuration);

            return builder;
        }
```

```
        public static void AddJwt(this IServiceCollection services, IConfiguration
configuration)
        {
            services.AddAuthorization();
            services.AddAuthentication(options =>
            {
                options.DefaultAuthenticateScheme =
JwtAuthenticationDefaults.AuthenticationScheme;
                options.DefaultChallengeScheme =
JwtAuthenticationDefaults.AuthenticationScheme;
            })
                .AddJwt(options =>
                {
                    options.Keys = new[] { configuration.GetSecret() };
                    options.VerifySignature = false;
                });
        }

        private static IServiceCollection AddApplicationMediatR(this IServiceCollection
serviceCollection)
        {
            serviceCollection.AddMediatR(cfg =>
            {
                cfg.RegisterServicesFromAssemblyContaining(typeof(ApiAssemblyMarker));

cfg.RegisterServicesFromAssemblyContaining(typeof(ICoreAssemblyMarker));
            });

            return serviceCollection;
        }

        public static WebApplicationBuilder AddAutomapper(this WebApplicationBuilder
builder)
        {
            builder.Services.AddAutoMapper(typeof(ApiAssemblyMarker),
typeof(ICoreAssemblyMarker));
            return builder;
        }

        public static WebApplicationBuilder ConfigureDatabase(this
WebApplicationBuilder builder)
        {
            builder.Services.AddDbContext<IDbContext,
VolunteerLinkDbContext>((serviceProvider, options) =>
            {
                var dbOptions =
options.UseSqlServer(builder.Configuration.GetConnectionString("DbConnectionString"));

                if (builder.Environment.IsDevelopment())
                {
                    dbOptions.LogTo(Console.WriteLine, LogLevel.Information);
                }
            });

            return builder;
        }

        private static void AddVersioning(this WebApplicationBuilder builder)
        {
            builder.Services.AddApiVersioning(setup =>
            {
```

```
                setup.DefaultApiVersion = new ApiVersion(1, 0);
                setup.AssumeDefaultVersionWhenUnspecified = true;
                setup.ReportApiVersions = true;
            });

            builder.Services.AddVersionedApiExplorer(setup =>
            {
                setup.GroupNameFormat = "'v'VVV";
                setup.SubstituteApiVersionInUrl = true;
            });
        }

        public static WebApplicationBuilder AddRepositories(this WebApplicationBuilder
builder)
        {
            builder.Services.AddScoped<IVolunteerRepository, VolunteerRepository>();
            builder.Services.AddScoped<IOrganisationRepository,
OrganisationRepository>();
            builder.Services.AddScoped<IEventRepository, EventRepository>();
            builder.Services.AddScoped<ITagRepository, TagRepository>();
            builder.Services.AddScoped<ICommentRepository, CommentRepository>();
            builder.Services.AddScoped<IDonationRepository, DonationRepository>();

            return builder;
        }

        public static WebApplicationBuilder AddSerilog(this WebApplicationBuilder
builder)
        {
            var logger = new LoggerConfiguration()
                .ReadFrom.Configuration(builder.Configuration)
                .Filter.ByExcluding(Matching.WithProperty<string>("RequestPath", t =>
t.Contains("/health/", StringComparison.OrdinalIgnoreCase)))
                .CreateLogger();

            builder.Logging.ClearProviders();
            builder.Logging.AddSerilog(logger);

            builder.Host.UseSerilog((ctx, services, lc) =>
            {
                lc.Enrich.FromLogContext();
                lc.Enrich.WithMachineName();
                lc.Enrich.WithThreadId();
                lc.Enrich.WithProperty("ApplicationName", "VolunteerLink");
                lc.WriteTo.Console(applyThemeToRedirectedOutput: true, theme:
AnsiConsoleTheme.Literate);
                lc.WriteTo.Debug();
            });

            return builder;
        }
    }
}
```

ErrorHandlerMiddlewareю.cs

```
using VolunteerLink.Api.Extensions;
using static Microsoft.AspNetCore.Http.StatusCodes;

namespace VolunteerLink.Api.Middleware;
```

```
public class ErrorHandlerMiddleware
{
    private readonly RequestDelegate _next;

    public ErrorHandlerMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        catch (InvalidOperationException ex)
        {
            await context.Response
                .WithStatusCode(Status422UnprocessableEntity)
                .WithJsonContent(ex.Message);
        }
        catch (ArgumentOutOfRangeException ex)
        {
            await context.Response
                .WithStatusCode(Status416RangeNotSatisfiable)
                .WithJsonContent(ex.Message);
        }
        catch (UnauthorizedAccessException ex)
        {
            await context.Response
                .WithStatusCode(Status401Unauthorized)
                .WithJsonContent(ex.Message);
        }
        catch (FormatException ex)
        {
            await context.Response
                .WithStatusCode(Status404NotFound)
                .WithJsonContent(ex.Message);
        }
        catch (Exception ex)
        {
            await context.Response
                .WithStatusCode(Status500InternalServerError)
                .WithJsonContent(ex.Message);
        }
    }
}
```

VolunteerController.cs

```
using AutoMapper;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using VolunteerLink.Contracts.Models.Requests.Volunteer;
using VolunteerLink.Core.Features.Volunteer.Commands.ChangePassword;
using VolunteerLink.Core.Features.Volunteer.Commands.CreateVolunteer;
using VolunteerLink.Core.Features.Volunteer.Commands.UpdateVolunteer;
using VolunteerLink.Core.Features.Volunteer.Queries.GetAllVolunteers;
using VolunteerLink.Core.Features.Volunteer.Queries.GetVolunteerById;
using VolunteerLink.Core.Features.Volunteer.Queries.LogIn;
```

```
namespace VolunteerLink.Api.Controllers;

[ApiController]
[ApiVersion("1.0")]
[Authorize]
[Route("api/v{version:apiVersion}/volunteer")]
public class VolunteerController : ControllerBase
{
    private readonly IMediator _mediator;
    private readonly IMapper _mapper;

    public VolunteerController(
        IMediator mediator,
        IMapper mapper)
    {
        _mediator = mediator;
        _mapper = mapper;
    }

    /// <summary>
    /// Get the volunteer.
    /// </summary>
    /// <param name="volunteerId"> The user id.</param>
    /// <response code="200"> Returns the user.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpGet]
    [Route("{volunteerId}")]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(500)]
    public async Task<IActionResult> GetVolunteerAsync(int volunteerId)
    {
        var query = new GetVolunteerByIdQuery()
        {
            VolunteerId = volunteerId
        };

        var result = await _mediator.Send(query);
        return Ok(result);
    }

    /// <summary>
    /// Get all volunteers.
    /// </summary>
    /// <response code="200"> Returns users.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpGet]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(500)]
    public async Task<IActionResult> GetAllVolunteersAsync()
    {
        var query = new GetAllVolunteersQuery();
        var result = await _mediator.Send(query);
        return Ok(result);
    }

    /// <summary>
```

```
    /// Create the volunteer.
    /// </summary>
    /// <param name="request"> The user request.</param>
    /// <response code="200"> Returns the user that was created.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="409"> User already exists.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpPost]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(409)]
    [ProducesResponseType(500)]
    [AllowAnonymous]
    public async Task<IActionResult> CreateVolunteerAsync(VolunteerRequest request)
    {
        var command = _mapper.Map<CreateVolunteerCommand>(request);
        await _mediator.Send(command);

        return Ok();
    }

    /// <summary>
    /// Log in.
    /// </summary>
    /// <param name="request"> The log in user request.</param>
    /// <response code="200"> Returns the user that was created.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="409"> User already exists.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpPost("logIn")]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(409)]
    [ProducesResponseType(500)]
    [AllowAnonymous]
    public async Task<IActionResult> LogInAsync(LogInRequest request)
    {
        var command = _mapper.Map<LogInQuery>(request);
        var result = await _mediator.Send(command);

        return Ok(result.Token);
    }

    /// <summary>
    /// Update the volunteer.
    /// </summary>
    /// <param name="request"> The user request.</param>
    /// <response code="200"> Returns the user that was updated.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpPut]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(409)]
    [ProducesResponseType(500)]
    public async Task<IActionResult> UpdateVolunteerAsync(int volunteerId,
UpdateVolunteerRequest request)
    {
        var command = _mapper.Map<UpdateVolunteerCommand>(request);
        command.VolunteerId = volunteerId;
        await _mediator.Send(command);
```

```
        return Ok();
    }

    /// <summary>
    /// Generate change password code.
    /// </summary>
    /// <response code="200"> Returns the user that was updated.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpPatch("generateCode")]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(409)]
    [ProducesResponseType(500)]
    public async Task<IActionResult> GenerateChangePasswordCodeAsync()
    {
        var command = new SendCodeCommand();
        await _mediator.Send(command);

        return Ok();
    }

    /// <summary>
    /// Check change password code.
    /// </summary>
    /// <response code="200"> Returns the user that was updated.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpPatch("checkCode")]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(409)]
    [ProducesResponseType(500)]
    public async Task<IActionResult> CheckPasswordCodeAsync(CheckPasswordCodeRequest
request)
    {
        var command = _mapper.Map<CheckPasswordCodeCommand>(request);
        var result = await _mediator.Send(command);

        return Ok(result);
    }

    /// <summary>
    /// Change password.
    /// </summary>
    /// <response code="200"> Returns the user that was updated.</response>
    /// <response code="400"> Validation restrictions.</response>
    /// <response code="500"> If there was an internal server error.</response>
    [HttpPatch("changePassword")]
    [ProducesResponseType(200)]
    [ProducesResponseType(400)]
    [ProducesResponseType(409)]
    [ProducesResponseType(500)]
    public async Task<IActionResult> ChangePasswordAsync(ChangePasswordRequest request)
    {
        var command = _mapper.Map<ChangePasswordCommand>(request);
        await _mediator.Send(command);

        return Ok();
    }
```

```
}
```

CreateCommentCommandHandler.cs

```
using AutoMapper;
using MediatR;
using VolunteerLink.Core.Abstractions;
using VolunteerLink.Core.Abstractions.Repositories;

namespace VolunteerLink.Core.Features.Comment.Commands.CreateComment;

public class CreateCommentCommandHandler : IRequestHandler<CreateCommentCommand>
{
    private readonly IMapper _mapper;
    private readonly IIdentity _identity;
    private readonly ICommentRepository _commentRepository;

    public CreateCommentCommandHandler(
        IMapper mapper,
        ICommentRepository commentRepository,
        IIdentity identity)
    {
        _mapper = mapper;
        _commentRepository = commentRepository;
        _identity = identity;
    }

    public async Task Handle(CreateCommentCommand command, CancellationToken
cancellationToken = default)
    {
        var userId = _identity.UserId;
        var comment = _mapper.Map<Entities.Comment>(command);
        comment.AuthorId = userId;

        await _commentRepository.CreateAsync(comment);
    }
}
```

GetCommentsByEventIdQueryHandler.cs

```
using AutoMapper;
using MediatR;
using VolunteerLink.Core.Abstractions.Repositories;
using VolunteerLink.Core.Features.Comment.Dto;

namespace VolunteerLink.Core.Features.Comment.Queries.GetCommentsByEventId;

public class GetCommentsByEventIdQueryHandler :
IRequestHandler<GetCommentsByEventIdQuery, IEnumerable<CommentDto>>
{
    private readonly IMapper _mapper;
    private readonly ICommentRepository _commentRepository;

    public GetCommentsByEventIdQueryHandler(IMapper mapper, ICommentRepository
commentRepository)
    {
        _mapper = mapper;
        _commentRepository = commentRepository;
    }
```

```
    public async Task<IEnumerable<CommentDto>> Handle(GetCommentsByEventIdQuery query,
CancellationToken cancellationToken)
    {
        var comments = await _commentRepository.GetAllByEventIdAsync(query.EventId);

        return _mapper.Map<IEnumerable<CommentDto>>(comments);
    }
}
```

GetAllEventsQueryHandler.cs

```
using AutoMapper;
using MediatR;
using VolunteerLink.Core.Abstractions.Repositories;
using VolunteerLink.Core.Features.Event.Dto;

namespace VolunteerLink.Core.Features.Event.Queries.GetAllEvents;

public class GetAllEventsQueryHandler : IRequestHandler<GetAllEventsQuery,
IEnumerable<EventDto>>
{
    private readonly IMapper _mapper;
    private readonly IEventRepository _eventRepository;

    public GetAllEventsQueryHandler(IMapper mapper, IEventRepository eventRepository)
    {
        _mapper = mapper;
        _eventRepository = eventRepository;
    }

    public async Task<IEnumerable<EventDto>> Handle(GetAllEventsQuery request,
CancellationToken cancellationToken)
    {
        var events = await _eventRepository.GetAllAsync();

        if (request.OrganisationIds?.Count() > 0)
        {
            events = FilterByOrganisationIds(events, request.OrganisationIds);
        }

        if (request.TagIds?.Count() > 0)
        {
            events = FilterByTagIds(events, request.TagIds);
        }

        if (request.Cities?.Count() > 0)
        {
            events = FilterByCities(events, request.Cities);
        }

        if (request.StartDate is not null)
        {
            events = FilterByStartDate(events, request.StartDate);
        }

        if (request.EndDate is not null)
        {
            events = FilterByEndDate(events, request.EndDate);
        }

        return _mapper.Map<IEnumerable<EventDto>>(events);
```

```
    }

    private IEnumerable<Entities.Event>
FilterByOrganisationIds(IEnumerable<Entities.Event> events, IEnumerable<int>
organisationIds) =>
        events.Where(e => organisationIds.Contains(e.OrganisationId));

    private IEnumerable<Entities.Event> FilterByTagIds(IEnumerable<Entities.Event>
events, IEnumerable<int> tagIds) =>
       events.Where(e => e.Tags.Any(t => tagIds.Contains(t.TagId)));

    private IEnumerable<Entities.Event> FilterByCities(IEnumerable<Entities.Event>
events, IEnumerable<string> cities) =>
        events.Where(e => cities.Contains(e.City));

    private IEnumerable<Entities.Event> FilterByStartDate(IEnumerable<Entities.Event>
events, DateTime? startDate) =>
        events.Where(e => e.StartDate >= startDate);

    private IEnumerable<Entities.Event> FilterByEndDate(IEnumerable<Entities.Event>
events, DateTime? endDate) =>
        events.Where(e => e.EndDate <= endDate);
}
```

GetProfileByIdHandler.cs

```
using MediatR;
using VolunteerLink.Core.Abstractions;
using VolunteerLink.Core.Abstractions.Repositories;

namespace VolunteerLink.Core.Features.Volunteer.Queries.LogIn;

public class GetProfileByIdHandler : IRequestHandler<LogInQuery, LogInResponse>
{
    private readonly IVolunteerRepository _volunteerRepository;
    private readonly ITokenService _tokenService;
    private readonly IPasswordHasher _passwordHasher;

    public GetProfileByIdHandler(
        IVolunteerRepository volunteerRepository,
        ITokenService tokenService,
        IPasswordHasher passwordHasher)
    {
        _volunteerRepository = volunteerRepository;
        _tokenService = tokenService;
        _passwordHasher = passwordHasher;
    }

    public async Task<LogInResponse> Handle(LogInQuery request, CancellationToken
cancellationToken)
    {
        var volunteer = await _volunteerRepository.GetVolunteerByEmail(request.Email);

        if (volunteer is null)
        {
            throw new FormatException($"Volunteer with {request.Email} email does not
exist");
        }

        var jwt = _tokenService.GetJwt(volunteer.Id);
```

```
        var isValidPassword = _passwordHasher.VerifyPassword(request.Password,
volunteer.Password, volunteer.PasswordSalt);

        if (!isValidPassword)
        {
            throw new UnauthorizedAccessException($"Password incorrect");
        }

        return new LogInResponse
        {
            Id = volunteer.Id,
            Image = volunteer.Photo,
            Token = jwt,
        };
    }
}
```

PasswordHasher.cs

```
using Microsoft.AspNetCore.Cryptography.KeyDerivation;
using Microsoft.Extensions.Options;
using System.Security.Cryptography;
using System.Text;
using VolunteerLink.Core.Abstractions;

namespace VolunteerLink.Core.Services;

public sealed class PasswordHasher : IPasswordHasher
{
    private readonly IOptions<HashingOptions> _hashingOptions;

    public PasswordHasher(IOptions<HashingOptions> options)
    {
        _hashingOptions = options;
    }

    public byte[] GenerateSalt()
    {
        var salt =
RandomNumberGenerator.GetBytes(HashingOptions.EncryptionBytesLength);
        return salt;
    }

    public string ComputePassword(string rawPassword, byte[] passwordSalt)
    {
        var hashed = GetHexString(KeyDerivation.Pbkdf2(
               password: rawPassword,
               salt: passwordSalt,
               prf: KeyDerivationPrf.HMACSHA256,
               iterationCount: _hashingOptions.Value.Iterations,
               numBytesRequested: HashingOptions.EncryptionBytesLength));

        return hashed;
    }

    public byte[] GetBytes(string input) => Convert.FromHexString(input);

    public byte[] GetSigningKeyBytes(string input) => Encoding.UTF8.GetBytes(input);

    public string GetHexString(byte[] input) => Convert.ToHexString(input);
```

```
    public bool VerifyPassword(string requestPassword, string userPassword, string
passwordSalt)
    {
        var passHash = ComputePassword(requestPassword, GetBytes(passwordSalt));

        return passHash == userPassword
            ? true
            : false;
    }
}
```

TelegramBotSenderService.cs

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Telegram.Bot;
using Telegram.Bot.Types.ReplyMarkups;
using Telegram.Bot.Types;
using VolunteerLink.Core.Abstractions.Repositories;
using VolunteerLink.Core.Entities;
using System.Text;
using System.Text.RegularExpressions;

namespace VolunteerLink.Core.Services;

public class TelegramBotSenderService : BackgroundService
{
    private readonly TelegramBotClient _botClient;
    private readonly IServiceProvider _serviceProvider;

    public TelegramBotSenderService(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
        _botClient = new
TelegramBotClient("6757566657:AAGMAMBOa526ZOZ_afb98ZcFrMrp_IO811g");
    }

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            await SendPeriodicMessageAsync(stoppingToken);

            try
            {
                await Task.Delay(TimeSpan.FromMinutes(5), stoppingToken);
            }
            catch (TaskCanceledException ex)
            {
                Console.WriteLine(ex);
            }
        }
    }

    private async Task SendPeriodicMessageAsync(CancellationToken cancellationToken)
    {
        string EscapeMarkdownV2(string text)
        {
            return Regex.Replace(text, @"([_()~`>#+-=|{}.!])", @"\$1");
        }
```

```
        try
        {
            using (var scope = _serviceProvider.CreateScope())
            {
                var eventNotificationRepository =
scope.ServiceProvider.GetRequiredService<IEventNotificationRepostory>();

                var eventsForChats = await
eventNotificationRepository.GetAllEventsForChats();

                foreach (var events in eventsForChats)
                {
                    StringBuilder stringBuilder = new StringBuilder();

                    var distinctedEvents = events.DistinctBy(x => x.Id);

                    foreach (var @event in distinctedEvents)
                    {
                        stringBuilder.AppendLine($"*{@event.Title}* у місті
{@event.City} \n Дата проведення {@event.StartDate.ToString("dd/MM/yyyy HH:mm")}" +
                            $"\n Подробиці за посиланням
http://localhost:5173/events/{@event.Id} \n");
                        @event.IsSent = true;
                    };

                    await eventNotificationRepository.UpdateAsync(distinctedEvents);

                    string message = $"Нові події, які можуть вас зацікавити: \n\n
{stringBuilder}";
                    string escapedMessage = EscapeMarkdownV2(message);

                    await _botClient.SendTextMessageAsync(
                      chatId: events.Key,
                      text: escapedMessage,
                      cancellationToken: cancellationToken,
                      parseMode: Telegram.Bot.Types.Enums.ParseMode.MarkdownV2);
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

TokenService.cs

```
using JWT.Algorithms;
using JWT.Builder;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Cryptography;
using VolunteerLink.Core.Abstractions;

namespace VolunteerLink.Core.Services;

public class TokenService : ITokenService
{
    private readonly RSA _publicKey;
```

```
    private readonly RSA _privateKey;
    public TokenService(RSA publicKey, RSA privateKey)
    {
        _publicKey = publicKey;
        _privateKey = privateKey;
    }

    public string GetJwt(int userId)
    {
        var token = JwtBuilder.Create()
                              .WithAlgorithm(new RS256Algorithm(_privateKey,
_publicKey));
        token
            .AddClaim("exp", DateTimeOffset.UtcNow.AddHours(1).ToUnixTimeSeconds())
            .AddClaim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
            .AddClaim(JwtRegisteredClaimNames.Sub, userId);

        return token.Encode();
    }
}
```

TwilioAdapter.cs

```
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using Twilio;
using Twilio.Rest.Api.V2010.Account;
using Twilio.Types;
using VolunteerLink.Core.Abstractions;

namespace VolunteerLink.Core.Services;

public class TwilioAdapter : IAdapter
{
    private readonly ILogger<TwilioAdapter> _logger;
    private readonly IOptions<TwilioAdapterConfiguration> _options;

    public TwilioAdapter(
        ILogger<TwilioAdapter> logger,
        IOptions<TwilioAdapterConfiguration> options)
    {
        _logger = logger;
        _options = options;
    }

    public async Task SendSmsAsync(string recepientPhoneNumber, string messageBody)
    {
        TwilioClient.Init(_options.Value.AccountSid, _options.Value.AuthToken);

        var messageResource = await MessageResource.CreateAsync(
            body: messageBody,
            from: new PhoneNumber(_options.Value.SenderPhoneNumber),
            to: new PhoneNumber(recepientPhoneNumber));

        _logger.LogInformation($"Sms sent via Twilio. Status:
{messageResource.Status}");
    }
}
```

EventNotificationRepository.cs

```
using Microsoft.EntityFrameworkCore;
using VolunteerLink.Core.Abstractions.Repositories;
using VolunteerLink.Core.Entities;

namespace VolunteerLink.Infrastructure.Repositories;

public class EventNotificationRepository : IEventNotificationRepostory
{
    private readonly VolunteerLinkDbContext _context;

    public EventNotificationRepository(VolunteerLinkDbContext context)
    {
        _context = context;
    }

    public async Task<IEnumerable<IGrouping<long, Event>>> GetAllEventsForChats()
    {
        var eventsNotifications = await _context.EventNotifications
            .AsNoTracking()
            .ToListAsync();

        var events = await _context.Event
            .Include(x => x.Tags)
            .Where(x => x.StartDate >= DateTime.UtcNow && x.EndDate >= DateTime.UtcNow)
            .Where(x => x.IsSent == false)
            .ToListAsync();

        var chatEventDictionary = (from en in eventsNotifications
                                   from ev in events
                                   where ev.Tags.Any(x => x.TagId == en.TagId)
                                   select new
                                   {
                                       TelegramChatId = en.TelegramChatId,
                                       Event = ev
                                   })
                                   .GroupBy(x => x.TelegramChatId, x => x.Event);

        return chatEventDictionary;
    }

    public async Task<bool> SaveTagByChatIdAndTagId(int tagId, long chatId)
    {
        _context.EventNotifications.Add(new EventNotification() { TagId = tagId,
TelegramChatId = chatId});
        var res = await _context.SaveChangesAsync();

        return res > 0;
    }

    public async Task UpdateAsync(IEnumerable<Event> events)
    {
        _context.Event.UpdateRange(events);
        await _context.SaveChangesAsync();
    }
}
```

Addcomment.cs

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;
```

```
#nullable disable

namespace VolunteerLink.Infrastructure.Migrations
{
    /// <inheritdoc />
    public partial class Addcomment : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Comments",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    AuthorId = table.Column<int>(type: "int", nullable: false),
                    Commnet = table.Column<string>(type: "nvarchar(max)", nullable:
false),
                    CreatedDate = table.Column<DateTime>(type: "datetime2", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Comments", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Comments_Volunteer_AuthorId",
                        column: x => x.AuthorId,
                        principalTable: "Volunteer",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateIndex(
                name: "IX_Comments_AuthorId",
                table: "Comments",
                column: "AuthorId");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Comments");
        }
    }
}
```

VolunteerLinkDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using VolunteerLink.Core.Abstractions;
using VolunteerLink.Core.Entities;

namespace VolunteerLink.Infrastructure;

public class VolunteerLinkDbContext : DbContext, IDbContext
{
    public VolunteerLinkDbContext(
    DbContextOptions<VolunteerLinkDbContext> options) : base(options)
    {
```

```
    }

    public DbSet<Organisation> Organisation { get; set; }
    public DbSet<Volunteer> Volunteer { get; set; }
    public DbSet<Tag> Tag { get; set; }
    public DbSet<Event> Event { get; set; }
    public DbSet<VolounterOrgnization> VolounterOrgnization { get; set; }
    public DbSet<Comment> Comments { get; set; }
    public DbSet<Donation> Donation { get; set; }

    public DbSet<EventNotification> EventNotifications { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<VolounterOrgnization>()
            .HasOne(u => u.Organisation)
            .WithMany(j => j.Volunteers)
            .HasForeignKey(u => u.OrganisationId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<VolounterOrgnization>()
            .HasOne(u => u.Volunteer)
            .WithMany(j => j.Organisations)
            .HasForeignKey(u => u.VolounterId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<VolounterOrgnization>()
            .HasKey(u => new { u.VolounterId, u.OrganisationId });

        modelBuilder.Entity<Organisation>()
            .HasOne(e => e.Creator)
            .WithMany(e => e.OwnerOrganisations)
            .HasForeignKey(v => v.CreatorId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<EventTag>()
            .HasOne(u => u.Event)
            .WithMany(j => j.Tags)
            .HasForeignKey(u => u.EventId)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<EventTag>()
            .HasOne(u => u.Tag)
            .WithMany(j => j.Events)
            .HasForeignKey(u => u.TagId)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<EventNotification>()
            .HasOne(u => u.Tag)
            .WithMany(j => j.EventNotifications)
            .HasForeignKey(u => u.TagId)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<EventTag>()
            .HasKey(u => new { u.EventId, u.TagId });
    }

}
```

Volunteer.cs

```
namespace VolunteerLink.Core.Entities;

public class Volunteer : BaseEntity
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Gender Gender { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string? Photo { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public string Password { get; set; }
    public string PasswordSalt { get; set; }
    public string? ChangePasswordCode { get; set; }
    public IEnumerable<VolounterOrgnization>? Organisations { get; set; }
    public IEnumerable<Organisation>? OwnerOrganisations { get; set; }
}
```

Organisation.cs

```
using System.ComponentModel.DataAnnotations.Schema;

namespace VolunteerLink.Core.Entities;

public class Organisation : BaseEntity
{
    public string OrganisationName { get; set; }
    public OrganisationType OrganisationType { get; set; }
    public string Goal { get; set; }
    public string Description { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
    public string? Address { get; set; }
    public string? WebSite { get; set; }
    public int CreatorId { get; set; }
    public string? Logo { get; set; }
    public Volunteer Creator { get; set; }
    public ICollection<VolounterOrgnization>? Volunteers { get; set; }
}
```

OrganisationRepository.cs

```
using Microsoft.EntityFrameworkCore;
using VolunteerLink.Core.Abstractions.Repositories;
using VolunteerLink.Core.Entities;

namespace VolunteerLink.Infrastructure.Repositories;

public class OrganisationRepository : IOrganisationRepository
{
    private readonly VolunteerLinkDbContext _context;

    public OrganisationRepository(VolunteerLinkDbContext context)
    {
        _context = context;
    }

    public async Task<IEnumerable<Organisation>> GetAllAsync()
    {
        return await _context.Organisation.Include(o => o.Creator)
```

```
            .ToListAsync();
    }
    public async Task CreateAsync(Organisation organisation)
    {
        await _context.Organisation.AddAsync(organisation);
        await _context.SaveChangesAsync();
    }

    public async Task<Organisation?> GetByIdAsync(int organisationId)
    {
        return await _context.Organisation
            .Include(o => o.Creator)
            .FirstOrDefaultAsync(cc => cc.Id == organisationId);
    }

    public async Task<IEnumerable<Organisation>> GetOrganisationsByCreatorId(int
cratorId)
    {
        return await _context.Organisation
            .Include(o => o.Creator)
            .Where(cc => cc.CreatorId == cratorId)
            .ToListAsync();
    }

    public async Task UpdateAsync(Organisation organisation)
    {
        _context.Organisation.Update(organisation);
        await _context.SaveChangesAsync();
    }

    public async Task JoinOrganisationAsync(VolounterOrgnization volounterOrgnization)
    {
        await _context.VolounterOrgnization.AddAsync(volounterOrgnization);
        await _context.SaveChangesAsync();
    }
}
```

CommentRepository.cs

```
using Microsoft.EntityFrameworkCore;
using VolunteerLink.Core.Abstractions.Repositories;
using VolunteerLink.Core.Entities;

namespace VolunteerLink.Infrastructure.Repositories;

public class CommentRepository : ICommentRepository
{
    private readonly VolunteerLinkDbContext _context;

    public CommentRepository(VolunteerLinkDbContext context)
    {
        _context = context;
    }

    public async Task CreateAsync(Comment comment)
    {
        await _context.Comments.AddAsync(comment);
        await _context.SaveChangesAsync();
    }

    public async Task<Comment?> GetByIdAsync(int commentId)
```

```
    {
        return await _context.Comments
            .AsNoTracking()
            .FirstOrDefaultAsync(cc => cc.Id == commentId);
    }

    public async Task<IEnumerable<Comment>> GetAllByEventIdAsync(int eventId)
    {
        return await _context.Comments
            .AsNoTracking()
            .Include(x => x.Author)
            .Where(cc => cc.EventId == eventId)
            .ToListAsync();
    }

    public async Task DeleteAsync(Comment comment)
    {
        _context.Comments.Remove(comment);
        await _context.SaveChangesAsync();
    }
}
```

ДОДАТОК Б

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу бакалавра
на тему:
«Розробка інформаційної системи для волонтерських організацій «Шлях до
перемоги» мовою програмування C#»
студентки групи 122-20-1 Котенко Олександри Павлівни

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н. **Л. В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом Котенко.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом Котенко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
diplom.zip	Архів. Містить коди програми.
Презентація	
Презентація Котенко.pptx	Презентація кваліфікаційної роботи.