НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

_____ /_____Голуб Б.Л._____/
(підпис) (ПІБ)

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Інформаційна система комунікації між волонтерами та військовими»

Спеціальність 122 – «Комп'ютерні науки»
ОП «Комп'ютерні науки»

Гарант освітньої програми
_____д.е.н.,_професор___ _____ _____Руденський Р.А._____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
_____к.т.н, доцент_____ _____ _____Голуб Бела Львівна_____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Гаврилюк Денис Вячеславович___
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Комп'ютерних наук_____
(назва кафедри)

___к.т.н., доцент_____Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"_16_" _грудня___ 2024 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Гаврилюку Денису Вячеславовичу

Спеціальність 122 – «Комп'ютерні науки»

ОП «Комп'ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Інформаційна система комунікації між волонтерами та військовими» затверджена наказом ректора НУБіП України від 16.12.2024 № _2246_С_
2. Термін подання завершеної роботи на кафедру__2025.05.25_____
(рік, місяць, число)
3. Вихідні дані до роботи: спосіб формування зборів коштів на платформах волонтерських фондів, соціальних мережах.
4. Перелік питань, що розглядаються:
   - Аналіз комунікації між волонтерами та військовими.
   - Проєктування і розробка інформаційного забезпечення системи.
   - Проєктування і розробка програмного забезпечення системи.
   - Впровадження та експлуатація системи.

Дата видачі завдання "_16_"_грудня_____ 2024 р.

Керівник бакалаврської кваліфікаційної роботи _____/ Голуб Б.Л. /
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:_____/ Гаврилюк Д.В. /
(підпис) (прізвище та ініціал

ЗМІСТ

Перелік умовних позначень

API – Application Programming Interface

AWS – Amazon Web Services

CD – Continue Delivary

CI – Continue Integration

CORS – Cross-Origin Resource Sharing

CRUD – Create Update Delete

DOM – Document Object Model

DTO – Data Transfer Object

ER – Entity-Relationship

HTTP – Hypertext Transfer Protocol

IDE – Intagrated Drive Electronics

IP – Internet Protocol

JSON – JavaScript Object Notation

JWT – JSON Web Tokens

NPM – Node Package Manager

ORM – Object-relational mapping

PNPM – Perfomant NPM

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

SSMS – SQL Server Management Studio

TCP – Transmission Control Protocol

YARN – Yet Another Resource Negotiator

БД – База даних

ПЗ – Програмне забезпечення

СУБД – Система Управління Базами Даних

РСУБД – Реляційна система управління базами даних

ВСТУП

Актуальність. Починаючи з 2014 через розпочату війну в Україні, став зростати волонтерський рух і досяг пікової активності у 2022, через що збільшилася необхідність ефективної взаємодії між волонтерськими організаціями та військовими структурами. Проблемна ситуація полягає у відсутності єдиної платформи для реєстрації, координації та адміністрування діяльності фондів і військових угруповань ускладнює взаємодію та прозорість збору коштів і ресурсів. Необхідно розробити систему, яка забезпечить надійну комунікацію, централізований облік і контроль над запитами та фінансуванням.

Мета розробки. Метою розробки є вирішення проблеми відсутності надійної комунікації, децентралізованого обліку і безконтрольного фінансування шляхом розробки інформаційної системи комунікації між волонтерами та військовими. Система дозволить автоматизувати облік зборів коштів, військових запитів і підтвердження взаємодій між фондами та бригадами.

Інформаційна система комунікації між волонтерами та військовими це значний крок у цифровій трансформації неформального логістичного забезпечення, формуючи підходи для інтеграції волонтерського руху з військовими структурами.

Методи та технології. У процесі реалізації системи було застосовано сучасний технологічний стек, орієнтований на створення масштабованого, безпечного та зручного у використанні веб-рішення з чітким розділенням відповідальності між клієнтською та серверною частиною. Основні технології охоплюють такі компоненти:

1. **ASP.NET Core Web API** – фреймворк на базі C#, який забезпечує розробку надійних, продуктивних і кросплатформних застосунків. Він оброблює HTTP-запити, забезпечує авторизацію та автентифікацію

користувачів, маршрутизацію, а також відповідальний за реалізацію бізнес-логіки.

2. **Entity Framework Core** – ORM-фреймворк для роботи з базами даних. Він значно спрощує доступ до СУБД, забезпечуючи відокремленість логіки збереження даних від решти застосунку. Крім того, інструмент підтримує міграції та забезпечує перевірку цілісності й взаємозв'язків між таблицями на рівні коду. Використання SQL Server 2024, продукт від компанії Microsoft, створює синергію між компонентами, що позитивно впливає на їхню інтеграцію та ефективність.

3. **Microsoft SQL Server 2024** для управління даними. Вона дає можливість ведення ефективного обліку користувачів, фондів, запитів, ролей, а також дозволяє створювати представлення, збережені процедури і обмеження для коректної обробки та підтримки цілісності даних.

4. **React** з використанням **TypeScript** – клієнтська частина, яка дозволяє створювати інтерактивні, багаторазово використовувані компоненти інтерфейсу користувача, маючи лише одну сторінку. Використання **TypeScript** – надбудови над **JavaScript** – підвищує безпеку коду, контролювання типізації та дозволяє виявляти помилки на етапі компіляції. У поєднані з **React** це дає змогу створювати структурований, масштабований клієнтський код із розвиненою системою типів і строгими інтерфейсами.

Апробація програмного додатку. Було опубліковано результати дипломної роботи у тезах виступу на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем '2025'»: **Гаврилюк Д. В.**, науковий керівник Голуб Б. Л. Інформаційна система комунікації між волонтерами та військовими / Тези доповіді на студентській конференції «Теоретичні та прикладні аспекти розробки комп'ютерних систем '2025'». – Київ, 2025.

Структура записки. Записка складається з 68 сторінок, 40 використаних джерел та 8 Додатків. Основна частина розділена на 4 розділи:

- **Аналіз комунікації між волонтерами та військовими.** У цьому розділі описується волонтерська діяльність в Україні, їх взаємодія та комунікація з військовими. Також описано існуючі волонтерські організації та яка спеціалізація кожного з них. Моделюється предметна область і постановка завдання.
- **Проектування і розробка інформаційного забезпечення системи.** У другому розділі проектується логічна модель даних, описуються сутності, їх атрибути та зв'язки. Також обґрунтовується вибір СУБД для реалізації бази даних. У кінці розділу створюється БД, описуються створені об'єкти БД.
- **Проєктування і розробка програмного забезпечення системи.** У цьому розділі описується організаційна структура програмного забезпечення, продемонстрована діаграма пакетів. Обґрунтовується вибір інструментарію для створення ПЗ, а також використані технології та бібліотеки. Проектуються backend і frontend застосунки, а також взаємодія з БД.
- **Впровадження та експлуатація системи.** У четвертому розділі демонструється фізична архітектура системи за допомогою діаграми розгортання. Описуються основні компоненти системи та взаємозв'язки між ними. Описані апаратні та програмні вимоги системи, відповідно до кожного компонента архітектури системи, використаних технологій та бібліотек. Також описано інсталяційний пакет і тестування системи.

1 АНАЛІЗ КОМУНІКАЦІЇ МІЖ ВОЛОНТЕРАМИ ТА ВІЙСЬКОВИМИ

1.1 Опис комунікації між волонтерами та військовими

Волонтерський рух в Україні. Сьогоднішній волонтерський рух в Україні є потужною та гнучкою мережею добровольців, які надають допомогу у різних сферах – від військової підтримки до соціальної, гуманітарної, медичної та психологічної допомоги. Особливо з 2014 року, а ще більше з початку повномасштабної війни у 2022 році, волонтерство стало масовим суспільним явищем, що охоплює всі регіони України та навіть закордонну діаспору.

Хто такий волонтер? Волонтер – це людина, яка добровільно, без матеріальної винагороди, допомагає іншим або підтримує певні ініціативи. Це можуть бути як окремі особи, так і об’єднання, такі як фонди, волонтерські групи, громадські організації.

Як виглядає волонтерська допомога в Україні? Вона може проявлятися у різних формах:

- Збір коштів, амуніції, продуктів та техніки для ЗСУ.
- Евакуація цивільного населення з зон бойових дій.
- Медична допомога, транспортування поранених.
- Ремонт та будівництво тимчасового житла.
- Допомога внутрішньо переміщеним особам.
- Координація логістики, перевезення гуманітарних вантажів.
- Інформаційна та психологічна підтримка.

Комунікація. Основна комунікація між волонтерами та військовими побудована таким чином.

1) Реєстрація та вступ до організацій.

Волонтери починають процес, заповнюючи форму реєстрації волонтерського фонду, що є першим кроком до вступу до складу організації. Якщо фонд уже зареєстрований, волонтер може запросити вступ до нього. Цей запит перевіряється і, за необхідності, підтверджується. Аналогічно, військові формують запит на вступ до військового угрупування, що також потребує перевірки та затвердження.

2) Створення та управління зборами коштів.

Після вступу до організації, волонтери або військові можуть ініціювати створення зборів коштів, які пов’язані із запитами на допомогу, волонтерськими проектами або конкретними потребами, що забезпечують фінансування.

3) Створення та управління військовими запитами.

У випадку, коли військовим необхідно отримати деяку матеріальну допомогу, а можливості та часу купувати необхідні товари немає, вони можуть створити військові запити, які будуть доступні лише волонтерським організаціям. У військовому запиті можна вказати сам предмет допомоги, його кількість, встановити категорії, прописати деякі особливості тощо.

4) Виконання військових запитів.

Після того, як військові запити було створено, волонтерські організації можуть встановити зв’язок із конкретним військовим формуванням, уточнити деталі запиту та запросити дозвіл на виконання. У разі схвалення військовими волонтери виконують заданий запит. Після успішного або неуспішного виконання запиту, військові закривають його.

5) Формування звітів.

Раз на місяць формуються звіти про витрати, отримані кошти та виконання військових запитів військових угруповань та волонтерських

фондів. Це забезпечує прозорість процесу, підзвітність та допомагає контролювати ефективність використання ресурсів.

Основна діяльність найбільших існуючих волонтерських організацій в Україні. Як приклади успішних волонтерських організацій можна навести фонд «Повернись живим», «Фонд Сергія Притули», «Пласт – Національна скаутська організація України», «Razom for Ukraine». Основна діяльність кожного відрізняється, так наприклад фонд «Повернись живим», заснований у 2014 році, займається постачанням технічного обладнання ЗСУ, навчання військових, аналітика та моніторинг ситуації на фронті, звіти про витрати коштів тощо. Фонд Сергія Притули направлений більше на закупівлі для армії дронами, бронетехнікою, автівками, тепловізорами, Starlink, а також розвиває різні компанії як «Байрактар для ЗСУ», «Мільйон для контрнаступу», «Народний супутник». А така організація як «Пласт – національна скаутська організація України» була заснована ще 1911 року (відновлена у незалежній Україні з 1990-х), коли Україна не мала статусу незалежної держави та не могла мати свою армію та зброю, тому волонтерська організація була направлена на гуманітарну допомогу, таку як допомога цивільним у період війни, організація таборів, логістики, евакуації, навчання молоді патріотизму, першій медичній допомозі та дисципліні.

1.2 Моделювання предметної області

Як результат аналізу предметної області, були побудовані моделі на основі UML, яка дозволяє спроектувати предметну область в об’єктно-орієнтованому стилі.

Для представлення діючих осіб (акторів) та їхньої взаємодії з предметною областю (прецедентів) була побудована діаграма **Use-Case** (Додаток А, рис. 1).

Були виділені основні діючі особи:

- Волонтер;

- Військовий;
- Волонтерський фонд;
- Військове угруповання.

У Таблиці 1 представлено зв’язок діючої особи з прецедентами.

Таблиця 1

Прецедент	**Актор**
Формування звіту по зборах коштів	Військове угруповання, Волонтерський фонд
Формування звіту по загальних витратах та надходжені кошти	Військове угруповання, Волонтерський фонд
Формування звіту по військовим запитам	Військове угруповання, Волонтерський фонд
Створення військового запиту	Військове угруповання
Створення збору коштів	Військове угруповання, Волонтерський фонд
Виконання військового запиту	Волонтерський фонд
Створення волонтерського проєкту	Волонтерський фонд
Вступ до складу волонтерського фонду	Волонтер
Вступ до складу військового угруповання	Військовий

Діаграма прецедентів демонструє лише зв’язок між діючими особами та прецедентами. Для визначення послідовності дій була побудована **діаграма послідовності**, яка представлена в Додатку А рис. 2. В якості об’єктів виступають актори, які обмінюються повідомленнями. У кожний момент часу активно лише одне повідомлення. Для демонстрації логіки взаємодії діючих осіб побудована **діаграма активності**, яка представлена в Додатку А рис. 3.

Для кожної діючої особи виділена доріжка, в осередку якої продемонстровано її активність.

1.3 Огляд інформаційних джерел та існуючих рішень

Серед існуючих рішень можна навести будь-який офіційний сайт благодійного або волонтерського фонду. Їх дуже багато, але головна проблема полягає в тому, що вони існують по принципу **«кожен сам за себе»**. Приклади існуючих сайтів продемонстровано на рис. 1.

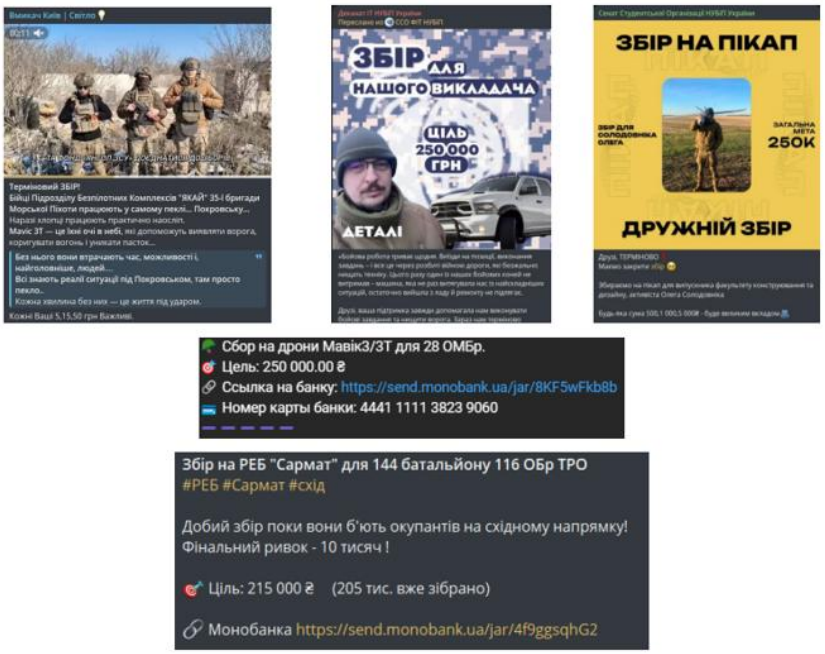

Рис. 1 Приклади оголошень зборів коштів на різних платформах

Децентралізований волонтерський рух в Україні став одним із символів стійкості громадянського суспільства, однак у сучасних умовах така структура має і свої **недоліки**, особливо коли йдеться про використання соціальних мереж та розрізнені платформи збору коштів.

Основні недоліки використання соцмереж та розпорошених каналів комунікації:

- Відсутність єдиного центру достовірної інформації.
- Низька прозорість та підвищений ризик шахрайства.
- Інформаційна перевантаженість.

- Недостатня координація між волонтерами.
- Втрата довіри з боку спонсорів та донорів.

Основні недоліки децентралізованих зборів коштів:

- Карткові перекази – непрозора система.
- Важко відстежити історію платежів.
- Міжнародним донорам складно переказувати кошти.
- Нестача технічних засобів безпеки.

Як підсумок можна сказати, що децентралізація створює гнучкість і швидкість реакції, але без належної координації, верифікації та об’єднання зусиль призводить до неефективної взаємодії, плутанини, шахрайства та зростання кіберризиків. Саме тому необхідно все більше рухатися до створення єдиних платформ обліку, систем звітності й комунікації хоча б на регіональному або галузевому рівні.

1.4 Постановка завдання

Необхідно розробити веб-платформу для автоматизації управління волонтерськими фондами та військовими угрупованнями, що забезпечить прозору реєстрацію, адміністрування та координацію їхньої діяльності. Система дозволить автоматизувати облік зборів коштів, військових запитів і підтвердження взаємодії між фондами та бригадами.

Функціональні вимоги

Система повинна забезпечувати наступний функціонал:

- Реєстрація та автентифікація користувачів.
- Створення та адміністрування волонтерських фондів.
- Реєстрація та підтвердження бригад.
- Управління діяльністю фондів і бригад.
- Облік зборів коштів із підтвердженням бригади.

- Облік військових запитів та їх виконання.
- Звітування волонтерів та військових щодо загальних витрат, зборів коштів та військових запитів.

Нефункціональні вимоги

- Безпека: Використання механізмів автентифікації та авторизації для захисту даних користувачів та управління фондами й бригадами.
- Продуктивність: Оптимізована обробка запитів і масштабованість системи для роботи з великою кількістю користувачів.
- Доступність: Підтримка веб-середовища та мобільних пристроїв.
- Надійність: Запобігання несанкціонованим змінам у структурі фондів і бригад через механізми підтвердження запитів

2 ПРОЕКТУВАННЯ І РОЗРОБКА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Логічна модель даних

ER-модель. Перед програмуванням системи необхідно спроектувати базу даних. Почати треба з моделювання сутностей та зв’язків між ними за допомогою ER-діаграми. ER-діаграма або ER-модель – модель даних, яка відображає сутності (таблиці), їхні атрибути та взаємозв’язки між ними. Використовується для зрозумілого та логічного опису предметної області. Для моделювання даних системи було побудовано ER-діаграму та зображено на рис. 2.

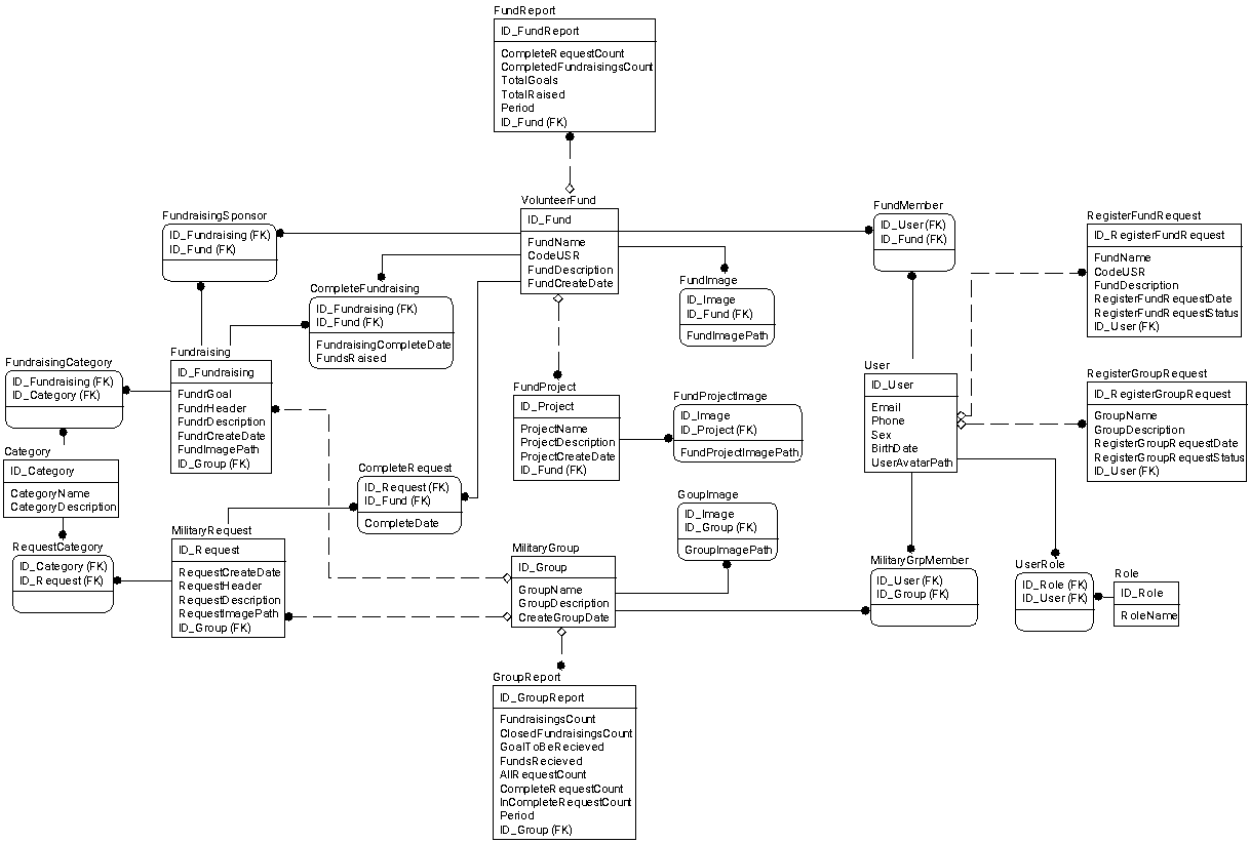

Рис. 2 Логічна модель даних.

Опис ER-моделі. Задана модель має такі сутності:

1) **User -** незалежна батьківська сутність.
   - Атрибути:
     - ID_User (PK)
     - Email, Phone, Sex, BitrhDate, UserAvatarPath
   - Зв’язки:
     - UserRole (1:N з Role)
     - FundMember, MilitaryGrpMember
     - RegisterFundRequest, RegisterGroupRequest (запити на реєстрацію фондів або угруповань)

2) **Role –** незалежна батьківська сутність.
   - Атрибути:
     - ID_Role (PK)
     - RoleName
   - Зв’язки:
     - UserRole

3) **UserRole** - залежна дочірня сутність.
   - Зв’язкова таблиця між User і Role
   - Ключі: складений PK з ID_User та ID_Role

4) **VolunteerFund –** незалежна батьківська сутність.
   - Атрибути:
     - ID_Fund (PK)
     - FundName, CodeUSR, FundDescription, CreateDate
   - Зв’язки:
     - FundMember (1:N з User)
     - FundImage
     - FundProject

- CompleteFundraising (1:N з Fundraising), FundraisingSponsor (1:N з Fundraising)

5) FundMember – залежна дочірня сутність.

- Зв’язкова таблиця між VolunteerFund і User
- Ключі: складений PK з ID_User та ID_Fund

6) FundProject – батьківська сутність, залежить від ключа VolunteerFund.

- Атрибути:
  - ID_Project (PK)
  - ProjectName, ProjectDescription, ProjectCreateDate
  - ID_Fund (FK)
- Зв’язки:
  - VolunteerFund, FundProjectImage

7) FundReport – батьківська сутність, залежить від ключа VolunteerFund.

- Атрибути:
  - ID_FundReport (PK)
  - CompleteRequestCount, CompletedFundraisingCount, TotalGoals, TotalRaised, Period
  - ID_Fund (FK)
- Зв’язок: VolunteerFund

8) MilitaryGroup – незалежна батьківська сутність.

- Атрибути:
  - ID_Group (PK)
  - GroupName, GroupDescription, CreateGroupDate
- Зв’язки:
  - MilitaryGrpMember (1:N з User)
  - GroupImage

- Fundraising
- MilitaryRequest
- GroupReport

9) MilitaryGrpMember – залежна дочірня сутність.

- Зв’язкова таблиця з MilitaryGroup та User
- Ключі: складений PK з ID_User та ID_Group

10) GroupReport – батьківська сутність, залежить від сутності MilitaryGroup.

- Атрибути:
  - ID_GroupReport (PK)
  - FundraisingsCount, ClosedFundraisingsCount, GoalToBeRecieved, FundsRecieved, CompleteRequestCount, InCompleteRequestCount, Period
  - ID_Group (FK)
- Зв’язок: VolunteerGroup

11) Fundraising – батьківська сутність, залежить від ключа MilitaryGroup.

- Атрибути:
  - ID_Fundraising (PK)
  - FundrGoal, FundrHeader, FundrDescription, FundrCreateDate, FundImagePath
  - ID_Group (FK)
- Зв’язки:
  - CompleteFundraising (1:N з VolunteerFund), FundraisingSponsor (1:N з VolunteerFund)
  - FundraisingCategory (1:N з Category)
  - MilitaryGroup

12) **FundraisingSponsor**, **CompleteFundraising –** залежні дочірні сутності.

- Зв’язкові таблиці між VolunteerFund та Fundraising
- Ключі: складений PK з ID_Fundraising та ID_Fund

13) Category – незалежна батьківська сутність.

- Атрибути:
  - ID_Category (PK)
  - CategoryName, CategoryDescription
- Зв’язки:
  - RequestCategory (1:N з Request)
  - FundraisingCategory (1:N з Fundraising)

14) FundraisingCategory – залежна дочірня сутність.

- Зв’язкова таблиця між Fundraising та Category
- Ключі: складений PK з ID_Fundraising і ID_Category

15) MilitaryRequest – батьківська сутність, залежить від ключа MilitaryGroup.

- Атрибути:
  - ID_Request (PK)
  - RequestCreateDate, RequestHeader, RequestDescription, RequestImagePath,
  - ID_Group (FK)
- Зв’язки:
  - RequestCategory (1:N з Category)
  - CompleteRequest (1:N з VolunteerFund)
  - MilitaryGroup

16) RequestCategory – залежна дочірня сутність.

- Зв’язкова таблиця між MilitaryRequest та Category
- Ключі: складений PK з ID_Request і ID_Category

17) **CompleteRequest –** залежеа дочірня сутність.

- Зв’язкова таблиця між MilitaryRequest та VolunteerFund
- Ключі: складений PK з ID_Request і ID_Fund

18) **RegisterFundRequest –** батьківська сутність, залежить від ключа User.

- Атрибути:
  - ID_RegisterFundRequest (PK)
  - FundName, CodeUSR, FundDescription, RegisterFundRequestDate, RegisterFundRequestStatus
  - ID_User (FK)
- Зв’язок: User

19) **RegisterGroupRequest –** батьківська сутність, залежить від ключа User.

- Атрибути:
  - ID_RegisterGroupRequest (PK)
  - GroupName, GroupDescription, RegisterGroupRequestDate, RegisterGroupRequestStatus
  - ID_User (FK)
- Зв’язок: User

Нормалізація структури БД. Вимоги першої нормальної форми (**1NF**) полягають у тому, щоб усі поля мали атомарні або неділимі значення, а всі записи – у вигляді таблиць із фіксованими колонками. Задана модель даних виконує першу нормальну форму, адже у неї немає списків або масивів у полях, а значення такі як Email, Sex, FundName тощо – атомарні.

Щодо другої нормальної форми (**2NF**) вимоги такі: таблиця повинна бути в першій нормальній формі, усі неключові атрибути мають залежати від всього первинного ключа, а не його частини. У заданій моделі даних в

таблицях як UserRole із складними первинними ключами жоден атрибут не залежить лише від одного з полів ключа.

Третя нормальна форма (**3NF**) вимагає, щоби модель була в другій нормальній формі та усі неключові атрибути повинні напряму залежати від ключа (тобто, відсутність транзитивної залежності). Наприклад, у базі даних MilitaryAidDB у таблиці User усі його неключові атрибути (UserName, Sex, Phone, BirthDate) залежать безпосередньо від ключа ID_User, тож ця таблиця відповідає третій нормальній формі.

Сутності User, VolunteerFund, MilitaryGroup, RegisterFundRequest, RegisterGroupRequest, Role, FundProject, Fundraising, MilitaryRequest і Category відповідають нормальній формі **Бойса-Кода**, оскільки містять неключові атрибути, які можуть стати ключем (альтернативні ключі).

2.2 Вибір системи управління інформаційною базою

Задана модель даних охоплює численні зв’язки між сутностями, підтримує нормалізовану структуру даних, використовує зовнішні ключі та вимагає забезпечення цілісності даних. Така структура є характерною для **реляційної** моделі даних, де сутності представляються у вигляді таблиць, а зв’язки реалізуються через ключі та обмеження. Тому для реалізації інформаційної бази даних доцільно використовувати саме **реляційну систему управління базами даних (РСУБД).**

Реляційна модель забезпечує надійне зберігання даних, запити за допомогою мови SQL, підтримку транзакцій та гнучких зв’язків між таблицями. У ролі прикладів РСУБД можна розглядати **MySQL**, **SQL Server**, **PostgreSQL**, **Oracle** чи **MS Access** для локальних рішень або невеликих систем, однак для корпоративного рівня та побудови клієнт-серверної архітектури потрібне більш потужне рішення.

В якості СУБД було обрано **Microsoft SQL Server** – потужна реляційна система управління базами даних, яка є однією з найпопулярніших у корпоративному середовищі завдяки надійності, масштабованості та легкої

інтеграції з іншими продуктами компанії Microsoft. Основна причина вибору цієї СУБД полягає в тому, що серверна частина та тестування розроблюються мовою C#, яка також створена компанією Microsoft на фреймворку .NET, що добре синергує з SQL Server.

Розташування БД у хмарному середовищі Google Cloud Platform. Щоб створити базу даних MS SQL Server у **Google Cloud Platform (GCP)**, потрібно скористатися сервісом Cloud SQL, де підтримується SQL Server. У GCP Console створюється інстанс Cloud SQL із вибором SQL Server як типу СУБД, задається конфігурація (версія, регіон, паролі, розмір диска), після чого база даних стає доступною через публічну IP-адресу або приватну мережу з можливістю підключення через SSMS чи застосунки.

Детальні кроки створення БД у GCP:

1) **Підготовка:**
   - зареєструватися в **Google Cloud Platform**;
   - увімкнути Billing (додати платіжну картку);
   - активувати Cloud SQL API у розділі API & Services / Library.
2) **Створення інстансу Cloud SQL (SQL Server):**
   - перейти до розділу SQL;
   - натиснути "Create Instance";
   - обрати тип бази даних — "Microsoft SQL Server";
   - вказати:
     - ім’я інстансу (унікальне);
     - пароль адміністратора sqlserver;
     - версію SQL Server (наприклад, 2019 Express, Standard тощо);
     - регіон і зону, у якій розмістити інстанс;
     - тип машини (CPU, RAM) — наприклад, db-custom-1-3840;
     - розмір SSD-диску;
     - опціонально — увімкнути High availability, backups, maintenance window.

3) **Налаштування доступу:**
   - увімкнути Public IP, якщо планується зовнішнє підключення;
   - у розділі Connections → Authorized networks додати IP-адресу свого комп'ютера (або 0.0.0.0/0 для відкритого доступу — не рекомендується);
   - опціонально — створити Private IP через VPC.
4) **Підключення до бази:**
   - після створення інстансу — перейти до розділу Users, створити додаткового користувача.
   - відкрити SSMS і ввести:
     - Server name – публічна IP-адреса інстансу, порт 1433;
     - Authentication – SQL Server Authentication;
     - Login – sqlserver;
     - Password – той, що вказано під час створення.
5) **Робота з базою:**
   - після підключення — створити базу (CREATE DATABASE), таблиці, збережені процедури тощо.

На рис. 3 показано фрагмент конфігурації хмарного інстансу.

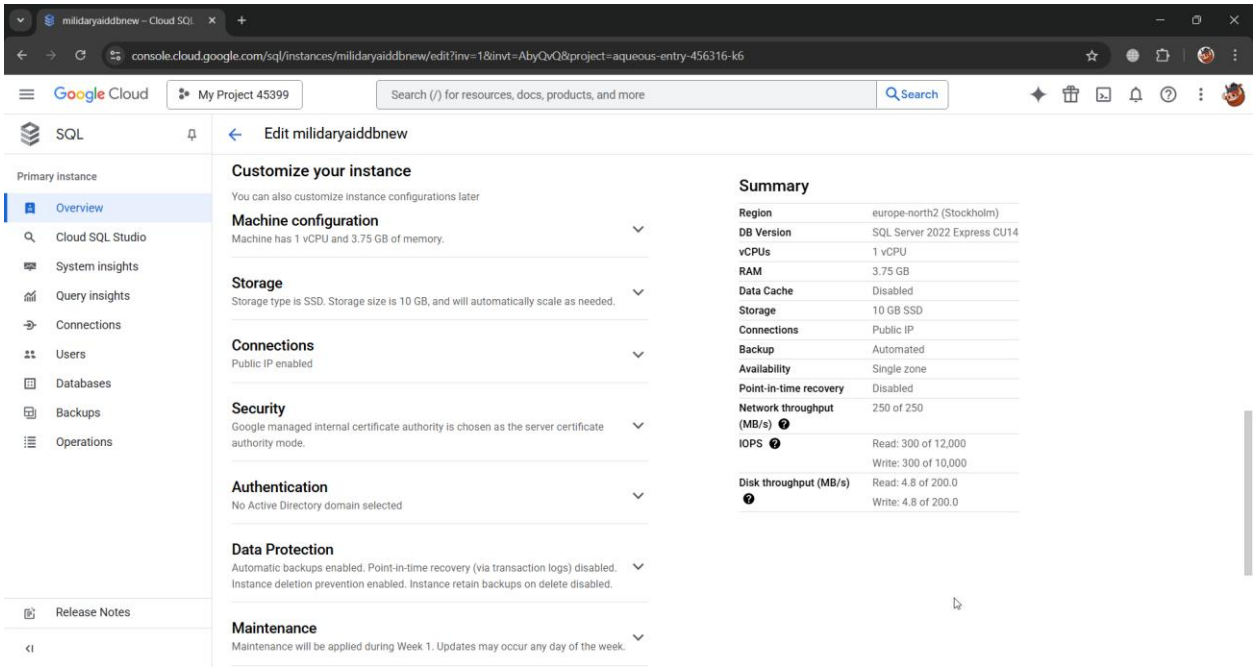

Рис. 3 Сторінка конфігурації інстансу в Google Cloud Platform

2.3 Створення бази даних

2.3.1 Таблиці в базі даних. База даних MilitaryAidDB побудована для підтримки взаємодії між волонтерами, фондами та військовими угрупованнями. Вона охоплює повний цикл від реєстрації користувачів і подання заявок до збору коштів, запитів від військових, її виконання та звітності. Код створення БД та таблиць вказано в Додатку Б.

Маємо такі базові таблиці як User, Role та UserRole – таблиці користувачів та їх ролі («User», «Волонтер», «Військовий», «Волонтерський фонд», «Військове угруповання» або «Адміністратор»).

Також створено волонтерські фонди, їх проекти учасники та звіти: VolunteerFund, FundMember, FundImage, FundProject та FundProjectImage, FundReport

Щодо військових угруповань створено таблиці MilitaryGroup, MilitaryGroupMember, GroupImage.

Основні військові запити, їх облік та звіти реалізовані у таблицях MilitaryRequest, RequestCategory, CompleteRequest, GroupReport.

Збори коштів обґрунтовані такими таблицями як Fundraising, FundraisingCategory, FundraisingSponsor, CompleteFundraising.

Також для реєстраційних запитів створено дві додаткові таблиці RegisterFundRequest та RegisterGroupRequest.

Зв’язки між таблицями. Таблиці пов’язані наступним чином:

- Користувачі – фонди / групи через таблиці FundMember / MilitaryGrpMember
- Запити та збори – прив’язані до військового угруповання.
- Виконані запити та збори – пов’язані з фондом, який їх закрив.
- Категоріях – використовуються і в запитах, і в зборах коштів.
- Звіти – формуються для фондів та груп по періодах.

Закінчення опису таблиць та залежності між ними продемонстровано фізичною моделлю даних на Додатку В.

2.3.2 Опис процедур. Збережена процедура (stored procedure) – набір SQL-інструкцій, які виконуються на сервері.

Збережені процедури дозволяють:

1) інкапсулювати логіку обробки даних;
2) повторно виконувати транзакції;
3) обмежувати права користувачів БД на обробку даних;
4) оптимізувати виконання запитів.

Основні процедури системи поділені на процедури створення, оновлення та видалення. Такий підхід дозволяє легко та зручно виконувати CRUD-операції програмним застосункам в клієнт-серверній архітектурі. Коди створення збережених процедур продемонстровано у Додатку Б.

Процедури створення.

1. **CreateUser:**
    - параметри: @Email, @UserName, @HashPassword, @Phone, @Sex, @BirthDate, @UserAvatarPath;
    - опис: створює нового користувача із вказаними персональними даними та зберігає його у таблиці [User].
2. **CreateFundRequest**:
    - параметри: @ID_User, @FundName, @CodeUSR, @FundDescription;
    - опис: реєструє запит на створення волонтерського фонду з вказаним кодом та описом.
3. **RejectCreateFundRequest**:
    - параметри: @ID_RegisterFundRequest;
    - опис: відхиляє запит на створення фонду, змінюючи його статус на "відхилено".

4. **CreateGroupRequest**:
    - параметри: @ID_User, @GroupName, @GroupDescription;
    - опис: створює запит на реєстрацію нової військової групи із зазначеним описом.
5. **CreateVolunteerFund**:
    - параметри: @ID_RegisterFundRequest, @UserID, @FundName, @CodeUSR, @FundDescription;
    - опис: створює волонтерський фонд, додає ініціатора як адміністратора та оновлює статус "схвалено" запиту на реєстрацію фонду.
6. **CreateMilitaryGroup**:
    - параметри: @UserID, @GroupName, @GroupDescription;
    - опис: створює нову військову групу, додає користувача як адміністратора.
7. **CreateFundMember**:
    - параметри: @UserID, @FundID;
    - опис: додає користувача до складу учасників певного волонтерського фонду.
8. **CreateGroupMember**:
    - параметри: @UserID, @GroupID;
    - опис: додає користувача до військової групи без адміністративних прав.
9. **CreateFundImage**:
    - параметри: @ID_Image, @ID_Fund, @FundImagePath;
    - опис: додає зображення до відповідного волонтерського фонду.
10. **CreateGroupImage**:
    - параметри: @ID_Image, @ID_Group, @GroupImagePath;
    - опис: додає зображення до військової групи.

11.CreateProjectImage:

- параметри: @ID_Image, @ID_Project, @ProjectImagePath;
- опис: зберігає зображення, пов’язане з певним проєктом фонду.

12.CreateFundraising:

- параметри: @FundrGoal, @FundrHeader, @FundrDescription, @ID_Group;
- опис: створює нову ініціативу зі збору коштів із вказаною ціллю та описом, прив’язану до конкретної військової групи.

13.CreateMilitaryRequest:

- параметри: @RequestDescription, @ID_Group;
- опис: створює запит від військової групи на отримання певної допомоги.

14.CreateFundProject:

- параметри: @ProjectName, @ProjectDescription, @ID_Fund;
- опис: створює новий волонтерський проєкт.

15.CreateFundrSponsor:

- параметри: @ID_Fundraising, @ID_Fund;
- опис: прив’язує певний фонд як спонсора до збору коштів.

16.CreateCompleteFundr:

- параметри: @ID_Fundraising, @ID_Fund, @FundsRaised;
- опис: фіксує завершення збору коштів із зазначенням дати завершення та зібраної суми.

17.CreateCompleteRequest:

- параметри: @ID_Request, @ID_Fund;
- опис: фіксує закритий військовий запит конкретним фондом.

18.CreateCategory:

- параметри: @ID_Category, @CategoryName, @CategoryDescription;
- опис: створює нову категорію для класифікації запитів чи ініціатив.

19.CreateFundrCategory:

- параметри: @ID_Fundr, @ID_Category;
- опис: додає категорію до ініціативи зі збору коштів.

20.CreateRequestCategory:

- параметри: @ID_Request, @ID_Category;
- опис: додає категорію до запиту військових.

Процедури оновлення.

1. **UpdateUser:**
    - параметри: @ID_User, @Email, @UserName, @HashPassword, @Phone, @Sex, @BirthDate, @UserAvatarPath;
    - опис: оновлює дані користувача за його унікальним ідентифікатором. Значення змінюються лише для тих параметрів, які не є NULL.
2. **UpdateMilitaryGroup:**
    - параметри: @ID_Group, @GroupName, @GroupDescription;
    - опис: оновлює назву та опис військової групи. Якщо параметри не передані, значення не змінюються.
3. **UpdateVolunteerFund:**
    - параметри: @ID_Fund, @FundName, @FundDescription;
    - опис: оновлює назву та опис волонтерського фонду за заданим ідентифікатором.
4. **UpdateGroupImage:**
    - параметри: @ID_Image, @ID_Group, @GroupImagePath;

- опис: Оновлює шлях до зображення, прикріпленого до військової групи.

5. **UpdateFundImage:**
    - параметри: @ID_Image, @ID_Fund, @FundImagePath;
    - опис: оновлює шлях до зображення, прикріпленого до волонтерського фонду.
6. **UpdateProjectImage:**
    - параметри: @ID_Image, @ID_Project, @ProjectImagePath;
    - опис: оновлює шлях до зображення, прикріпленого до проєкту фонду.
7. **UpdateFundraising:**
    - параметри: @ID_Fundraising, @FundrHeader, @FundrDescription, @FundrImagePath;
    - опис: оновлює заголовок, опис та шлях до зображення ініціативи збору коштів.
8. **UpdateRequest:**
    - параметри: @ID_Request, @RequestHeader, @RequestDescription, @RequestImagePath;
    - опис: оновлює заголовок, опис та зображення запиту військової групи.
9. **UpdateFundProject:**
    - параметри: @ID_Project, @ProjectName, @ProjectDescription;
    - опис: оновлює назву та опис проєкту, пов’язаного з волонтерським фондом.
10. **UpdateCategory:**
    - параметри: @ID_Category, @CategoryName, @CategoryDescription;

- опис: Оновлює назву та опис категорії, яка використовується для класифікації ініціатив або запитів.

Процедури видалення.

1. **DeleteFundImage:**
    - параметри: @ID_Image, @ID_Fund;
    - опис: видаляє зображення, прикріплене до певного волонтерського фонду за його ідентифікатором та кодом зображення.
2. **DeleteGroupImage:**
    - параметри: @ID_Image, @ID_Group;
    - опис: видаляє зображення, пов’язане з конкретною військовою групою.
3. **DeleteProjectImage:**
    - параметри: @ID_Image, @ID_Project;
    - опис: видаляє зображення, прикріплене до проєкту фонду.
4. **DeleteFundMember:**
    - параметри: @ID_User, @ID_Fund;
    - опис: видаляє користувача з учасників конкретного волонтерського фонду.
5. **DeleteGroupMember:**
    - параметри: @ID_User, @ID_Group;
    - опис: видаляє користувача з військової групи, в якій він був учасником.
6. **DeleteFundrCategory:**
    - параметри: @ID_Fundraising, @ID_Category;
    - опис: видаляє прив’язку категорії до збору коштів.
7. **DeleteRequestCategory:**
    - параметри: @ID_Request, @ID_Category;

- опис: видаляє категорію, прив’язану до військового запиту.

2.3.3 Опис тригерів. Окрім системних тригерів, автоматично заданих при створені бази даних в Microsoft SQL Server, було створено тригер «**trg_AssignDefaultUserRole**» для автоматичного надання ролі «User» при створенні користувача в сутності [User]. Робота тригеру працює за умови існування таблиць [Role], [User] і [UserRole], а також заповнення таблиці [Role] необхідними ролями. Код створення тригеру показано у Додатку Б.

2.3.4 Опис уявлень. Уявлення – це віртуальні таблиці, які не зберігають дані фізично, але формуються як результат виконання SQL-запиту. Вони базуються на одній або кількох таблицях і відображають лише певну частину даних із заданою логікою.

В «Інформаційній системі комунікації між волонтерами та військовими» були сформовані уявлення: **ListOfActiveFundraisings** (список активних зборів коштів), **ListOfCompleteFundraisings** (список закритих зборів коштів), **ListOfActiveRequests** (список активних військових запитів), **ListOfCompleteRequests** (список закритих військових запитів), **ListOfCategories** (список існуючих категорій), та для формування звітно-статистичної інформації волонтерами та військовими – **ListOfFundrWithGroup** (список зборів коштів конкретного військового угруповання), **ReportForGroupRecievedFund** (звіт щодо отриманих коштів відповідним військовим угрупованням за деякий період часу), **ReportForGroupRequests** (звіт щодо створених та закритих військових запитів відповідного військового угруповання за деякий період часу), **ReportForCompletedRequestsByFund** (звіт щодо завершених військових запитів відповідним волонтерським фондом за деякий період часу), **ReportForFundRecievedFunds** (звіт щодо отриманих коштів конкретного волонтерського фонду за деякий період часу). Коди створення уявлень вказано у Додатку Б.

2.3.5 Опис користувачів в БД. Для мінімального користування базою даних достатньо одного **sa** (системного адміністратора), від лиця якого будуть проводитися усі транзакції. Такий підхід не є безпечним з точки зору адміністрування бази даних, проте система забезпечує інші міри безпеки на декількох рівнях ще до звертання до БД, тож одного користувача буде достатньо для її користування.

Для більш надійного та гнучкого адміністрування БД на корпоративному рівні рекомендується створення принаймні 5 ролей із різними правами доступу до певних ресурсів БД: системний адміністратор, волонтер, військовий, волонтерський фонд та військове угруповання.

3 ПРОЄКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Організаційна структура програмного забезпечення.

Організаційна структура ПЗ була зображена на рис.3 у вигляді діаграми пакетів

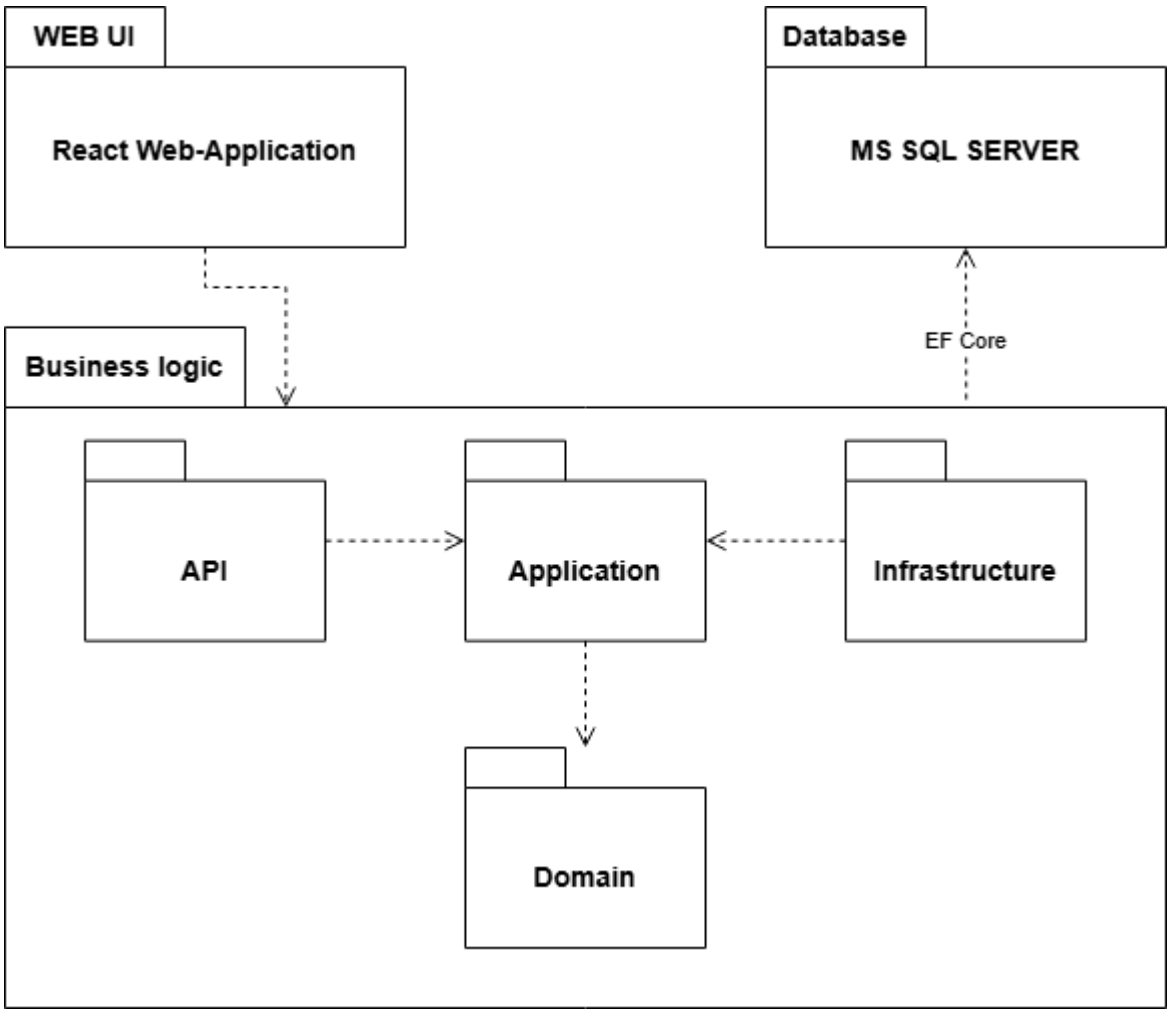

Рис. 4 Діаграма пакетів

Система, що розглядається, побудована за принципами **багаторівневої архітектури**, з чітким розподілом ролей між клієнтським інтерфейсом, серверною логікою та базою даних. Вона складається з трьох пакетів.

1. **Клієнтський застосунок (Web UI).**
   - Тип: односторінковий вебзастосунок (SPA), реалізований з використанням бібліотеки React і строгою типізацією за допомогою TypeScript.
   - Функціональність:
     - забезпечує взаємодію користувача із системою;
     - відправляє HTTP-запити до серверної частини (REST API);
     - відображає дані у вигляді таблиць, форм, повідомлень тощо.
   - Комунікація:
     - працює через браузер;
     - взаємодіє з API за допомогою бібліотеки Axios;
     - дані передаються у форматі JSON.
2. **Серверний застосунок (Business logic).**
   - Побудований на ASP.NET Core Web API.
   - Архітектура: реалізовано за підходом **Clean Architecture**, яка передбачає поділ на такі підпакети.
     1) **Domain**:
        - ядро всієї системи: сутності, інтерфейси, бізнес-правила;
        - не залежить від жодного іншого шару.
     2) **Application**:
        - містить сервісну логіку, DTO, інтерфейси до зовнішніх джерел;
        - залежить лише від Domain.
     3) **Infrastructure**:
        - реалізація доступу до бази даних через **Entity Framework Core**;
        - реалізує інші зовнішні сервіси (збереження файлів, зображень, email тощо);
        - залежить від Application та Domain.

4) **API**:
   - шар взаємодії з клієнтом: контролери, авторизація, маршрутизація;
   - взаємодіє лише з Application.

- Комунікація:
  - Приймає HTTP-запити з клієнтського застосунку.
  - Обробляє їх відповідно до бізнес-логіки і повертає відповіді.

3. База даних (MS Sql Server).

- Тип: реляційна база даних.
- Розміщення: на віддаленому сервері або в хмарі.
- Використовується для збереження даних системи, а також для взаємодії з серверною частиною.
- Доступність: наявний лише один взаємозв’язок із серверним застосунком.

3.2 Вибір інструментарію для створення програмного забезпечення

Для реалізації інформаційної системи було обрано сучасний стек технологій, який забезпечує високу продуктивність, масштабованість та зручність у розробці. Основними компонентами системи є веб-клієнт (frontend), серверна частина (backend) та база даних. Вибір інструментів базувався на їхній сумісності, підтримці корпоративних стандартів, широкій екосистемі та активному розвитку спільноти.

У якості клієнтського веб-застосунку було використано **React** — одну з найпопулярніших JavaScript-бібліотек для побудови односторінкових застосунків (SPA). **React** забезпечує високу швидкодію, модульність інтерфейсу та зручну організацію компонентів. Завдяки віртуальному DOM та ефективному ререндерингу, React дозволяє створювати динамічні інтерфейси, які швидко реагують на дії користувача. У поєднанні з цим було обрано **TypeScript** — надбудову JavaScript з підтримкою статичної типізації, яка

дозволяє знижувати кількість помилок, покращувати автодоповнення у редакторі та забезпечувати кращу читабельність коду в масштабних проєктах.

Для реалізації серверної частини було обрано **ASP.NET Core** — сучасний вебфреймворк від Microsoft, який забезпечує побудову масштабованих RESTful API. Розробка ведеться мовою C#, яка є строго типізованою, об'єктно-орієнтованою та має широку підтримку як середовища Visual Studio, так і серед хмарних сервісів. Вибір **ASP.NET Core** був зумовлений його високою продуктивністю, зручними інструментами для обробки запитів, широкою підтримкою компанії Microsoft, гнучкою системою маршрутизації та підтримкою шаблону **Clean Architecture**.

Відповідно пункту 2.2 використовується **Microsoft SQL Server**, яка добре інтегрується з .NET-екосистемою. Саме сумісність з **SQL Server** стала одним із ключових факторів вибору **ASP.NET Core** для backend-застосунку. Обидва продукти розроблені корпорацією Microsoft і мають повну інтеграцію через **Entity Framework Core** — ORM-бібліотеку для роботи з реляційними даними. Створення та адміністрування баз даних проводитиметься в SSMS – офіційного інструменту від тої ж компанії Microsoft для роботи з **SQL Server**. Вибір цього IDE було зумовлено тим, що він повністю сумісний з обраною СУБД, забезпечує гнучку роботу з таблицями, уявленнями, збереженими процедурами та зв’язками. Крім того, SSMS дозволяє формувати та виконувати складні SQL-запити, створювати діаграми бази даних, проводити аналіз та зручну навігацію по структурі БД.

Таким чином, обраний набір інструментів дозволив реалізувати систему на основі перевірених технологій, з дотриманням принципів багаторівневої архітектури, що підвищує її стабільність, гнучкість і можливість розширення в майбутньому.

3.3 Проектування інтерфейсної частини (Frontend)

Інтерфейсна частина «Інформаційної системи комунікації між волонтерами та військовими» реалізована як односторінковий застосунок (SPA) з використанням бібліотеки **React** та мови **TypeScript**.

3.3.1 Структура інтерфейсу користувача. Проєкт має чітку та логічну структуру, яка розділена на відповідні директорії, компоненти, шаблони. Є одна .html сторінка, яка завантажує єдиний JavaScript-скрипт. Цей скрипт завантажує перший найголовніший компонент **main.tsx** (Додаток Д), у ньому ми описуємо контекстні компоненти-обгортки, які знаходяться над **App.tsx** (Додаток Д) – батьківську компоненту, яка розподіляє весь контент сторінки а також зазначає маршрутизацію сторінок. Також в батьківському компоненті описується **HeaderComponent.tsx** та **Footer.tsx** – компоненти хедера та футера відповідно. Основний контент застосунку обгорнутий в компоненту **Layout.tsx** (Додаток Д). Ця компонента є обгорткою інших компонентів-сторінок з визначеним маршрутом, стилями а також контекстом. Також є компонента **RequireAuth.tsx**, яка є обгорткою для тих захищених сторінок, що мають бути доступними для користувачів із певними ролями.

Основні директорії:

- **src/** – містить вихідний код застосунку;
  - **components/** – багаторазові UI-компоненти, які використовуються на різних сторінках (наприклад HeaderComponent, Footer, App, Main тощо);
  - **pages/** – компоненти, що відповідають окремим маршрутам застосунку;
  - **services/** – модулі для взаємодії з API, зокрема, обгортки над HTTP-запитами;
  - **hooks/** – створені функції-хуки для зручного перевикористання деякої логіки;

- **context/** – компоненту контекстів для керування станами застосунку;
- **types/** – визначення типів TypeScript для забезпечення статичної типізації;
- **assets/** – статичні ресурси, такі як зображення та стилі.

Бібліотеки та інструменти:

- **React**: забезпечує компонентний підхід до побудови інтерфейсу, що дозволяє створювати багаторазові та ізольовані компоненти.
- **TypeScript**: додає статичну типізацію до JavaScript, що підвищує надійність коду та полегшує його підтримку.
- **Vite**: сучасний інструмент для збірки, який забезпечує швидкий запуск та гаряче перезавантаження під час розробки.
- **React Router**: бібліотека для реалізації маршрутизації в SPA, дозволяє створювати навігацію між сторінками без перезавантаження.
- **Axios**: використовується для здійснення HTTP-запитів до серверного API.
- **CSS Modules**: дозволяє писати стилі, які ізольовані на рівні компонентів, запобігаючи конфліктам імен класів.

3.3.2 Взаємодія інтерфейсу користувача з серверною частиною. Клієнтський застосунок взаємодіє з серверною частиною через **RESTful API**. Для цього використовується бібліотека **Axios**, яка дозволяє здійснювати HTTP-запити та обробляти відповіді. Дані передаються у форматі JSON, що є стандартом для веб-застосунків.

3.4 Проектування бізнес-логіки (Backend)

Серверна частина інформаційної системи реалізована з використанням фреймворку **ASP.NET Core** та мови програмування C#, що забезпечує високу продуктивність, масштабованість і підтримку сучасних архітектурних

підходів. Основною архітектурною моделлю обрано **Clean Architecture**, яка сприяє чіткому розділенню відповідальностей між різними шарами застосунку та полегшує його тестування й підтримку.

3.4.1 Структура серверного застосунку. Серверна частина має модульну структуру, яка відповідає принципам **Clean Architecture.**

MVProject.Domain. Містить основні бізнес-об'єкти (сутності) та інтерфейси, що визначають контракти для інших шарів. Цей шар не залежить від жодних зовнішніх бібліотек або фреймворків, що забезпечує його незалежність і можливість повторного використання. Приклад класу сутності показано на рис. 4

```
public partial class FundProject
{
    public Guid ID_Project { get; set; }

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

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

    public DateOnly? ProjectCreateDate { get; set; }

    public Guid? ID_Fund { get; set; }

    public virtual ICollection<FundProjectImage> FundProjectImages { get; set; }
= new List<FundProjectImage>();


    public virtual VolunteerFund? ID_FundNavigation { get; set; }
}
```

Рис. 5 Клас сутності **FundProject**

MVProject.Application. Реалізує бізнес-логіку застосунку, включаючи сервіси, обробники команд і запитів. Наприклад, **Application** відповідальний за формування звітно-статистичної інформації, яка необхідна для аналізу діяльності фонду чи військового угруповання. Алгоритм формування звіту продемонстровано на рис. 5, а фрагмент коду на Додатку Е.

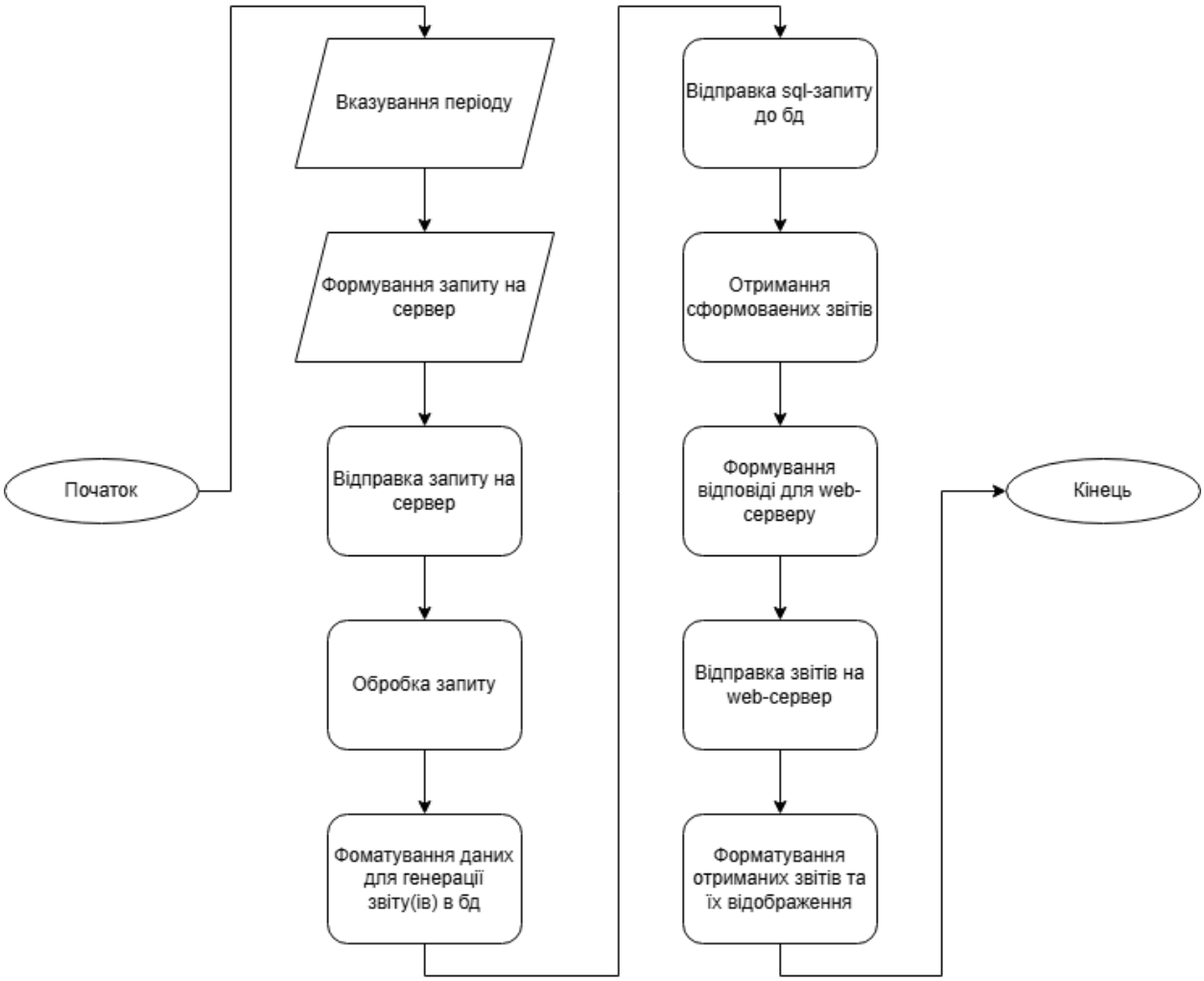

Рис. 6 Блок-схема формування звітів

Шар **Application** залежить лише від шару **Domain** і визначає, як саме повинна працювати система. На рис. 6 продемонстровано фрагмент коду сервісу **UserService**, який відповідає за реєстрацію користувача.

```
public async Task<string> Register(RegisterUserRequest request)
{
    var userExists = await _userRepository.GetByEmailAsync(request.Email);
    if (userExists != null)
        return "Дана пошта вже зареєстрована!";

    var passwordHash = _passwordHasher.HashPassword(request.Password);

    var user = new User
    {
        Email = request.Email,
        UserName = request.UserName,
        HashPassword = passwordHash
    };

    await _userRepository.RegisterUser(user);
    return "Ви успішно зареєстровані!";
}
```

Рис. 7 Функція реєстрації у **UserService**

MVProject.Infrastructure. Містить реалізації інтерфейсів з шару **Application**, зокрема, доступу до бази даних через **Entity Framework Core** та використання репозиторіїв, сервісів для роботи з файлами тощо. Цей шар залежить від зовнішніх бібліотек і фреймворків. На рис. 7 показано функцію **GetUserFunds** у **UserRepository** який за допомогою **DbContext** знаходить усі волонтерські фонди користувача у базі даних. Фрагменти коду репозиторіїв показано у Додатку Е.

```
public async Task<ICollection<FundMember>> GetUserFunds(Guid id)
{
    return await _context.FundMembers
        .AsNoTracking()
        .Include(f => f.ID_FundNavigation)
        .Include(f => f.ID_UserNavigation)
        .Where(f => f.ID_User == id)
        .ToListAsync();
}
```

Рис. 8 Функція знаходження фондів користувача в **UserRepository**

BackEndWebApplication (API). Стартова частина налаштовує веб-сервер, маршрутизацію, залежності та інші аспекти, що пов'язані з запуском застосунку. Тут також реалізовано контролери, які обробляють HTTP-запити та взаємодіють із шаром **Application**. На рис. 8 продемонстрована функція **Logout**, яка відповідальна за вихід користувача з системи. Оскільки

авторизація та сесія користувача реалізована за допомогою **JWT**, необхідно лише видалити токен з **Cookie**. У Додатку Е показано фрагменти коду контролерів API-застосунку.

```
[HttpPost("logout")]
public IActionResult Logout()
{
    Response.Cookies.Delete("tasty-cookies", new CookieOptions
    {
        HttpOnly = true,
        Secure = true,
        SameSite = SameSiteMode.None
    });

    return Ok(new { message = "Ви успішно вийшли" });
}
```

Рис. 9 Функція виходу з систему в **AuthController**

3.4.2 Бібліотеки та інструменти. Використовуються такі бібліотеки:

- **ASP.NET Core** – сучасний веб-фреймворк від Microsoft, який забезпечує побудову високопродуктивних веб-застосунків і API.
- **Microsoft.Entity.Framework.Core** – об'єктно-реляційний мапер (ORM), що дозволяє працювати з базою даних за допомогою об'єктів C#, спрощуючи доступ до даних і їхнє управління.
- **DotNetEnv** – пакет, що дозволяє зчитувати змінні середовища з .env-файлів у C#-застосунках.
- **AutoMapper.Extensions.Microsoft.DependencyInjection** – пакет AutoMapper спрощує спілкування та взаємозв’язок між об’єктами (наприклад, між DTO та сутностями бази даних).
- **Docker** – інструмент для контейнеризації застосунків, що дозволяє створювати ізольовані середовища для розгортання та тестування.

3.4.3 Взаємодія з клієнтською частиною. Серверна частина надає RESTful API, з яким взаємодіє клієнтський застосунок. Обмін даними між клієнтом і сервером здійснюється за допомогою HTTP у форматі JSON з чіткою структурою відповідей.

3.5 Проектування взаємодії з базою даних

Для взаємодії із MS SQL Server у серверному застосунку було використано ORM-фреймворк **Entity Framework Core** (**EF Core**). Його впровадження дозволило спростити роботу з базою даних, зберігши при цьому чітке розмежування відповідальності між шарами застосунку згідно з принципами **Clean Architecture**.

3.5.1 Архітектура взаємодії. Уся взаємодія з базою даних реалізується у шарі **Infrastructure**, який містить:

- реалізації репозиторіїв, що відповідають за доступ до сутностей;
- контекст бази даних (**AppDbContext**), який представляє сесію з базою даних;
- конфігурації сутностей (**EntityTypeConfiguration**) та мапінг зв’язків.

Шар **Application** містить лише абстракції — інтерфейси (IUserRepository, IFundRepository тощо), які реалізуються в **Infrastructure**, що відповідає принципу інверсії залежності (**Dependency Inversion Principle**).

3.5.2 AppDbContext як центральна точка доступу. Уся робота з базою даних ведеться через клас **AppDbContext**, який наслідується від **DbContext**, і є центральною точкою підключення до БД. В цьому класі:

- описано набори сутностей (DbSet<User> Users, DbSet<Fund> Funds тощо);
- реалізовано виклик збережених процедур;
- застосовано Fluent API для конфігурації зв’язків між таблицями.

Контекст налаштовується у **Program.cs** через метод **AddDbContext** із підключенням до рядка з'єднання, який формується із файлу з розширенням «.env» або з файлу appsettings.json.

3.5.3 Моделювання сутностей. Кожна таблиця в базі даних має відповідну клас-модель у **Domain** (наприклад, **User**, **VolunteerFund**, **MilitaryGroup**). Сутності відображають:

- поля таблиці (ID, назва, опис тощо);
- зовнішні ключі (ForeignKey);
- навігаційні властивості (Navigation Properties) для роботи зі зв’язками (один-до-багатьох, багато-до-багатьох).

3.5.4 Репозиторії та сервіси. У шарі **Infrastructure** реалізовано патерн "Репозиторій", що інкапсулює логіку доступу до БД. Наприклад:

- **UserRepository** реалізує **IUserRepository** і містить методи GetUserById, CreateUser, UpdateUser, DeleteUser тощо;
- сервіси в **Application** викликають методи репозиторіїв і не взаємодіють з **EF Core** напряму.

Це дозволяє легко замінити реалізацію доступу до даних у майбутньому, наприклад, на іншу СУБД або Web API, без зміни бізнес-логіки.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Діаграма розгортання

Після зображення основних пакетів системи необхідно показати її фізичну інфраструктуру та взаємозв'язки між основними компонентами. У процесі розробки складних веб-застосунків особливу увагу приділяють архітектурі системи, оскільки вона визначає її масштабованість, стійкість до збоїв та легкість в обслуговуванні. Серед найбільш поширених архітектурних рішень сучасних інформаційних систем можна виділити:

- **Монолітну архітектуру** — уся логіка, інтерфейс і доступ до даних об'єднані в одному застосунку.
- **Клієнт-серверну архітектуру** — логіка поділена на клієнтську (інтерфейс) та серверну (обробка даних).
- **Мікросервісну архітектуру** — система складається з незалежних модулів (сервісів), кожен з яких виконує окрему функцію.
- **Серверлес-моделі** — логіка виконується в безсерверному середовищі (наприклад, AWS Lambda, Azure Functions).

Для реалізації системи було обрано **клієнт-серверну архітектуру** з окремим фізичним розгортанням інтерфейсної та серверної частини, а також бази даних системи. Такий підхід забезпечує кращу гнучкість, масштабованість, а також можливість незалежного оновлення та тестування окремих частин без впливу на всю систему.

На рис. 9 спроектовано діаграма розгортання, яка зображує клієнт-серверну архітектуру з мінімальною кількістю компонентів для стабільної роботи системи.

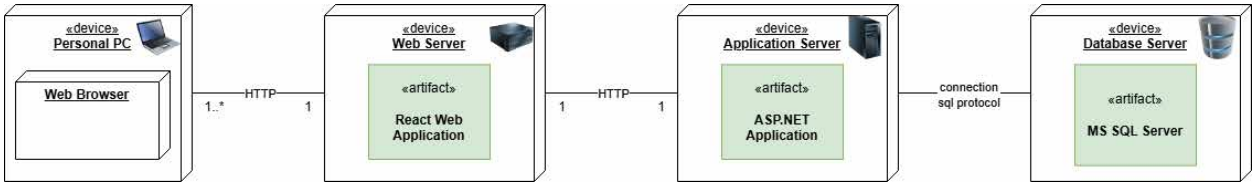

Рис. 10 Діаграма розгортання системи

Основні компоненти:

1. **Personal PC (Клієнтський пристрій).**
    - Тип пристрою: кінцевий пристрій користувача.
    - Компонент: Web Browser.
    - Призначення: через браузер користувач взаємодіє із системою. Застосунок, розгорнутий на Web Server, завантажується у браузер як SPA.
    - Комунікація: надсилає HTTP-запити до Web Server для завантаження React-застосунку.
2. **Web Server.**
    - Компонент: React Web Application.
    - Призначення: сервер статичного вмісту, який обслуговує frontend застосунок. Його завантажують користувачі при відкритті веб-сайту.
    - Комунікація: React-застосунок, який завантажується клієнту у браузері, надсилає HTTP-запити до ASP.NET Application, що розгорнутий на Application Server.
3. **Application Server.**
    - Компонент: ASP.NET Application.
    - Призначення: реалізує бізнес-логіку та API, до якого звертається клієнтський застосунок. Забезпечує обробку запитів, авторизацію, валідацію даних і взаємодію з базою даних.
    - Комунікація: обмін даними відбувається у форматі JSON через REST API. Також з’єднується з базою даних через SQL Connection Protocol.
4. **Database Server.**
    - Компонент: MS SQL Server.

- Призначення: реляційна СУБД, що зберігає всю структуровану інформацію про волонтерів, військових, фонди, угруповання, збори коштів, військові запити тощо.
- Комунікація: серверний застосунок на Application Server використовує SQL-протокол (TSP/IP) для взаємодії з базою через ORM-бібліотеку Entity Framework Core.

4.2 Апаратні та програмні вимоги до системи

4.2.1 Апаратні вимоги. Основні вимоги до пристроїв системи такі.

Апаратні вимоги до пристрою користувача:

- процесор – Одноядерний або двоядерний процесор із тактовою частотою 1.6 GHz і вище;
- оперативна пам’ять – не менше 2Гб;
- диск – не менше 500 МБ вільного простору;
- відеоадаптер – Вбудована графіка з підтримкою HTML5 та WebGL;
- мережеве підключення: стабільне інтернет-з’єднання для зв’язку з сервером по HTTP/HTTPS.

Апаратні вимоги до frontend частини (React):

- процесор: 2-4vCPU 2.4GHz+;
- оперативна пам’ять: не менше 10Гб;
- сховище: SSD/HDD від 10Гб;
- мережеве підключення: Gigabit Ethernet або краще.

Апаратні вимоги до серверної частини (ASP.NET WebAPI):

- процесор: 2-4vCPU 2.4+Ghz;
- оперативна пам’ять: не менше 10Гб;
- сховище: SSD не менше 10Гб під логи, кеш, бекапи;

- операційна система: Windows Server 2022 або Ubuntu Server 22.04 LTS;
- мережеве підключення: Gigabit Ethernet або краще.

Апаратні вимоги до сервера БД (SQL Server):

- процесор: Intel Core i5 / Xeon або потужніший;
- оперативна пам’ять: від 8 ГБ;
- диск: SSD, мінімум 20 ГБ вільного місця;
- операційна система: Windows Server 2022;
- мережева карта: Gigabit Ethernet або краще.

4.2.2 Програмні вимоги. Основні вимоги ПЗ такі.

Програмні вимоги користувача:

- будь-який веб-браузер, який використовує рендерне ядро **WebKit**, **Blink** або **Gecko**.

Програмні вимоги frontend частини (React + TypeScript):

- Node.js – 18+ (LTS);
- NPM / YARN / PNPM, NPM 8+ або YARN 1.22+;
- система збірки – Vite;
- веб-сервер – Nginx / Apache / Caddy / Express;
- Reverse Proxy (опціонально) – Nginx або IIS, якщо серверна та веб-серверна частина розгортаються на одному домені;
- SSL-сертифікат HTTPS – Let’s Encrypt або комерційний;
- система контролю версій – Git для CI/CD або автоматизації оновлень;
- CI/CD (опціонально) – GitHub Actions, GitLab CI.

Функціональні вимоги frontend частини:

- обслуговування index.html для всіх маршрутів (SPA логіка: try_files $uri /index.html;);
- підтримка HTTP/2 або HTTPS-з’єднання;
- роздача статичних файлів з кешуванням та стисненням (gzip / brotli);
- логування звернення (access/error logs);
- наявність підтримки CORS (якщо frontend і backend на різних доменах або портах).

Програмні вимоги до backend частини (ASP.NET Core):

- .NET SDK / Runtime: .NET 8.0;
- ASP.NET Core Hosting Bundle для розгортання на IIS (Windows);
- Веб-сервер – Kestrel, або IIS (Windows), або Nginx (Linux + reverse proxy);
- система контролю версій – Git для CI/CD або деплою;
- Docker для створення образу застосунку;
- CI/CD (опціонально) – GitHub Actions або Azure DevOps;
- сертифікати безпеки SSL – Let’s Encrypt, або Cloudflare або інший сертифікат HTTPS;
- моніторинг / логування – Serilog, Seq, ELK stack.

Функціональні вимоги серверної частини: продемонстровано у Таблиці 2.

Таблиця 2

Категорія	**Вимоги**
REST API	Підтримка CRUD-операцій (Create, Read, Update, Delete) через HTTP.
Формат відповідей	JSON (application/json), з відповідними HTTP-статусами (200, 201, 400…).
Аутентифікація	Підтримка JWT токенів (Bearer Token), розділення Access / Refresh.
Авторизація	Контроль доступу за ролями (Користувач, Адмін).
Обробка помилок	Централізоване логування та видача структурованих помилок (ProblemDetails).
Підтримка CORS	Конфігурація доступу до API з інших доменів (для SPA на React).
Валідація вхідних даних	Анотації [Required], [MaxLength], [EmailAddress] або FluentValidation.
Асинхронність	Усі контролери й репозиторії мають використовувати async/await.
Розширюваність	Чіткий розподіл шарів (Controller → Service → Repository → DB).
Безпека з’єднання	Всі запити повинні бути через HTTPS (TLS 1.2 або вище).

Категорія	Вимоги
Масштабованість	Підтримка горизонтального масштабування (Docker, Load Balancer).
Підтримка логування	Інтеграція Serilog, Seq, або Elastic (з traceId / requestId).
Docker-ready	Працездатність у Docker-контейнері з ENTRYPOINT.

Програмні вимоги до сервера БД (MS Sql Server):

- SQL Server 2022 (Express / Standard / Enterprise);
- конфігурація бази даних: створена база MilitaryAidDb, всі таблиці, збережені процедури, уявлення та користувачі налаштовані відповідно до логіки роботи програми.

4.3 Інсталяційний пакет

Інсталяційний пакет описує, яким чином окремі компоненти інформаційної системи можуть бути розгорнуті в продакшн-середовищі, які існують варіанти хостингу та які сервіси використовуються у розробленій системі. Відповідно пункту 4.1 система складається з трьох основних частин: веб-застосунок (frontend), серверний застосунок (backend) та база даних (в MS SQL Server), кожна з яких потребує окремого розгортання.

4.3.1 Варіанти розгортання Frontend (React Web Application). Веб-застосунки, побудовані за принципом SPA, зазвичай розгортаються як статичні сайти.

Найпопулярніші сервіси для розгортання:

- **Vercel** — найзручніше рішення для React/Next.js застосунків, підтримує автоматичний деплой з GitHub;

- **Netlify** — простий та ефективний сервіс із CI/CD;
- **Firebase Hosting** — частина Google Cloud Platform для хостингу статичних сайтів;
- **GitHub Pages** — безкоштовний сервіс для простих статичних проєктів.

4.3.2 Варіанти розгортання Backend (ASP.NET Web API). Для серверних застосунків на ASP.NET Core доступні такі сервіси:

- **Render** — популярна платформа з підтримкою Docker і .NET Core, з автоматичним деплоєм із GitHub;
- **Azure App Service** — від Microsoft, з глибокою інтеграцією з .NET і SQL Server;
- **Heroku** — зручний для REST API, але має обмежену підтримку C#;
- **VPS-сервери** (наприклад, на **DigitalOcean** або **Hetzner**) — повна свобода розгортання, але потребує ручного налаштування.

4.3.3 Варіанти розгортання бази даних MS SQL Server. Серед найпопулярніших способів розгортання реляційних баз даних:

- **Azure SQL Database** — кероване рішення від Microsoft;
- **Google Cloud SQL** (SQL Server) — хмарна служба баз даних з підтримкою SQL Server;
- **Amazon RDS for SQL Server** — керована база в AWS;
- **Власний сервер або VPS** — повна гнучкість та відповідальність.

4.3.4 Реалізація архітектури системи, що розроблюється. У розробленій інформаційній системі використано сучасний підхід до CI/CD,

при якому всі компоненти автоматично розгортаються із GitHub-репозиторіїв на відповідні хостингові сервіси:

- **Frontend** (веб-застосунок на React TypeScript) було розгорнуто на платформі **Vercel**. Репозиторій, розміщений на GitHub, автоматично збирається та деплоїться при кожному оновленні основної гілки (main/master). Це забезпечує швидке оновлення вебінтерфейсу та зручність у тестуванні змін;
- **Backend** (серверний застосунок ASP.NET Core) розгорнуто на сервісі **Render**, який забезпечує повноцінне середовище для запуску API-застосунків, з підтримкою .NET Core та інтеграцією з GitHub. Це дозволило реалізувати RESTful API для взаємодії з клієнтською частиною;
- **База даних в MS SQL Server** була розгорнута на платформі **Google Cloud Platform** за допомогою керованої служби Cloud SQL. Це забезпечує надійність, масштабованість, автоматичне резервне копіювання та високий рівень доступності даних.

4.4 Тестування системи

Тестування програмного забезпечення — це процес перевірки функціональності, надійності, безпеки та відповідності програмного продукту встановленим вимогам. Його головна мета — визначити, як саме повинна себе поводити певна функціональність системи, виявити помилки на ранніх етапах розробки та забезпечити стабільну роботу системи в умовах реального використання. У Таблиці 3 показано основні типи тестування системи.

Тестування дозволяє:

- перевірити коректність реалізації функціональності;
- гарантувати стабільність при оновленнях;
- зменшити ризики виявлення критичних помилок у продакшені;

- підвищити якість коду та довіру до системи.

Таблиця 3

Тип тестування	Опис
Unit-тестування	Перевірка окремих функцій або методів у ізоляції.
Інтеграційне	Перевірка взаємодії між модулями або компонентами.
End-to-End (E2E)	Повна перевірка системи від інтерфейсу до бази даних.
Функціональне	Перевірка конкретної бізнес-функціональності згідно з вимогами.
Регресійне	Повторне тестування після змін, щоб упевнитися у відсутності нових помилок.
Навантажувальне	Визначення продуктивності системи під тиском великої кількості запитів.

4.4.1 Тестування клієнтської частини (React + TypeScript). React-застосунки зазвичай тестуються з використанням таких технологій:

- **Jest** — фреймворк для unit та інтеграційного тестування.
- **React Testing Library** — бібліотека для тестування компонентів через взаємодію з DOM.
- **Cypress / Playwright** — для end-to-end тестування.

Типи тестування:

- **Unit-тести** компонентів (expect(component).toRender()).
- **Інтеграційні тести** форм, валідації, взаємодії з API.
- **E2E тести** сценаріїв користувача (авторизація, створення збору коштів, військових запитів тощо).

Для тестування інтерфейсної частини було обрано **React Testing Library**. Далі будуть продемонстровані фрагменти кодів тестування деяких компонентів застосунку та їх функціоналу.

Тестування компонента LoginForm. На рис. 10 показано код тестування контенту компонента **LoginForm**. Цей тест перевіряє, чи відображаються поля введення для електронної пошти та пароля, а також кнопка входу.

```
import { render, screen } from "@testing-library/react";
import LoginForm from "./LoginForm";

test("відображає поля введення та кнопку входу", () => {
  render(<LoginForm />);
  expect(screen.getByLabelText(/електронна пошта/i)).toBeInTheDocument();
  expect(screen.getByLabelText(/пароль/i)).toBeInTheDocument();
  expect(screen.getByRole("button", { name: /увійти/i })).toBeInTheDocument();
});
```

Рис. 11 Тестування контенту компонента **LoginForm**

Тестування взаємодії з формою компонента LoginForm. На рис. 11 показано код тестування функціональності форми авторизації. Цей тест перевіряє, чи можна вводити значення в поля форми та чи викликається функція обробки при натисканні кнопки.

```
import { render, screen } from "@testing-library/react";
import userEvent from "@testing-library/user-event";
import LoginForm from "./LoginForm";

test("введення даних та надсилання форми", async () => {
  const handleSubmit = jest.fn();
  render(<LoginForm onSubmit={handleSubmit} />);

  await userEvent.type(
    screen.getByLabelText(/електронна пошта/i),
    "test@example.com"
  );
  await userEvent.type(screen.getByLabelText(/пароль/i), "пароль123");
  await userEvent.click(screen.getByRole("button", { name: /увійти/i }));

  expect(handleSubmit).toHaveBeenCalledWith({
    email: "test@example.com",
    password: "пароль123",
  });
});
```

Рис. 12 Тестування взаємодії з **LoginForm**

Також було використано **Manual** тестування. Це тестування функціональності системи, зручності використання та відповідності вимогам здійснюється вручну і без використання автоматизованих скриптів або спеціалізованих фреймворків. На рис. 12-15 продемонстровано скріншоти реєстрації користувача та оновлення його профілю.

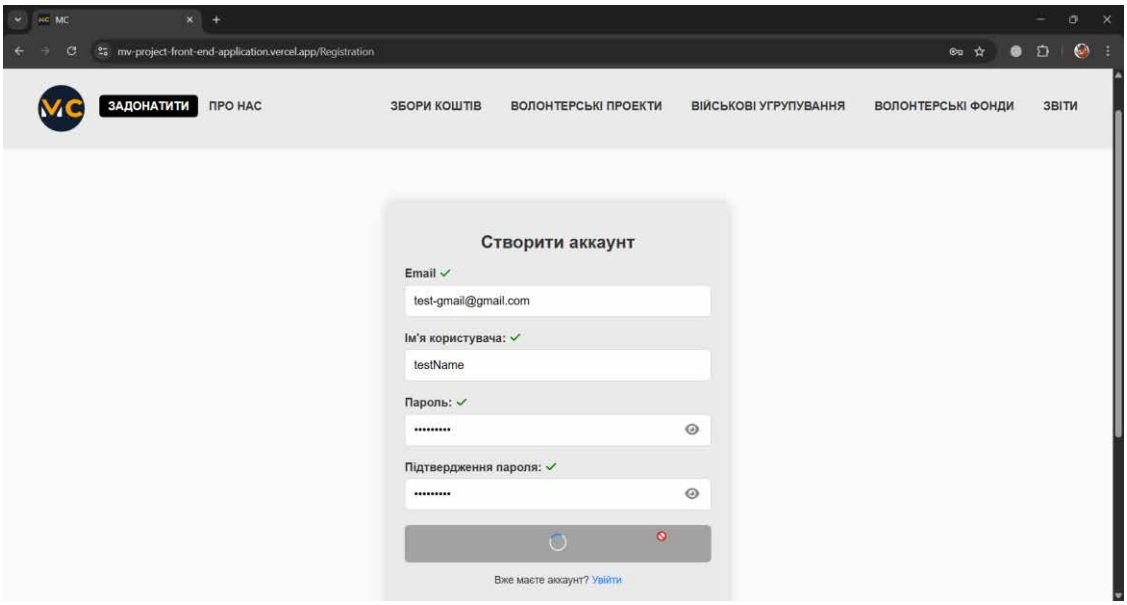

Рис. 13 Сторінка реєстрації користувача

Після натискання на кнопку реєстрації, користувача пересилає на сторінку авторизації з повідомленням про успішну реєстраці.

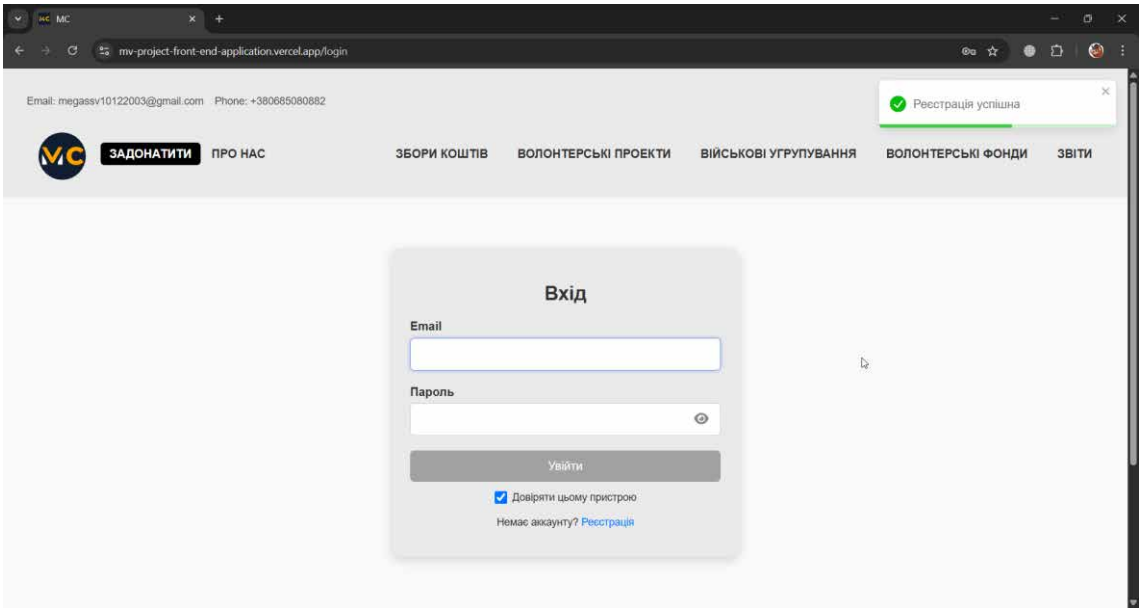

Рис. 14 Повідомлення про успішну реєстрацію користувача

Для оновлення профілю користувача, необхідно зайти на відповідну сторінку, увійти в режим редагування, вказати нові дані та натиснути кнопку «Редагувати»

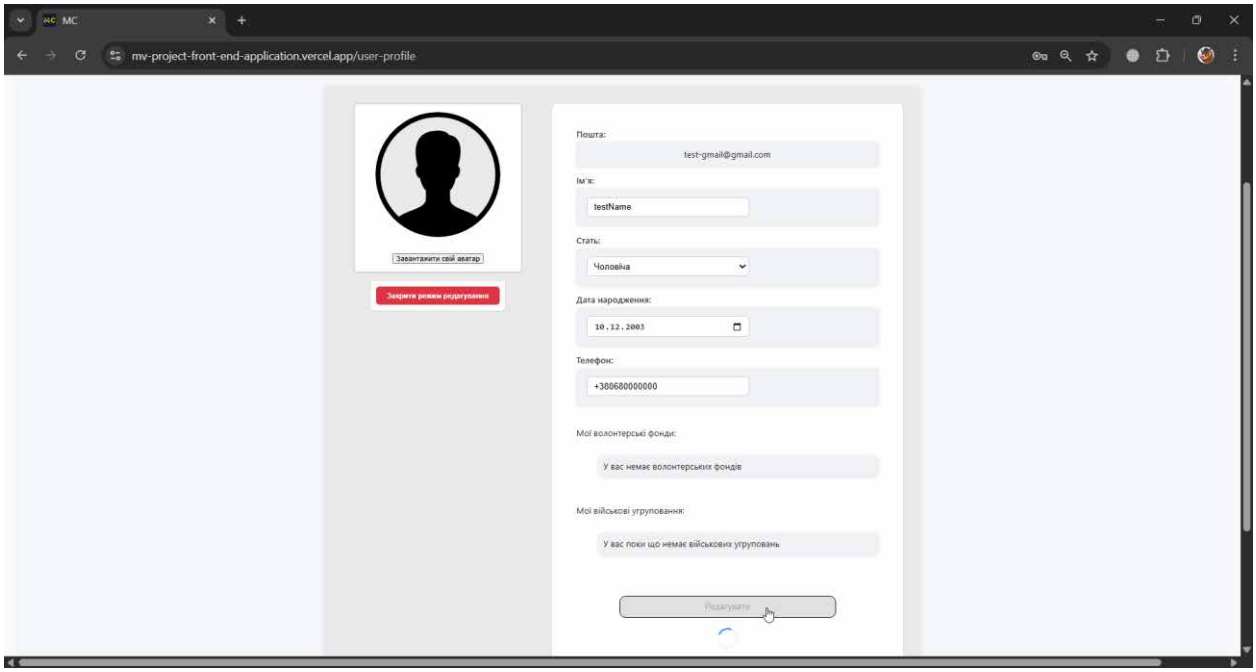

Рис. 15 Сторінка оновлення профілю користувача

Після успішного оновлення отримаємо повідомлення про зелене повідомлення

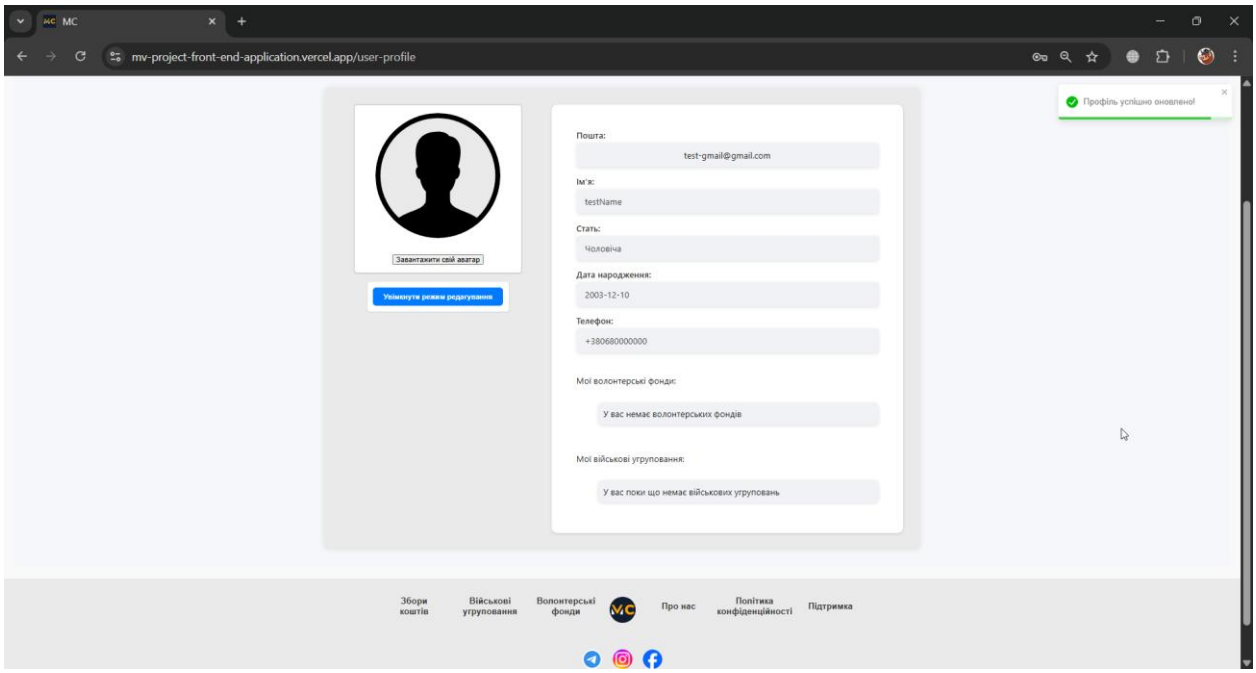

Рис. 16 Повідомлення про успішне оновлення профілю користувача

4.4.2 Тестування серверної частини (ASP.NET Core Web API). Для .NET застосунків використовуються:

- **xUnit / NUnit / MSTest** — популярні тестові фреймворки.
- **Moq** — бібліотека для створення підставних об’єктів (mocks).
- **TestServer** — для інтеграційного тестування API без розгортання сервера.

Типи тестування:

- **Unit-тести** сервісів і логіки (Assert.Equal(expected, actual)).
- **Інтеграційне тестування** контролерів із підключенням до in-memory бази.
- **E2E тестування** авторизації, валідації, обробки помилок.

Для тестування бізнес-логіки на серверному застосунку було обрано **NUnit** як основний інструмент для написання модульних тестів. **NUnit** — це один із найпопулярніших і стабільних фреймворків для тестування в екосистемі .NET, який забезпечує високу гнучкість, підтримку сучасних стандартів і зручний синтаксис.

Чому було обрано саме **NUnit**:

- Система потребує гнучкого та зрозумілого механізму тестування бізнес-логіки та контролерів.
- **NUnit** дозволяє швидко писати асинхронні та параметризовані тести, що важливо для API, які працюють з даними через **EF Core**.
- Розробники мають можливість масштабувати тестовий набір у майбутньому без зміни інфраструктури.
- Широке використання **NUnit** у корпоративних проєктах зробило його надійним вибором з точки зору підтримки, стабільності та документації.

На рис. 12 показано фрагмент коду тестування функціоналу **UserService**. Цей тест перевіряє, що метод **CreateUser** контролера успішно створює нового користувача і повертає HTTP 201 Created з очікуваним результатом.

```
[TestFixture]
public class UserCreationTests
{
    [Test]
    public async Task CreateUser_ReturnsCreatedAtAction()
    {
        // Arrange
        var newUser = new User { UserName = "NewUser" };
        var mockService = new Mock<IUserService>();
        mockService.Setup(s => s.CreateUserAsync(newUser))
                   .ReturnsAsync(newUser);

        var controller = new UserController(mockService.Object);

        // Act
        var result = await controller.CreateUser(newUser);

        // Assert
        Assert.IsInstanceOf<CreatedAtActionResult>(result);
        var created = result as CreatedAtActionResult;
        Assert.That(((User)created!.Value!).UserName, Is.EqualTo("NewUser"));
    }
}
```

Рис. 17 Тестування створення нового користувача

На рис 13 показано фрагмент коду тестування функціоналу авторизації користувача, а саме, генерація JWT. Цей тест перевіряє коректність генерації JWT-токена методом **GenerateAccessToken**. Він дозволяє перевірити бізнес-логіку генерації JWT-токенів без зовнішньої залежності, що особливо важливо для аутентифікації та авторизації в системі.

```
[TestFixture]
public class JwtTokenTests
{
    [SetUp]
    public void Setup()
    {
        Environment.SetEnvironmentVariable("JWT_KEY",
"supersecretkey1234567890");
        Environment.SetEnvironmentVariable("JWT_ISSUER", "TestIssuer");
        Environment.SetEnvironmentVariable("JWT_AUDIENCE", "TestAudience");
        Environment.SetEnvironmentVariable("JWT_EXPIRES", "60");
    }

    [Test]
    public void GenerateAccessToken_ShouldContainCorrectClaims()
    {
        // Arrange
        var user = new User
        {
            ID_User = Guid.NewGuid(),
            Email = "test@example.com",
            UserName = "TestUser",
            ID_Roles = new List<Role>
            {
                new Role { ID_Role = 1000 }
            }
        };

        var jwtService = new JwtService();

        // Act
        var tokenString = jwtService.GenerateAccessToken(user);
        var handler = new JwtSecurityTokenHandler();
        var token = handler.ReadJwtToken(tokenString);

        // Assert
        Assert.That(token.Issuer, Is.EqualTo("TestIssuer"));
        Assert.That(token.Audiences, Does.Contain("TestAudience"));
        Assert.That(token.Claims, Has.Some.Matches<Claim>(c =>
            c.Type == ClaimTypes.Email && c.Value == user.Email));
        Assert.That(token.Claims, Has.Some.Matches<Claim>(c =>
            c.Type == ClaimTypes.Name && c.Value == user.UserName));
        Assert.That(token.Claims, Has.Some.Matches<Claim>(c =>
            c.Type == ClaimTypes.NameIdentifier && c.Value ==
user.ID_User.ToString()));
        Assert.That(token.Claims,
Has.Exactly(user.ID_Roles.Count).Matches<Claim>(c =>
            c.Type == ClaimTypes.Role));
    }
}
```

Рис. 18 Тестування коректної генерації JWT

4.4.3 Тестування баз даних (MS SQL Server). Тестування бази даних фокусується на:

- перевірці збережених процедур (EXEC + очікуваний результат);
- тестуванні пакетів міграцій / seed-даних;

- перевірці обмежень цілісності (foreign keys, unique, check);
- використанні SQL Server Unit Test Projects або ручного тестування через SSMS.

Типи тестів:

- функціональні тести запитів (правильність вибірки / вставки);
- тести на обробку edge case-даних (null, duplicates, delete cascade);
- перевірка тригерів або бізнес-логіки в SQL.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було реалізовано повноцінний цикл створення інформаційної системи — від глибокого аналізу предметної області до практичного впровадження готового рішення.

Ключові досягнення такі.

1. **Комплексне розуміння предметної області та формалізація вимог.**

Було детально досліджено сферу волонтерської діяльності у контексті взаємодії з військовими, виявлено ключові потреби користувачів і визначено проблеми, які потребують автоматизації. Проведений аналіз наявних рішень дозволив сформулювати чітке технічне завдання та створити основні UML-діаграми, що стали основою для подальшого проєктування системи.

2. **Професійне проєктування та реалізація бази даних.**

Розроблена логічна та фізична структура бази даних із врахуванням нормалізації, безпеки доступу (ролі) та підтримки цілісності даних (через тригери та збережені процедури). Вибір **MS SQL Server** обґрунтований надійністю, масштабованістю та підтримкою інтеграції з .NET-стеком.

3. **Архітектурна побудова та реалізація системи за сучасними підходами.**

Було реалізовано розподілену клієнт-серверну архітектуру, яка охоплює frontend (на базі сучасних веб-технологій), backend (на **ASP.NET**) та безпечну інтеграцію з БД. Діаграма пакетів і модульна структура забезпечують підтримуваність та масштабованість проєкту.

4. **Проєктування, інсталяція та тестування програмного продукту.**

Сформовано чіткий план розгортання, описані вимоги до інфраструктури, реалізовано **github-based delivery-процес**. Визначено і впроваджено методи тестування, що охоплюють функціональні й

нефункціональні аспекти для всіх компонентів системи, що свідчить про системний підхід до забезпечення якості.

5. Реальне впровадження та практична цінність системи.

В результаті впровадження система значно спростила облік волонтерської активності, формування зборів коштів та обробку запитів від військових підрозділів. Практичне застосування довело ефективність рішення — зменшено кількість помилок і час реагування на потреби користувачів.

Розроблена інформаційна система комунікації між волонтерами та військовими є повноцінним програмним продуктом, що реалізує сучасні вимоги до архітектури, безпеки, зручності користування і продуктивності. Вона має потенціал для подальшого масштабування і інтеграції в інші соціально-важливі проекти, що підтверджує доцільність її впровадження у реальному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ВОЛОНТЕРСЬКИЙ РУХ В УКРАЇНІ 2014-2019
   URL: http://resource.history.org.ua/cgi-bin/eiu/history.exe?Z21ID=&I21DBN=DOP&P21DBN=EIU&S21STN=1&S21REF=10&S21FMT=eiu_all&C21COM=S&S21CNR=20&S21P01=0&S21P02=0&S21P03=TRN=&S21COLORTERMS=0&S21STR=volonterskyj_rukh_v_ukrajini_2014_2019 (дата звернення: 20.02.2025).
2. Волонтерство.
   URL: http://www.nbuv.gov.ua/node/5963 (дата звернення: 20.02.2025).
3. Волонтерські організації, які зараз працюють в Україні.
   URL: https://finance.ua/ua/saving/volonterskie-organizacii-v-ukraine (дата звернення: 22.02.2025).
4. Інструкція, як будувати UML-діаграми.
   URL: https://dou.ua/forums/topic/40575/ (дата звернення: 26.02.2025).
5. Що таке функціональні вимоги: приклади, визначення, повний посібник.
   URL: https://visuresolutions.com/uk/%D0%B1%D0%BB%D0%BE%D0%B3/%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%96-%D0%B2%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8/ (дата звернення 30.02.2025).
6. База даних. Поняття ER-моделі. Поняття сутностей (entity). Атрибути. Види атрибутів.
   URL: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/ (дата звернення: 01.03.2025).
7. Основи організації баз даних.

URL: https://dglib.nubip.edu.ua/server/api/core/bitstreams/e4e27e98-f9e5-4be4-a3ed-800c041a0044/content (дата звернення: 02.03.2025).

8. СУБД: які бувають, як вибрати.
   URL: https://highload.tech/uk/subd-yaki-buvayut-yak-vibrati/ (дата звернення: 05.03.2025).
9. Windows workloads: SQL Server (Google Cloud documentation).
   URL: https://cloud.google.com/compute/docs/instances/windows#sql_server (дата звернення 06.03.2025).
10. Creating SQL Server VM instances (Google Cloud documentation).
    URL: https://cloud.google.com/compute/docs/instances/sql-server/creating-sql-server-instances (дата звернення: 06.03.2025).
11. Create a stored procedure (Microsoft documentation).
    URL: https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/create-a-stored-procedure?view=sql-server-ver17 (дата звернення 07.03.2025).
12. Create trigger (Microsoft documentation).
    URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver17 (дата звернення: 06.03.2025).
13. Create view (Microsoft documentation).
    URL: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver16 (дата звернення 06.03.2025).
14. Package Diagram Tutorial.
    URL: https://www.lucidchart.com/pages/uml-package-diagram (дата звернення: 12.03.2025).
15. Методології розробки програмного забезпечення.
    URL: https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija (дата звернення: 14.03.2025).
16. Git documentation.
    URL: https://git-scm.com/doc (дата звернення: 15.03.2025).

17. Docker documentation.
URL: https://docs.docker.com/ (дата звернення: 16.03.2025).
18. Ключові інструменти та технології Front-end, що використовуються для розробки веб-сайтів.
URL: https://webbookstudio.com/ua/articles/key-tools-and-technologies-used-in-front-end-website-development/ (дата звернення: 16.03.2025).
19. JavaScript documentation (DevDocs).
URL: https://devdocs.io/javascript/ (дата звернення: 17.03.2025).
20. Сучасний посібник з JavaScript.
URL: https://uk.javascript.info/ (дата звернення: 19.03.2025).
21. React (React documentation).
URL: https://react.dev/learn (дата звернення: 22.03.2025).
22. TypeScript Documentation.
URL: https://www.typescriptlang.org/docs/ (дата звернення: 23.03.2025).
23. Що таке backend технологія? Необхідні навички та знання.
URL: https://mc.today/uk/shho-take-backend-rozrobka/ (дата звернення: 25.03.2025).
24. C# .Net: Посібник.
URL: https://programm.top/uk/c-sharp/tutorial/ (дата звернення: 26.03.2025).
25. C# Guide (Microsoft documentation).
URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 28.03.2025).
26. ASP.NET documentation (Microsoft documentation).
URL: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0 (дата звернення: 02.04.2025).
27. ASP.NET Web APIs - Rest APIs with .NET and C# | .NET.
URL: https://dotnet.microsoft.com/en-us/apps/aspnet/apis (дата звернення: 05.04.2025).

28. Entity Framework documentation hub (Microsoft documentation). URL: https://learn.microsoft.com/en-us/ef/ (дата звернення: 08.04.2025).
29. Common web application architectures: Clean architecture (Microsoft documentation). URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення: 09.04.2025).
30. Як порозумітися бекендеру і фронтендеру, обираючи архітектуру. URL: https://dou.ua/lenta/columns/front-back-developer-communication/ (дата звернення: 15.04.2025).
31. Діаграма розгортання: Підручник з UML із ПРИКЛАДОМ. URL: https://www.guru99.com/uk/deployment-diagram-uml-example.html (дата звернення: 20.04.2025).
32. Створення схеми розгортання UML. URL: https://support.microsoft.com/uk-ua/office/%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D1%81%D1%85%D0%B5%D0%BC%D0%B8-%D1%80%D0%BE%D0%B7%D0%B3%D0%BE%D1%80%D1%82%D0%B0%D0%BD%D0%BD%D1%8F-uml-60811688-d811-4387-9715-c1f02f30b125 (дата звернення: 20.04.2025).
33. Визначення вимог до програмних систем. URL: https://elib.lntu.edu.ua/sites/default/files/elib_upload/%D0%95%D0%9D%D0%9F%20%D0%86%D0%9F%D0%97%2019.04.16%20%D0%A8%D0%BE%D0%BB%D0%BE%D0%BC%20%D0%9F%D0%B0%D0%B2%D0%BB%D0%BE%20%D0%A1%D1%82%D0%B5%D0%BF%D0%B0%D0%BD%D0%BE%D0%B2%D0%B8%D1%87/page9.html (дата звернення: 21.04.2025).
34. Що таке deploy і навіщо він потрібен.

URL: https://foxminded.ua/deploi-tse/ (дата звернення: 22.04.2025).

35. Як розгорнути React додаток.
URL: https://blog.ithillel.ua/articles/how-to-deploy-a-react-app (дата звернення: 23.04.2025).

36. Host and deploy ASP.NET Core.
URL: https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/?view=aspnetcore-9.0 (дата звернення: 24.04.2025).

37. Тестування програмного забезпечення: типи, види та застосування.
URL: https://foxminded.ua/testuvannia-prohramnoho-zabezpechennia/ (дата звернення: 30.04.2025).

38. Тестування React додатків.
URL: https://jestjs.io/uk/docs/tutorial-react (дата звернення: 02.05.2025).

39. Testing ASP.NET Core services and web apps.
URL: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/test-aspnet-core-services-web-apps (дата звернення: 05.05.2025).

40. Підручник з тестування бази даних.
URL: https://www.guru99.com/uk/data-testing.html (дата звернення: 07.05.2025).