Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Проектування та розробка веборієнтованої системи моніторингу та розподілу матеріально-технічних ресурсів волонтерської організації

Виконав(ла): студент(ка) IV курсу, групи СП-43
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

		Петрик О.М.
	(підпис)	(прізвище та ініціали)
Керівник		Мудрик І.Я.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М.Р
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Петрику Олександру Миколайовичу

1. Тема роботи Проєктування та розробка веборієнтованої системи моніторингу то розподілу матеріально-технічних ресурсів волонтерської організації

Керівник роботи Мудрик Іван Ярославович док. Філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від « 06 » квітня 2026 року № _____
2. Термін подання студентом роботи 22.06.2026
3. Вихідні дані до роботи Технічне завдання на розробку веборієнтованої системи розподілу матеріально-трехнічних ресурсів, наукова та навчальна література
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ.
1 Аналіз вимог до системи.
2. Проєктування та розробка системи
3. Тестування системи.
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги .5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку.
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	к.т.н., доц. Мариненко С.Ю.		
Нормоконтроль	Стоянов Ю. М. к.т.н., доц. каф. ПІ		

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Розробка технічного завдання, аналіз предметної області та збір вихідних даних логістики волонтерського хабу		
2.	Робота над першим розділом «Аналіз вимог до системи»		
3.	Робота над другим розділом «Проєктування та розробка системи»		
4.	Робота над третім розділом «Тестування, впровадження та підтримка»		
5.	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»		
6.	Оформлення пояснювальної записки і графічного матеріалу		
7.	Перевірка на академічний плагіат, перевірка керівником та консультантами		
8.	Попередній захист кваліфікаційної роботи бакалавра		
9.	Захист кваліфікаційної роботи бакалавра		

Студент _____ Петрик О. М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мудрик І.Я.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Петрик Олександр Миколайович, Проєктування та розробка веборієнтованої системи моніторингу та розподілу матеріально-технічних ресурсів волонтерської організації. – сторінок – 68, джерел – 24, рисунків – 12, додатків – 9 .

Ключові слова: інформаційна система, волонтерська логістика, складський облік, алгоритм FEFO, патерн MVC, Apache JMeter.

У даній кваліфікаційній роботі проведено розробку та реалізацію веборієнтованої системи для автоматизації процесів моніторингу та розподілу матеріально-технічних ресурсів волонтерської організації. Метою роботи є створення системи, що відповідає сучасним вимогам, та надає користувачам керувати складськими залишками, відстежувати терміни придатності, формувати комплексні гуманітарні набори за шаблонами та контролювати стадії виконання волонтерських заявок . Реалізація супроводжувалась з використанням технологій: мови програмування C# , платформи .NET 10 , фреймворку ASP.NET Core , реляційної СУБД MS SQL Server , технології Entity Framework Core , а також серверної розмітки Razor Views. У процесі розробки було проведено аналіз вимог, функціональні та нефункціональні вимоги , спроектовано архітектуру, використано патерни проектування «MVC» і «DTO» , а також спроектовано відповідні UML-діаграми. Було створено інтерфейс користувача , реалізовано URL-маршрути та представлення до них, щоб забезпечити можливість взаємодії користувача з системою. Перевірка функціональності та показників швидкодії супроводжувалась модульним , автоматизованим UI-тестуванням та навантажувальним тестуванням продуктивності в середовищі Apache JMeter.

Розроблена система демонструє доцільність створення спеціалізованих логістичних платформ у сфері благодійної та гуманітарної діяльності. Вона є перспективною для подальшого розвитку: впровадження нових аналітичних модулів, інтеграції з іншими зовнішніми сервісами доставки тощо. Проєкт виконано з дотриманням сучасних стандартів програмної інженерії, що забезпечує його актуальність та конкурентоспроможність

ABSTRACT

Petryk Oleksandr, Design and development of a web-oriented system for monitoring and distribution of material and technical resources of a volunteer organization. – pages – 68, sources – 24, figures – 12, appendices – 9.

Keywords: information system, volunteer logistics, warehouse accounting, FEFO algorithm, MVC pattern, Apache JMeter.

In this qualification work, the development and implementation of a web-oriented system for automating the processes of monitoring and distributing material and technical resources of a volunteer organization was carried out. The aim of the work is to create a system that meets modern requirements and enables users to manage warehouse stocks, track expiration dates, generate complex humanitarian kits based on templates, and monitor the stages of volunteer requests fulfillment. The implementation was accompanied by the use of technologies: C# programming language, .NET 10 platform, ASP.NET Core framework, MS SQL Server relational DBMS, Entity Framework Core technology, as well as Razor Views server-side markup.

In the course of development, an analysis of requirements was carried out, functional and non-functional requirements were identified, the architecture was designed, the "MVC" and "DTO" design patterns were applied, and the corresponding UML diagrams were designed. A user interface was created, URL routes and views for them were implemented to enable user interaction with the system. Functionality and performance verification was accompanied by modular testing, automated UI testing, and performance load testing in the Apache JMeter environment.

The developed system demonstrates the feasibility of creating specialized logistical platforms in the field of charitable and humanitarian activities. It is promising for further development: introducing new analytical modules, integration with other external delivery services, etc. The project was implemented in compliance with modern software engineering standards, which ensures its relevance and competitiveness.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

МТР – матеріально-технічні ресурси.

ERP (Enterprise Resource Planning) – система планування та управління ресурсами підприємства.

RBAC (Role-Based Access Control) – рольова модель розмежування та контролю доступу користувачів.

MVC (Model-View-Controller) – архітектурний патерн розділення бізнес-логіки, інтерфейсу та керування.

HTML (HyperText Markup Language) – стандартизована мова розмітки вебсторінок та документів.

DTO (Data Transfer Object) – об'єкт-контейнер строго типізованої передачі та транспортування даних між шарами системи.

SSMS (SQL Server Management Studio) – інтегроване середовище адміністрування та конфігурування СУБД MS SQL Server.

FEFO (First Expired, First Out) – логістичний алгоритм відвантаження товарів, що вимагає першочергового вилучення партій із найближчим терміном придатності.

CRUD (Create, Read, Update, Delete) – базовий набір операцій управління сутностями (створення, читання, оновлення, вилучення).

SQL (Structured Query Language) – декларативна мова структурованих запитів до реляційних баз даних.

UI (User Interface) – графічний користувацький інтерфейс взаємодії людини з програмним комплексом.

E2E (End-to-End Testing) – метод наскрізного тестування програмного продукту, що повністю імітує реальні сценарії користувача.

SSL / HTTPS – розширювані криптографічні протоколи захищеної та шифрованої передачі мережевого трафіку

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний стан суспільно-політичного життя в Україні зумовив стрімке зростання ролі волонтерського руху. Волонтерські організації сьогодні виступають ключовою ланкою у забезпеченні потреб як цивільного населення, так і військових підрозділів, оперуючи величезними обсягами матеріально-технічних ресурсів. Управління такими ресурсами в умовах високої динаміки надходжень та запитів вимагає від волонтерських хабів виключної точності та швидкодії логістичних процесів.

Традиційні підходи до обліку, що базуються на використанні паперових носіїв або загальних офісних пакетів наприклад, Microsoft Excel, у сучасних реаліях демонструють свою неефективність. Основною проблемою є відсутність централізованого моніторингу в реальному часі, що призводить до дублювання даних, виникнення помилок «людського фактору» та несвоєчасного розподілу критично важливих вантажів. Окрему складність становить процес комплектування гуманітарної допомоги, коли одиничні ресурси необхідно розподіляти за складними шаблонами залежно від специфіки запиту.

Впровадження спеціалізованої веб-орієнтованої системи дозволяє автоматизувати моніторинг залишків, оптимізувати процеси розподілу та забезпечити прозорість діяльності волонтерської організації перед донорами та громадськістю. Таким чином, розробка програмного забезпечення для автоматизації моніторингу та розподілу ресурсів волонтерських організацій є актуальним завданням, що має пряме практичне значення для підвищення обороноздатності та соціальної стійкості держави.

Мета і задачі дослідження**.** Метою кваліфікаційної роботи є проектування та розробка веб-орієнтованої інформаційної системи для автоматизації процесів моніторингу, обліку та розподілу матеріально-технічних ресурсів волонтерської організації. Для досягнення поставленої мети було вирішено наступні задачі:

- Проаналізувати предметну область та існуючі методи управління ресурсами в умовах волонтерської діяльності.

- Провести порівняльний аналіз аналогічних програмних продуктів та обґрунтувати вибір технологічного стеку для реалізації проекту.
- Сформулювати функціональні та нефункціональні вимоги до системи моніторингу.
- Спроектувати архітектуру програмної системи та розробити модель бази даних для зберігання інформації про ресурси та транзакції.
- Розробити алгоритмічне забезпечення для автоматизованого формування комплексних наборів допомоги на основі шаблонів.
- Реалізувати програмний продукт та провести його тестування.
- Дослідити питання забезпечення безпеки життєдіяльності та охорони праці при роботі з розробленою системою.

Об’єкт дослідження – процеси моніторингу, обліку та розподілу матеріально-технічних ресурсів у волонтерських організаціях.

Предмет дослідження – методи, моделі та програмні засоби розробки веб-орієнтованої системи для автоматизації складської логістики та контролю видачі гуманітарної допомоги.

Практичне значення одержаних результатів полягає у створенні готового до впровадження веборієнтованої системи, яка може бути використана волонтерськими штабами для підвищення ефективності управління запасами, контролю термінів придатності та забезпечення прозорої звітності.

РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО СИСТЕМИ

У даному розділі проведено аналіз предметної області волонтерської логістики та складського обліку матеріально-технічних ресурсів, досліджено існуючі програмні аналоги для автоматизації гуманітарних штабів, а також обґрунтовано вибір сучасного вебтехнологічного стеку, методології розробки Scrum. На основі проведеного аналізу сформовано рольову модель доступу користувачів, а також визначено ключові функціональні та нефункціональні вимоги до проектованої веборієнтованої інформаційної системи.

1.1 Порівняння з існуючими аналогами

У процесі дослідження було розглянуто три основні класи систем, які найчастіше використовуються благодійними та волонтерськими організаціями для обліку матеріально-технічних ресурсів.

Одним з найперших та найпопулярніших інструментів до якого звертаються волонтерські організації на етапі свого формування є електроні таблиці такі як Google Sheets. Це зумовлено їхньою абсолютною доступністю та відсутністю фінансових витрат на старті. На платформі Google Sheets можна швидко організувати базові списки товарів, проте для складних інженерних задач логістики цей інструмент виявляється непридатним.

Рисунок 1.1 – Логотип Google Sheets [1]

Переваги використання Google Sheets:

- Нульова вартість впровадження та утримання. Для роботи потрібен лише безкоштовний обліковий запис, не потрібно орендувати сервери чи купувати ліцензії.
- Низький поріг входження. Практично кожен володіє базовими навичками роботи з таблицями, що нівелює потребу в тривалому навчанні персоналу.
- Гнучкість структури. Користувач може в будь-який момент додати нову колонку, відредагувати колір комірки або вручну змінити дані.
- Недоліки:
- Відсутність реляційної цілісності даних. Таблиці не мають строгої схеми даних. Можна випадково ввести текстове значення замість числової кількості, зробити помилку в назві категорії чи видалити формулу розрахунку, що повністю руйнує точність усього обліку.
- Проблема паралельного доступу. При одночасній роботі кількох десятків учасників можуть виникають затримки синхронізації, що призводить до втрати або затирання інформації.

- Відсутність транзакційної безпеки. У таблицях неможливо реалізувати принцип ACID, через що система дозволяє списувати товари в «від'ємний залишок», створюючи хаос у звітності.
- Складність автоматизації бізнес-процесів. Створення логіки вимагає написання важких макросів або перевантажених формул, які легко зламати при щоденній експлуатації.

Також одним із лідерів на ринку є модульна open-source платформа Odoo ERP. Вона надає професійний модуль Inventory для складського обліку, побудований на базі мови Python та СУБД PostgreSQL.

Рисунок 1.2 – Логотип Odoo ERP[2]

Переваги використання:

- Строгий складський контроль. Система підтримує дворівневу систему записів, що гарантує відстеження переміщення кожної одиниці МТР між складами та зонами зберігання.
- Автоматизація контролю термінів придатності. Вбудовані алгоритми дозволяють налаштувати сповіщення про наближення дати непридатності для медикаментів та харчових продуктів.
- Висока масштабованість. Архітектура системи дозволяє підключати додаткові модулі наприклад, управління персоналом чи аналітику.

Недоліки:

- Надмірна комерційна функціональність. Odoo створена для комерційного бізнесу. Інтерфейс перевантажений термінами: «інвойси», «податкові накладні», «маржинальність», «прайс-листи», «клієнтські ліди». Для волонтерської діяльності більшість цього функціоналу є зайвим і лише сповільнює роботу.
- Висока вартість розгортання та підтримки. Хоча є безкоштовна Community версія, її встановлення, конфігурування під специфіку волонтерського хабу та оренда хмарного сервера вимагають значних фінансових витрат та залучення системних адміністраторів.
- Високий поріг входження та тривале навчання. В Odoo досить складний інтерфейс. Ознайломлення з ним для нового користувача буде коштувати певного часу. В умовах високої плинності кадрів у волонтерських організаціях це призводить до зниження загальної ефективності роботи штабу.

Альтернативою важким ERP-системам є платформа ERPNext, яка позиціонується як легке, монолітне open-source рішення для малого та середнього бізнесу, побудоване на метаданих та фреймворку Frappe.

Рисунок 1.3 – Логотип ERPNext[3]

Переваги ERPNext:

- Повністю відкритий вихідний код. Система не має прихованих платних функцій у базовій версії, що дозволяє безкоштовно використовувати її модулі обліку.
- Сучасний та більш чистий інтерфейс. У порівнянні з класичними ERP-системами, інтерфейс ERPNext є більш гнучким та легким для сприйняття користувачами.
- Надійна реляційна цілісність**.** Система працює на базі реляційних баз даних, що виключає ризик дублювання карток товарів або випадкового видалення критичних логістичних зв'язків.

Недоліки:

- Потреба у глибокій кастомізації. Система не має специфічних інструментів для волонтерської логістики. Якщо її використовувати суто під волонтерські процеси то прийдеться розробляти власні плагіни.
- Складність оновлення після модифікації коду. Якщо внести зміни у внутрішню структуру ERPNext для адаптації під волонтерські процеси, кожне наступне офіційне оновлення системи безпеки від розробників може призвести до критичних збоїв у роботі кастомізованих модулів.

Критичний огляд ринку програмного забезпечення чітко вказує на існування функціонального розриву. З одного боку, існують прості, але архітектурно ненадійні Google Sheets, з іншого – потужні, але надто дорогі, складні та неадаптовані під волонтерські реалії комерційні гіганти Odoo та ERPNext.

Це повністю обґрунтовує доцільність проектування та розробки спеціалізованої веб-орієнтованої інформаційної системи. Вона має поєднати в собі:

- Строгу реляційну безпеку, транзакційність та цілісність даних за рахунок використання ASP.NET Core та Entity Framework Core.
- Простий, мінімалістичний та адаптивний користувацький інтерфейс, час навчання роботі в якому для нового волонтера не перевищуватиме 10 хвилин.
- Вбудовані оптимізовані функції які будуть задовільняти потреби.

1.2 Визначення вимог до системи

Після порівняння вже існуючих конкуретнів на ринку слід сформувати вимоги до проєкту та виділити основні ролі користувачів, які взаємодіятимуть із системою.

Аналіз вимог – це процес визначення очікувань користувачів від нового або оновленого програмного продукту, який гарантує, що вимоги будуть зрозумілі всім зацікавленим сторонам. Це також ретельне вивчення вимог з метою виявлення помилок, прогалин та інших недоліків.[4]

Отже, після дослідження ринку ми можемо сформувати функціональні та нефункціональні вимоги та ролі користувачів веб-застосунку.

Архітектура розроблюваної веб-орієнтованої системи передбачає реалізацію рольової моделі доступу – RBAC із трьома основними акторами:

- Волонтер – базовий користувач системи, який відповідає за комунікацію з отримувачами допомоги та створення заявок.
- Комірник – користувач із розширеними правами, який здійснює управління фізичним складом і верифікацію матеріальних цінностей. Він повністю успадковує функціонал волонтера.
- Адміністратор – користувач із максимальним рівнем привілеїв, який здійснює повне керування системою, конфігурує глобальні параметри та контролює безпеку. Він успадковує функціонал волонтера та комірника.

Вимоги до ролі волонтера:

- Автентифікація в системі: можливість безпечного входу до персонального кабінету за допомогою ідентифікатора та пароля.
- Керування персональним профілем: можливість редагування особистих даних, включаючи завантаження/зміну фотокартки, додавання імені, прізвища та номера телефону.
- Моніторинг складських залишків: можливість перегляду актуального стану складу, каталогу доступних ресурсів та їхнього розподілу за категоріями.

- Конфігурування індивідуальних шаблонів: можливість формування власних персональних шаблонів (комплектів) для швидкого створення повторюваних запитів допомоги.
- Формування заявок на допомогу: можливість створення та надсилання в систему нових запитів на розподіл матеріально-технічних ресурсів.
- Контроль виконання запитів: можливість відстеження поточного статусу та стану формування створеного запиту допомоги в реальному часі.

Вимоги до ролі комірника:

- Реєстрація надходжень: можливість фіксації та внесення в базу даних інформації про нові партії матеріально-технічних ресурсів, що прибувають до хабу від постачальників.
- Партійно-терміновий контроль: можливість обов'язкової фіксації кінцевих термінів придатності МТР безпосередньо під час їхнього прийому та розміщення на складі.
- Утилізація та списання ресурсів: можливість проведення операцій вилучення та списання з системи непридатних, пошкоджених або прострочених матеріальних ресурсів.
- Верифікація та видача вантажів: можливість підтвердження факту фізичного комплектування наборів та безпосередньої видачі ресурсів за заявками, що надходять від волонтерів.

Вимоги до ролі адміністратора:

- Адміністрування користувачів: можливість реєстрації нових облікових записів волонтерів у системі, налаштування їхніх прав, а також видалення у разі потреби.
- Управління глобальними специфікаціями: можливість створення, редагування та архівації базових шаблонів та комплектів допомоги, які використовуються всією організацією.
- Аналітичний моніторинг: доступ до спеціалізованого модуля аналітики, та оцінки загальної ефективності діяльності волонтерського хабу.

Отже, сформувавши вимоги для акторів можем перейти до формування функціональних вимог розроблювальної системи.

Функціональні вимоги визначають конкретні дії та операції, які повинна виконувати система:

- Реєстрація користувачів: Система повинна дозволяти реєструвати нових користувачів із обов'язковим призначенням ролі.
- Автентифікація: Система повинна забезпечувати перевірку облікових даних користувача таких як Email та пароль під час входу та підтримувати стан захищеної сесії.
- Керування профілем: Система повинна надавати авторизованому користувачу інтерфейс для оновлення персональних даних: імені, прізвища, контактного номера телефону та завантаження фотокартки профілю.
- Класифікація: Система повинна забезпечувати можливість створення та видалення категорій ресурсів.
- Контроль критичного залишку: Система повинна відстежувати дефіцитні ресурси на складі.
- Прийом вантажів: Система повинна забезпечувати реєстрацію нових партій ресурсів від постачальників чи донорів із фіксацією кількості та дати надходження.
- Терміновий моніторинг: Для ресурсів із категорій, що потребують контролю якості система повинна вимагати введення кінцевого терміну придатності та автоматично підсвічувати в інтерфейсі партії, цей термін у яких добігає кінця або вже вичерпаний.
- Списання ресурсів: Система повинна дозволяти проводити операції списання непридатних або прострочених партій ресурсів із бази даних.
- Проектування комплектів: Система повинна надавати інструментарій для створення шаблонів комплексних наборів допомоги.

- Специфікація складу: Система повинна дозволяти додавати поодинокі ресурси до складу шаблону із зазначенням їхньої точної кількості, необхідної для збирання рівно одного комплекту.

Тепер перейдемо до формування нефункціональних вимог проєкту.

Нефункціональні вимоги визначають обмеження, технічні рамки та критерії якості розроблюваного ПЗ:

- Швидкодія**:** Час відповіді сервера на стандартні операції читання/запису даних у базу даних не повинен перевищувати 200 мілісекунд.
- Безпека: Передача даних між клієнтом та сервером має здійснюватися суворо за захищеним протоколом HTTPS. Паролі в БД повинні зберігатися виключно у захешованому вигляді.
- Надійність та відмовостійкість: Рівень доступності системи має становити не менше 99.5% режим роботи 24/7. Бекенд повинен містити глобальний обробник виняткових ситуацій.
- Дизайн та інтерфейс: Користувацький інтерфейс має бути легким в розумінні.

Отже, після формування всіх вимог до системи можна побудувати діаграму варіантів використання для розроблювальної системи.

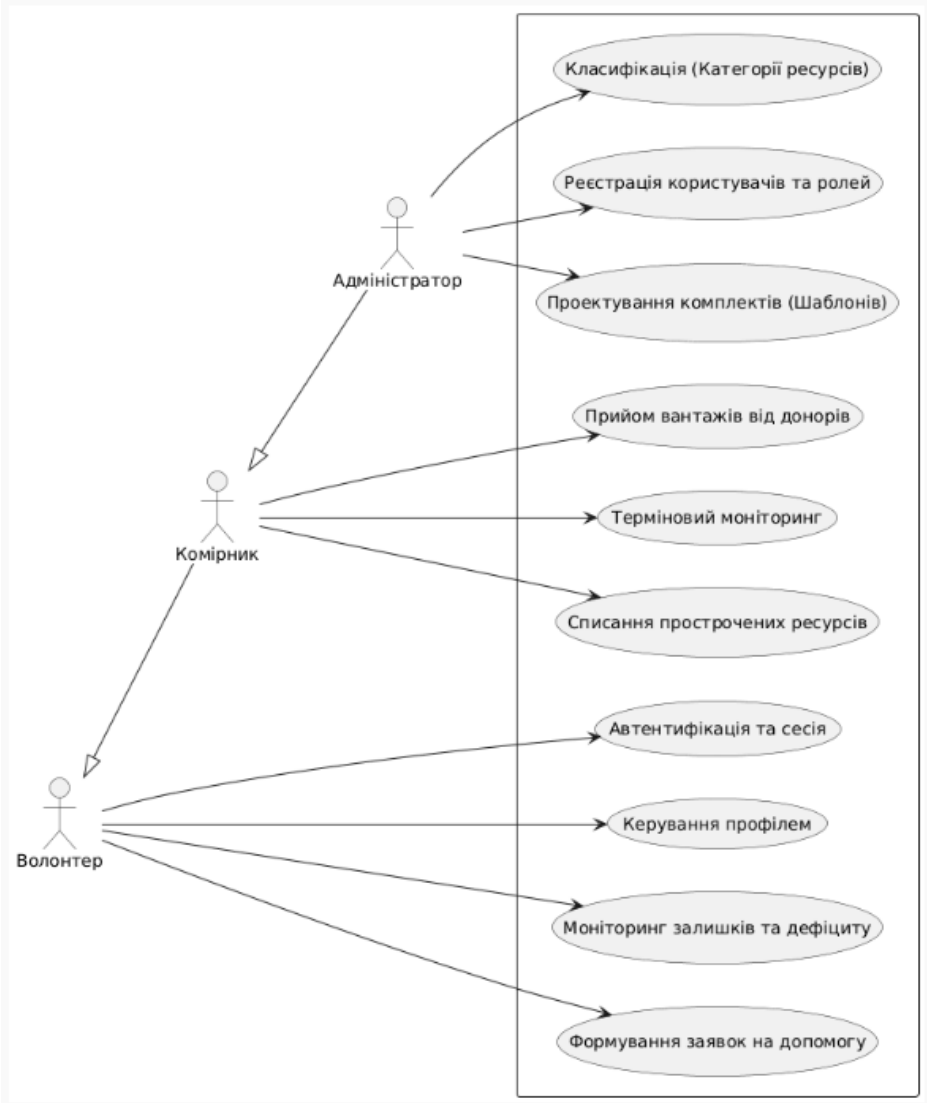


Рисунок 1.4 – Діаграма варіантів використання

1.3 Визначення технологій розробки, інструментів, методології.

Базовою мовою розробки бекенд-частини обрано об'єктно-орієнтовану мову C#. Вона поєднує в собі строгу типізацію, високу швидкодію та потужні інструменти для проектування складної бізнес-логіки. Платформою для розробки виступає .NET 10. Ця платформа є кросплатформною, відкритою та оптимізованою для створення хмарних та веб-застосунків будь-якого рівня складності[5].

Для побудови логіки розподілу ресурсів та взаємодії з клієнтом використовуватиметься фреймворк ASP.NET.

Для надійного та безпечного збереження даних у проєкті обрано реляційну систему управління базами даних MS SQL Server. Вибір саме цієї СУБД зумовлений її високою відмовостійкістю та ідеальною архітектурною сумісністю із платформою .NET.

Взаємодія між кодом на C# та СУБД реалізується за допомогою технології Entity Framework Core (EF Core) із застосуванням підходу Code First. Такий підхід дозволяє проектувати структуру бази даних у вигляді стандартних C#-класів та автоматично генерувати міграції, мінімізуючи написання сирого SQL-коду та гарантуючи реляційну цілісність[5].

Управління життєвим циклом розробки та організація робочого процесу над проєктом базується на гнучких методологіях Agile, а саме – на фреймворку Scrum. Scrum – це не просто набір правил, а легкий фреймворк, який допомагає командам і організаціям створювати цінність за допомогою адаптивних рішень для складних проблем.[6] Вибір цієї методології зумовлений специфікою волонтерської сфери, де вимоги можуть динамічно змінюватися.

Процес розробки вебзастосунку розбивається на окремі часові ітерації – спринти тривалістю у два тижні, протягом яких створюється готовий до використання продукт.

1.4 Висновок до розділу 1

У першому розділі кваліфікаційної роботи було проведено аналіз вимог до системи моніторингу та розподілу матеріально-технічних ресурсів волонтерської організації.

На основі порівняльного аналізу існуючих аналогів було доведено необхідність створення спеціалізованого веб-застосунку, який поєднує простоту інтерфейсу із суворою реляційною та транзакційною безпекою даних. У розділі було описано рольову модель користувачів та сформульовано повний перелік функціональних і нефункціональних вимог до системи.

Також було обґрунтовано вибір технологічного, обрано гнучку методологію розробки Scrum.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

У даному розділі проведено архітектурне проєктування веборієнтованої системи, розроблено логічну та фізичну схеми реляційної бази даних, а також здійснено статичне і динамічне моделювання сценаріїв взаємодії компонентів за допомогою графічних нотацій UML. На основі побудованих моделей та ергономічних вимог до інтерфейсу Razor Views реалізовано програмний комплекс із суворим розподілом бізнес-логіки за патернами MVC і DTO та впровадженням транзакційного алгоритму розподілу залишків за пріоритетним критерієм FEFO.

2.1 Архітектурна модель та проєктування системи

Для реалізації веборієнтованої системи моніторингу та розподілу матеріально-технічних ресурсів на системному рівні було обрано класичну 3-рівневу архітектуру, яка концептуально реалізує модель взаємодії «Клієнт – Сервер – База даних».

3-рівнева архітектура – це архітектура програмного забезпечення, у якій функціональні процеси розділені на три незалежні рівні. Така структура забезпечує високу гнучкість, оскільки кожен рівень можна розробляти, оновлювати та масштабувати окремо.[7] Архітектурна модель передбачає поділ інформаційної системи на три ізольовані рівні, кожен з яких має чітко визначену зону відповідальності та взаємодіє з іншими рівнями через стандартизовані мережеві протоколи:

1. Клієнт – це графічний інтерфейс користувача та шар застосунку, де відбувається вся взаємодія з кінцевим клієнтом. Основне призначення цього шару – збір інформації від користувача та її відображення на екрані.[7] Шар функціонує у веббраузері клієнта, відповідає за первинну перевірку коректності введення даних на стороні клієнта і відправку HTTP/HTTPS-запитів до сервера.

2. Сервер – це шар застосунку який витягує інформацію з рівня представлення, обробляє її за допомогою певних бізнес-правил та керує

переміщенням даних між рівнем представлення та рівнем даних. Простіше кажучи, це мозок усього вебдодатка.[7]

3. База даних або рівень збереження даних – це рівень який складається з серверів баз даних, де зберігається та за потреби витягується вся необхідна інформація. Цей шар забезпечує ізоляцію та безпеку даних, оскільки рівень представлення не може взаємодіяти з ним безпосередньо.[7] В вебзастосунку рівень представлений реляційною СУБД MS SQL Server.

Для розв'язання задачі раціонального поділу внутрішнього коду середньої ланки та усунення надлишкової пов'язаності програмних компонентів було застосовано логічну багатошарову архітектуру розробки.

N-рівнева або багатошарова архітектура є перевіреним архітектурним патерном програмного забезпечення, в якому всі функціональні процеси та компоненти системи суворо розподілені між кількома автономними, незалежними рівнями.[12] Основний інженерний принцип такої організації полягає в тому, що кожен рівень має власну зону відповідальності, є повністю ізольованим і надає сервіси для вищих шарів, що дозволяє розробникам змінювати, масштабувати або оновлювати будь-який ярус окремо, не зачіпаючи роботу решти системи.[12]

Впровадження N-tier підходу надає вагомі переваги для розроблюваної платформи: воно гарантує високу гнучкість управління кодом, суттєво спрощує масштабованість при зростанні обсягів даних, полегшує повторне використання окремих модулів та підвищує загальний рівень кібербезпеки завдяки ізоляції критичних даних від зовнішнього клієнтського інтерфейсу.[12] Згідно з обраною інженерною стратегією, середню ланку розроблювальної системи було декомпоновано на п’ять окремих архітектурних шарів, де кожен елемент відповідає за визначений етап обробки інформації:

1. View Layer: Клієнт-орієнтований прошарок системи, реалізований на базі серверної технології розмітки Razor. Цей пакет повністю відокремлений від прямого доступу до бази даних. Його єдиним обов'язком є візуалізація графічного інтерфейсу, відображення динамічних таблиць складських залишків, логістичних

звітів та збір даних із вебформ для формування вихідних HTTP/HTTPS-запитів користувача.

2. Transport Layer: Шар, що складається з об’єктів передачі даних. Класи DTO є пасивними контейнерами без бізнес-логіки. Впровадження цього рівня дозволяє повністю відокремити внутрішню структуру реляційних моделей від клієнтської сторони, оперувати лише необхідними масивами інформації, оптимізувати мережевий трафік та захистити систему від критичних вразливостей, пов'язаних із несанкціонованою модифікацією системних полів.

3. Controller Layer: Керуючий модуль системи, що виступає центральним координатором патерну MVC. Контролери перехоплюють HTTP-запити від рівня представлення, здійснюють первинну декларативну перевірку валідності отриманих транспортних об'єктів та делегують виконання обчислювальних операцій відповідним службам бізнес-логіки.

4. Business Logic Layer: Центральне обчислювальне ядро вебзастосунку, побудоване на принципах слабкої пов'язаності коду за допомогою механізму впровадження залежностей. Шар представлений сервісами та їх абстрактними контрактами. Саме тут зосереджено алгоритми обчислень, аналізу, сортування.

5. Data Layer: Нижчий рівень системи, представлений контекстом Entity Framework Core та класами моделей предметної області. Він забезпечує збереження інформації у базі даних.

Логічна організація програмного коду всередині Controller Layer, Business Logic Layer та Data Layer підпорядковується архітектурному патерну MVC. Це популярний паттерн проектування, за допомогою якого створюється структура проєкту. Його головна мета – розділити бізнес-логіку програми, користувацький інтерфейс та керуючу логіку, щоб зробити код чистішим і простішим для тестування.[8] Використання цього шаблону дозволяє декомпозувати серверний код на три взаємопов'язані компоненти з чітким розділенням відповідальності.

1. Model – це компонент який містить у собі всю бізнес-логіку застосунку, дані та методи для роботи з ними. Який нічого не знає про те, як ці дані будуть відображатися користувачу або які контролери їх викликають.[8] Моделі в

проэкті представлені набором C# класів, які відображаються на таблиці бази даних і відповідають за доменну валідацію та обмеження на введення некоректних значень.

2. View – це компонент який відповідає за те, як дані з моделі будуть виглядати для кінцевого користувача. Це зовнішній вигляд програми, інтерфейс, який рендериться на основі отриманої інформації.[8] В ASP.NET Core для цього використовується технологія Razor, яка генерує чистий HTML-код безпосередньо на сервері. Це зменшує навантаження та забезпечує високу швидкість відображення форм запитів.

3. Controller – це компонент який виступає в ролі сполучної ланки між моделлю та представленням. Він приймає вхідні запити від користувача, обробляє їх, за потреби передає команди моделі для зміни стану даних і вибирає, яке саме представлення повернути користувачу у відповідь.[8]

Для підвищення рівня безпеки та забезпечення чистоти при передачі інформації між компонентами контролерів та представлень додатково впроваджено патерн DTO. Це об'єкт перенесення даних, який використовується суто для транспортування інформації між шарами системи. Він містить лише поля для збереження даних і не несе в собі жодної бізнес-логіки.[9] Використання DTO дозволяє повністю відокремити внутрішню структуру таблиць бази даних MS SQL Server від шару відображення, що передається у веббраузер Клієнта. Це захищає систему від критичних уразливостей, таких як Overposting або Mass Assignment, коли зловмисник може підробити HTTP-запит і змінити системні поля, до яких у нього не повинно бути доступу.[9] Також це дозволяє оптимізувати обсяг мережевого трафіку, передаючи через канали зв'язку лише ті поля, які необхідні для поточного відображення волонтеру в інтерфейсі.

Для демонстрації взаємодії між архітектурними шарами та напрямків потоків даних у межах системи було розроблено діаграму пакетів, яка зображена на рисунку 2.1.

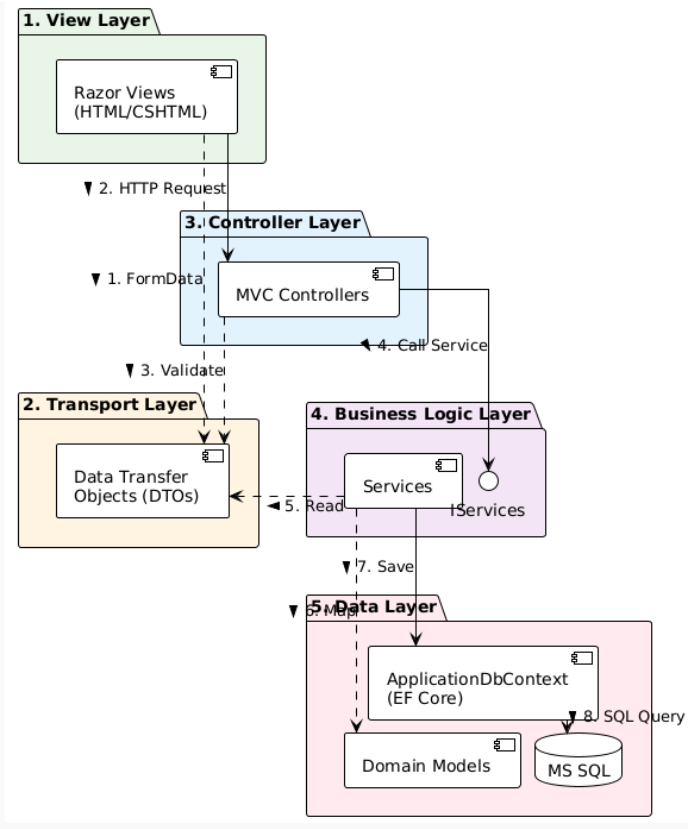


Рисунок 2.1 – Діаграма шарів системи

2.2 Проєктування схеми бази даних

Даний єтап є фундаментальним при проєктуванні розроблювальної системи, оскільки стабільність бізнес-логіки напряму залежить від правильності побудови реляційної структури даних.

Система управління базами даних є спеціалізованим комп'ютерним програмним забезпеченням, за допомогою якого користувачі та розробники можуть систематизовано створювати, вилучати, оновлювати, зберігати та контролювати великі масиви інформації.[10]

На основі визначених функціональних вимог та з використанням технології Entity Framework Core було спроєктовано схему бази даних у СУБД MS SQL Server, яка складається з системних таблиць авторизації та 8 взаємопов'язаних бізнес-сутностей.

Фізичну схему спроєктованої бази даних, згенеровану за допомогою засобів адміністрування СУБД та представлено в додатку А.

Нижче наведено детальну специфікацію основних спроєктованих таблиць із зазначенням типів даних, первинних та зовнішніх ключів відповідно до розробленої схеми.

1. Таблиця AspNetUsers інтегрована зі стандартною екосистемою безпеки ASP.NET Core Identity. Зберігає облікові та профільні дані користувачів:

- Id : унікальний ідентифікатор користувача.
- FirstName / LastName : ім'я та прізвище користувача.
- RegistrationDate: дата реєстрації в системі.
- IsActive: статус облікового запису, активний або заблокований.
- Email: адреса електронної пошти користувача.
- PasswordHash: стійкий криптографічний хеш пароля.
- ProfilePicturePath: шлях до завантаженого фото профілю.

Таблиця пов'язана із допоміжними системними сутностями AspNetUserRoles, AspNetUserClaims, AspNetUserLogins та AspNetUserTokens.

2. Таблиця ResourceCategories служить класифікатором матеріально-технічних ресурсів волонтерського хабу:

- Id: автоінкрементний ідентифікатор категорії.
- Name: назва категорії.
- Description (nvarchar(max)): опис категорії товарів.

3. Таблиця Resources виступає глобальним номенклатурним паспортом матеріального ресурсу:

- Id: унікальний ідентифікатор ресурсу.
- Name: повне найменування ресурсу.
- Unit: одиниця виміру.
- Description: технічний чи логістичний опис товару.
- CategoryId: посилання на таблицю ResourceCategories, реалізує відношення один-до-багатьох.

4. Таблиця ResourceBatches відображає реальну фізичну наявність МТР на складі у розрізі партій надходження:

- Id: унікальний ідентифікатор партії вантажу.
- ResourceId: посилання на паспорт товару в таблиці Resources, зв'язок один-до-багатьох.
- Quantity: поточна доступна кількість одиниць ресурсу в цій партії.
- ReceivedDate: дата прийняття вантажу від постачальника.
- ExpirationDate: кінцевий термін придатності.
- SupplierName: назва донора чи постачальника партії.
- ShelfLocation: точне місце зберігання партії на території складу.

5. Таблиця KitTemplates зберігає заголовки специфікацій для формування комплексних збірних вантажів:

- Id: унікальний ідентифікатор шаблону.
- Name : назва комплекту.
- Description: опис та інструкція щодо збирання пакунка.
- CreatedById: посилання на AspNetUsers для інформаці хто створив шаблон.

6. Таблиця KitItems технічна таблиця зв'язку багато-до-багатьох, яка описує точний склад набору:

- Id: ідентифікатор рядка специфікації.
- KitTemplateId: посилання на цільовий шаблон у KitTemplates.
- ResourceId: посилання на необхідний компонент у Resources.
- RequiredQuantity: точна кількість одиниць цього ресурсу для збору одного готового комплекту.

7. Таблиця VolunteerRequests призначена для ініціалізації, обліку та комплексного контролю кожної окремої заявки на видачу допомоги.

- Id: унікальний номер волонтерської заявки.
- VolunteerId: посилання на волонтера-автора запиту в таблиці AspNetUsers.
- Status: числовий статус обробки документа.

- CreatedAt та CompletedAt: часові мітки створення та успішного закриття заявки.
- Notes: додаткові примітки та логістична інформація про отримувача.

8. Таблиця RequestDetails рядки специфікації, які деталізують, які саме товари та в якій кількості замовлено в заявці:

- Id: ідентифікатор рядка специфікації запиту.
- VolunteerRequestId: посилання на батьківський документ у VolunteerRequests.
- KitTemplateId: посилання на шаблон, якщо волонтер замовляє збірний набір.
- ResourceId: посилання на окремий товар, якщо він видається поштучно.
- Quantity: кількість одиниць товарів або комплектів, необхідна для видачі.

2.3 Побудова UML-діаграм ієрархії класів

Зважаючи на те, що архітектурна декомпозиція системи базується на чіткому розподілі обов'язків, наступним етапом проєктування стало моделювання її внутрішньої статичної структури – тобто визначення конкретних програмних класів, їх функціональних атрибутів, методів та типів взаємозв'язків.

Для розв'язання цієї задачі було використано інструментарій мови моделювання UML, а саме діаграму класів. Діаграма класів є центральним елементом об'єктно-орієнтованого моделювання, який визначає статичну структуру системи в термінах класів предметної області та логічних відношень між ними [11]. Вона відображає внутрішню архітектуру програми, типи даних, принципи інкапсуляції атрибутів, а також типи зв'язків, що є основою для генерації вихідного коду [11].

Конструювання діаграми класів дозволяє створити точні об'єктні специфікації, на базі яких під час виконання програми будуть ініціалізуватися та взаємодіяти екземпляри класів в оперативній пам'яті сервера.

Перед побудовою загальної ієрархії компонентів системи було обґрунтовано вибір шаблонів проєктування. Патерн проєктування є типовим і перевіреним рішенням поширеної архітектурної проблеми, що виникає в процесі конструювання програмного забезпечення [13]. На відміну від готових бібліотек чи специфічних функцій, які можна без змін інтегрувати в код, шаблон не є якимось конкретним програмним фрагментом, а являє собою лише загальний концептуальний принцип і опис алгоритму розв'язання задачі, який необхідно індивідуально адаптувати та реалізувати під потреби конкретної програми [13].

Усі архітектурні шаблони різняться між собою за рівнем деталізації, складності та масштабом охоплення розроблюваної системи [13]. Для проєктування платформи волонтерської логістики було обрано та впроваджено два ключові архітектурні шаблони: MVC та DTO, взаємодія яких підсилена виділеним прошарком бізнес-сервісів, що дозволяє структурувати систему на рівні абстракції та забезпечити чіткий розподіл обов'язків між об'єктами.

Оскільки базовою платформою розробки виступає фреймворк ASP.NET Core, застосування патерну MVC є фундаментальним рішенням, яке розділяє внутрішню логіку системи на три ізольовані складові, де компонент Model презентує рівень даних предметної області, описує структуру інформаційних об'єктів, інкапсулює правила доменної валідації та транслюється ORM-рушієм у реляційні таблиці бази даних MS SQL Server [8]. Компонент View є відповідальним за зовнішній вигляд системи та формування графічного інтерфейсу для кінцевого користувача, де в межах технології Razor представлення динамічно генерують HTML-код на основі даних, отриманих від контролера [8]. Компонент Controller виступає сполучною ланкою архітектури, яка приймає запити від браузера, здійснює маршрутизацію, комунікує з рівнем бізнес-логіки та обирає відповідне представлення для повернення результату користувачу [8].

Для оптимізації обміну інформацією між компонентами патерну MVC та підвищення безпеки застосунку було впроваджено патерн DTO, об'єкти якого є пасивними структурами без власної бізнес-логіки [9]. Вони використовуються для транзиту даних від представлень до контролерів, запобігаючи прямому доступу

користувача до доменних моделей бази даних, що дозволяє усунути ризики надлишкового отримання даних та ізолювати логіку представлення від змін у реляційній структурі [9]. Взаємодія між контролерами та моделями здійснюється через додатково інтегровані інтерфейси сервісів, які приймають об'єкти DTO, виконують логістичні розрахунки, оновлюють стан моделей та фіксують зміни. Для забезпечення розуміння компонентного складу системи, у таблиці 2.1 наведено класифікацію розроблених програмних компонентів відповідно до архітектурних патернів MVC та DTO, вказано їхніх типових представників та визначено характер взаємодії в системі.

Таблиця 2.1 – Класифікація та роль компонентів патернів MVC і DTO

Архітектурний компонент	Приклади	Функціональні обов'язки та характер взаємодії
Models / Entities	ResourceBatch, VolunteerRequest, Resource, KitTemplate	Моделі даних патерну MVC. Визначають структуру об'єктів предметної області, інкапсулюють правила внутрішньої валідації та відображаються на таблиці БД.
Controllers	VolunteerRequestsController, InventoryController, AccountController	Керуюча ланка патерну MVC. Виступають точками входу для HTTP-запитів, ініціюють валідацію вхідних об'єктів DTO та координують роботу представлень.
Views	Index.cshtml, Create.cshtml	Візуальна складова патерну MVC. Відображають користувачеві інтерфейси складського обліку та генерують HTML-сторінки на основі моделей.
DTOs	CreateVolunteerRequestDTO, LoginDTO, CreateResourceDTO	Компоненти патерну DTO. Забезпечують ізольований, безпечний і строго типізований транзит даних від форм представлення до методів обробки контролера.

Продовження таблиці 2.1

Архітектурний компонент	Приклади	Функціональні обов'язки та характер взаємодії
Абстракції та сервіси бізнес-логіки	IVolunteerRequestService, VolunteerRequestService, IInventoryService	Службові компоненти, що забезпечують слабку зв'язність системи через Dependency Injection. Обробляють дані DTO, реалізують бізнес-алгоритми та змінюють стан моделей.

Після детального аналізу взаємодії архітектурних компонентів та розподілу обов'язків у межах патернів MVC і DTO, було спроєктовано репрезентативну UML-діаграму класів, на якій представлено наскрізний ланцюжок взаємодії на прикладі ключового процесуального модуля управління волонтерськими заявками та базового логістичного каталогу ресурсів, див. додаток Б.

2.4 Динамічне моделювання сценаріїв взаємодії

Якщо діаграма класів та фізична схема бази даних відображають статику інформаційної системи, її компонентний склад та зафіксовані типи даних, то для повноцінного аналізу поведінки вебзастосунку необхідно виконати його динамічне моделювання. UML-моделювання поведінкових аспектів дозволяє візуалізувати динаміку процесів, описати логіку та показати, як саме елементи системи взаємодіють між собою під час виконання сценаріїв.[14] Це дає змогу деталізувати часову послідовність обміну повідомленнями між об'єктами різних архітектурних рівнів і зафіксувати алгоритми виконання критично важливих логістичних операцій.

Відповідно до методології проєктування, поведінковий аспект вебзастосунку описано за допомогою двох взаємодоповнюючих графічних нотацій:

1. UML-діаграма послідовності – динамічна модель, що відображає часовий життєвий цикл обробки запиту волонтера, показуючи покрокову

взаємодію між користувацьким інтерфейсом Razor, об'єктами передачі даних DTO, контролером, сервісним шаром та репозиторієм бази даних.

2. UML-діаграма діяльності – поведінкова модель, яка деталізує покроковий алгоритм роботи серверного ядра системи під час перевірки дефіциту матеріально-технічних ресурсів.

2.4.1 Моделювання прецедентів взаємодії на основі UML-діаграми

Діаграма послідовності належить до групи діаграм взаємодії в UML і призначена для моделювання часового виміру обміну повідомленнями між різними об'єктами в межах одного сценарію.[14] У розроблюваній системі вона описує життєвий цикл обробки запиту на створення нової волонтерської заявки VolunteerRequest для розподілу допомоги. Цей процес є наскрізним, оскільки він проходить крізь усі спроєктовані компоненти системи, демонструючи практичну реалізацію принципу розділення відповідальності. Побудована діаграма послідовності для прецеденту створення заявки знаходиться в додатку В.

Аналіз життєвого циклу обробки повідомлень:

1. Ініціалізація процесу: Волонтер через веббраузер взаємодіє з Razor-представленням Create.cshtml. Після заповнення реквізитів отримувача допомоги RecipientInfo та списку матеріальних ресурсів, натискання кнопки відправки генерує асинхронний HTTPS POST запит на сервер.

2. Перехоплення та валідація: Запит приймається контролером VolunteerRequestsController. Серверний рушій .NET Core автоматично мапить вхідні FormData параметри у плоский об'єкт передачі даних CreateVolunteerRequestDTO. Контролер виконує перевірку атрибутів ModelState.IsValid. Якщо валідація успішна, керування передається бізнес-сервісу VolunteerRequestService через виклик асинхронного методу CreateRequestAsync.

3. Транзакційність на рівні даних: Сервіс взаємодіє з ApplicationDbContext, віддаючи команду на старт ізольованої транзакції бази даних (BEGIN TRANSACTION). Це критично для запобігання стану гонитви (Race

Condition), коли кілька користувачів можуть одночасно намагатися списати ті самі обмежені партії товарів.

4. Розгалуження логіки сценарію:

- Сценарій А негативний результат: Якщо внутрішній аналітичний алгоритм сервісу виявляє, що сумарна кількість потрібних ресурсів перевищує наявні залишки у таблиці ResourceBatches, транзакція скасовується ROLLBACK. Контролеру повертається структурована інформація про дефіцит. Користувач отримує зворотний зв'язок у вигляді Razor-сторінки з підсвіченими червоним кольором позиціями, яких не вистачає на складі.
- Сценарій Б Позитивний результат: Якщо перевірка наявності пройшла успішно, сервіс через Entity Framework формує об'єкт моделі VolunteerRequest та набір об'єктів RequestDetail. Одночасно з цим у циклі зменшуються значення поля Quantity у відповідних рядках складських партій ResourceBatches. Зміни фіксуються в БД командами INSERT та UPDATE і закріплюються операцією COMMIT. Контролер отримує статус успішного виконання та виконує перенаправлення користувача на сторінку моніторингу заявок.

2.4.2 Алгоритмічне моделювання логіки розподілу ресурсів

Діаграма діяльності є специфічним типом UML-діаграм поведінки, яка фокусується на відображенні послідовності кроків, умов розгалуження та паралельних процесів у межах певного алгоритму чи бізнес-логіки.[14] За своєю суттю вона є інженерним об'єктно-орієнтованим аналогом класичної блок-схеми, що дозволяє детально зафіксувати логіку функціонування серверного ядра.

Центральною частиною бізнес-логіки системи є алгоритм аналізу складських залишків. Особливістю архітектури є те, що система підтримує роботу як з поодинокими ресурсами, так і зі складними комплексними комплектами, сформованими на основі технологічних шаблонів KitTemplate. Коли волонтер

створює запит на певну кількість комплектів, система повинна розгорнути кожен шаблон, вирахувати сумарну потребу в розрізі кожного окремого базового ресурсу, перевірити наявність фізичних партій на складі з урахуванням термінів придатності і виконати списання.

Процес функціонування цього алгоритму змодельовано за допомогою UML-діаграми діяльності,діаграму дивитись в додатку Д.

Покроковий опис виконання алгоритму розроблюваним серверним ядром:

1. Агрегація та калькуляція потреби: Алгоритм приймає вхідний масив даних DTO. Оскільки користувач міг ввести кілька різних комплектів, що містять однакові базові ресурси система створює динамічний словник resourcesRequirement у пам'яті сервера. Проходячи циклом по рядках, система автоматично розгортає шаблони комплектів, перемножує кількість одиниць набору на кількість компонентів всередині специфікації KitItem та сумує фінальну потребу для кожної унікальної одиниці номенклатури.

2. Валідація залишків та сортування: На другому етапі запускається цикл перевірки реальної наявності ресурсів на складах хабу. Для кожного унікального ідентифікатора ресурсу система через Entity Framework Core виконує оптимізований SQL-запит до таблиці ResourceBatches. важливим рішенням є примусове сортування партій за вихідним критерієм кінцевого терміну придатності в порядку зростання. Це реалізує принцип FEFO, що гарантує автоматичне першочергове вилучення тих ресурсів, термін придатності яких добігає кінця, повністю запобігаючи псуванню вантажів.

3. Логіка списування з партій, вкладений цикл: Якщо сумарний залишок по всіх партіях покриває потребу, запускається алгоритм розподілу списання. Система аналізує найпершу партію: якщо кількість у ній більша або рівна необхідній — потрібна кількість резервується, а цикл завершується. Якщо ж перша партія не здатна повністю покрити потребу, вона списується в нуль, залишок потреби зменшується, а алгоритм переходить до аналізу наступної за терміном придатності партії ресурсу.

4. Етап прийняття рішень транзакційне завершення**:** По завершенню обходу система перевіряє лічильник дефіциту. Наявність хоча б одного дефіцитного елемента повністю блокує виконання операції. База даних виконує швидкий відкат стану, відновлюючи початкові значення залишків, а волонтер отримує деталізований звіт про дефіцит. Якщо дефіциту немає, зміни масово вносяться до таблиць ResourceBatches, VolunteerRequests та RequestDetails в межах єдиної транзакції, забезпечуючи точність складського обліку інформаційної системи.

Завдяки детальному моделюванню динамічних сценаріїв у підрозділі 2.4, ми повністю специфікували алгоритмічну та поведінкову логіку системи. Робота містить усі необхідні UML-артефакти поведінки, що повністю відповідає критеріям.

2.5 Проєктування користувацького інтерфейсу та засобів взаємодії

Проєктування інтерфейсу користувача та засобів його взаємодії з системою є етапом розробки веборієнтованих систем. Головною метою проєктування інтерфейсу користувача є створення максимально інформативного, інтутивно зрозумілого та адаптивного середовища взаємодії, яке мінімізує кількість рутинних кліків для виконання операцій та знижує когнітивне навантаження на людину. У ASP.NET Core, шар відображення повністю реалізується на основі серверної технології генерації розмітки Razor Views.

2.5.1 Архітектурна структура Razor Views та карта навігації

У межах фреймворку ASP.NET Core MVC файли представлень із розширенням .cshtml групуються у фізичні каталоги всередині папки Views, назви яких відповідають іменам контролерів системи, що спрощує трасування коду. Для забезпечення графічної консистентності інтерфейсу та повторного використання коду застосовується майстер-шаблон _Layout.cshtml. Цей глобальний файл

визначає загальну структуру сторінок, містить підключення базових стилів і скриптів , а також описує спільні елементи навігації.

У розроблюваному вебзастосунку реалізовано єдину наскрізну структуру навігації. Управління видимістю та доступністю функціональних модулів здійснюється динамічно на стороні сервера під час рендерингу Razor-сторінки за допомогою перевірки автентифікаційного контексту користувача та його поточних ролей через вбудований інструментарій безпеки (@if (User.IsInRole("RoleName"))). Такий спрощує модифікацію меню та гарантує безпеку, запобігаючи несанкціонованому відображенню посилань. Загальна ієрархія елементів динамічного меню має наступну структуру:

1. Базовий рівень який жоступний усім автентифікованим користувачам після входу в систему:
   - Головна сторінка: стартова інформаційна панель із загальними даними про поточну діяльність волонтерського хабу.
   - Персональний кабінет: інтерфейс перегляду та редагування особистих даних, зміни пароля та перевірки поточного рівня доступу.
2. Модуль Волонтер який додатково активується для користувачів із роллю Volunteer:
   - Моніторинг залишків: табличний інтерфейс перегляду доступної номенклатури ресурсів із можливістю швидкої фільтрації за категоріями.
   - Журнал заявок: реєстр створених поточним користувачем документів із відстеженням станів обробки.
   - Форма створення заявки: інтерактивний конструктор для формування нових запитів на розподіл допомоги.
3. Модуль Комірник який додатково активується для користувачів із роллю Warehouse:
   - Керування складом: реєстр складських партій вантажів з відображенням термінів придатності.
   - Прийом партій: спеціалізована форма додавання нових гуманітарних надходжень від донорів.

4. Модуль Адміністратор який додатково активується для користувачів із роллю Admin:

- Керування користувачами: панель верифікації, призначення ролей та блокування/активації облікових записів.
- Конфігуратор специфікацій: редактор технологічних карт і шаблонів для формування комплексних наборів допомоги.
- Модуль аналітики: інтегрований дашборд оцінки ефективності хабу.

2.5.2 Проєкування інтерфейсів

Для забезпечення ергономіки системи було спроєктовано макети для трьох основних користувацьких інтерфейсів.

1. Інтерфейс створення заявки на допомогу. Ця сторінка є точкою введення даних для DTO-об'єкта. Її спроєктовано у вигляді покрокової форми, що містить такі зони:

- Блок отримувача: Текстове поле великого розміру для введення реквізитів бенефіціара та поле приміток.
- Динамічна таблиця позицій: Користувач може натисканням однієї кнопки додавати нові рядки до специфікації. Кожен рядок містить випадаючий список із пошуком, де можна обрати або поодинокий ресурс, або комплексний шаблон, а також поле введення кількостів.

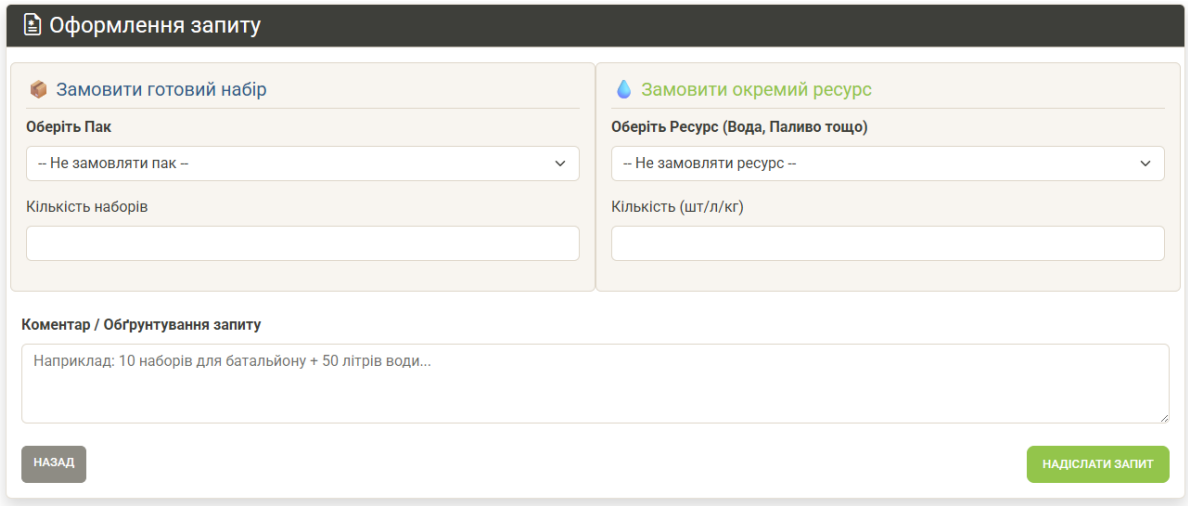


Рисунок 2.2 – Інтерфейс створення заявки на допомогу.

2. Інтерфейс партійного обліку залишків кабінет Комірника. Екранна форма моніторингу складських партій. Вона містить велику таблицю, де кожна стрічка відображає дані моделі:

- Кольорова індикація термінів придатності: Реалізовано логіку динамічного CSS-стилізування. Якщо поле ExpirationDate для партії медикаментів чи їжі добігає кінця, рядок автоматично підсвічується жовтим кольором, якщо партія вже прострочена — червоним. Це дозволяє комірнику візуально ідентифікувати критичні вантажі.
- Елементи швидкої дії: Навпроти кожної партії розміщено кнопки швидкого доступу: «Перемістити» та «Списати».

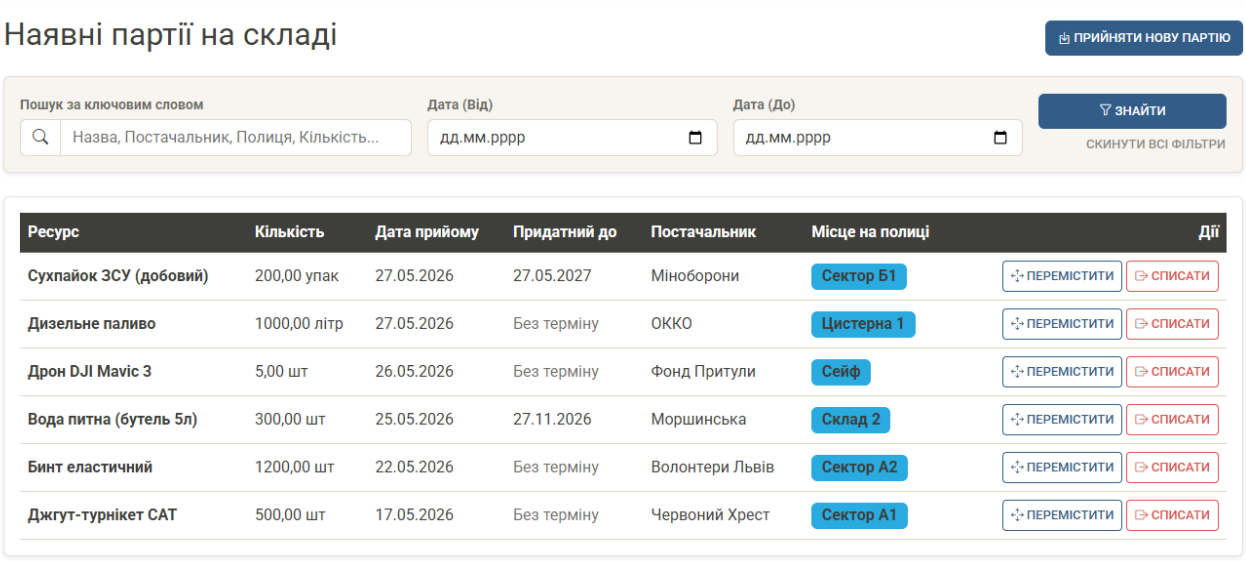


Рисунок 2.3 – Інтерфейс партійного обліку

3. Аналітичний дашборд панель Адміністратора. Екран призначений для вищого керівництва організації й базується на даних класу DashboardStatsDTO. На ньому спроєктовано три інформаційні блоки:

- Картки швидких метрик: Чотири блоки у верхній частині екрана, що великим шрифтом відображають загальну кількість активних волонтерів, обсяг заявок у статусі New, сумарну вагу або кількість виданих вантажів за поточний місяць.
- Діаграма розподілу за категоріями: Графічний віджет, що візуалізує відсоткове співвідношення затребуваності категорій МТР.
- Таблиця критичного дефіциту: Окреме вікно, куди сервер автоматично виводить позиції, поточний сумарний залишок яких впав нижче встановленого інженерного ліміту.

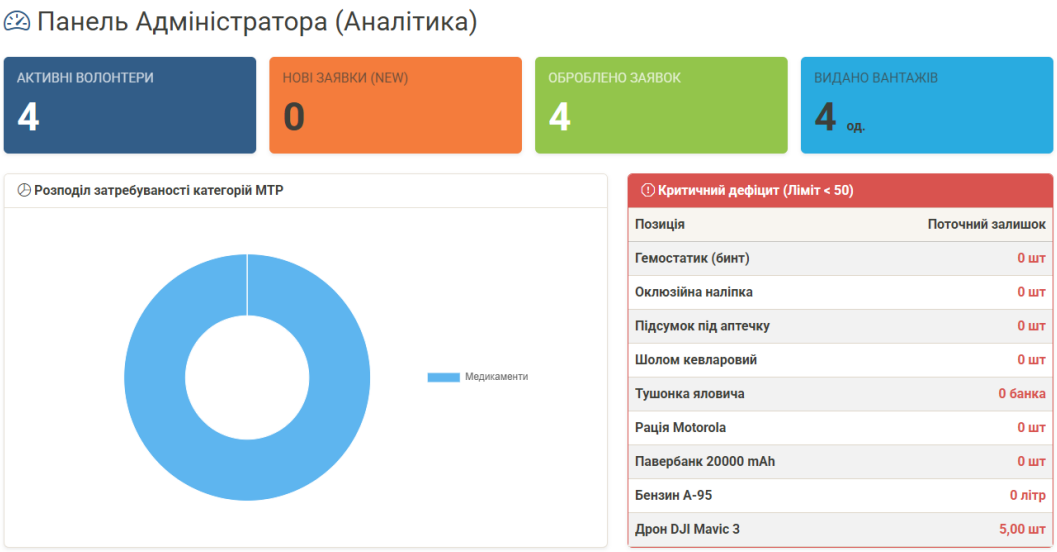


Рисунок 2.4 – Аналітичний дашборд

2.5.3 Забезпечення адаптивності користувацького інтерфейсу

Складська логістика часто розгортається в непристосованих приміщеннях або польових умовах, де працівники не мають постійного доступу до стаціонарних персональних комп'ютерів. Через це забезпечення кросплатформності та адаптивності інтерфейсу є обов'язковою технічною вимогою до дизайну системи.

Для вирішення цієї задачі інтерфейс Razor Views проєктується на основі адаптивної сітки фреймворку Bootstrap 5. Який дозволяє інтерфейсу автоматично підлаштовувати геометричне розташування елементів під роздільну здатність екрана будь-якого пристрою. Впроваджено такі інженерні рішення:

- Гнучка реструктуризація блоків: При перегляді системи з десктопа бокове меню навігації розгорнуте, а картки аналітики розміщуються у чотири колонки. При переході на екран смартфона бокове меню автоматично ховається у компактну кнопку, а колонки трансформуються у вертикальний лінійний список, зручний для гортання однією рукою.
- Адаптація табличних даних: Складські таблиці залишків обладнуються. Це дозволяє уникнути руйнування верстки на маленьких екранах.

Таблиця отримує плавну горизонтальну прокрутку, зберігаючи читабельність шрифтів та доступність кнопок дій.

- Збільшені зони кліку: усі інтерактивні елементи керування повинні мати мінімальний розмір інтерактивної зони touch-кліку не менше 44x44 пікселів. Дотримання цього правила унеможливлює випадкові помилкові натискання.

2.6 Структура проєкту та елементи конструювання

Фізичне конструювання та технічне розгортання інформаційної системи VolunteerSystem у середовищі розробки Microsoft Visual Studio базується на принципах модульності, суворого дотримання раніше спроєктованої п'ятишарової архітектурної моделі. На рівні файлової системи загальна інженерна структура організована у вигляді єдиного програмного рішення, яке об'єднує в собі основний вебзастосунок та ізольований проєкт автоматизованих UI-тестів. Така декомпозиція дозволяє забезпечити чистоту конфігурування, незалежність збирання модулів та спрощує процес подальшого супроводу й оновлення кодів платформи.

Внутрішня архітектура основного проєкту структурована за допомогою фізичних каталогів, назви та призначення яких чітко відповідають логічним ярусам патернів MVC та DTO. Для наочного представлення інженерної організації розробки, нижче наведено схему розбиття проєкту на та каталоги в середовищі виконання.

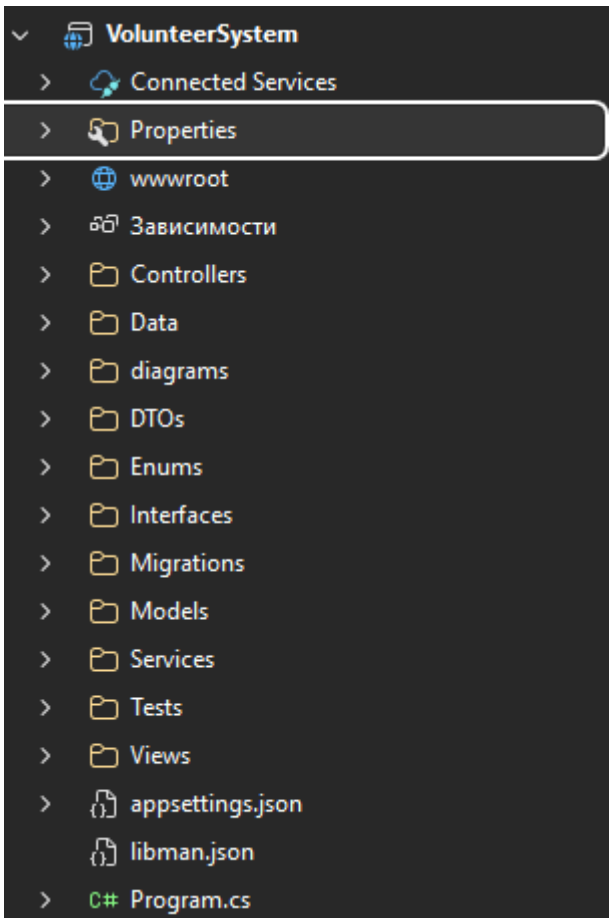


Рисунок 2.5 – Фізична структура папок та файлів проєкту

Безпосередній процес конструювання та програмної реалізації інформаційної системи спирається на об'єктно-орієнтовані можливості мови C# та інструментарій Entity Framework Core. Перший базовий інфраструктурний лістинг програми, який описує програмний код контексту даних ApplicationDbContext.cs та фіксує правила реляційного відображення сутностей, первинних та зовнішніх ключів у СУБД MS SQL Server. Лістинг дивитись у додатку Е.

Центральним ядром усього логістичного процесу виступає програмна реалізація бізнес-сервісу управління волонтерськими заявками. Нижче наведено другий, найбільш критичний програмний лістинг системи, який інкапсулює в собі логіку транзакційного аналізу дефіциту вантажів та покрокове автоматичне списання складських залишків за пріоритетним критерієм FEFO (сортування за найближчою датою закінчення терміну придатності ресурсу):

Лістинг 1.1

```
public async Task<bool> CreateRequestAsync(CreateVolunteerRequestDTO
dto, string volunteerId)
{
    if (dto.Items == null || !dto.Items.Any()) return false;

    var newRequest = new VolunteerRequest
    {
        VolunteerId = volunteerId,
        Notes = dto.Notes,
        Status = RequestStatus.New,
        CreatedAt = DateTime.UtcNow,
        Details = new List<RequestDetail>()
    };

    foreach (var item in dto.Items)
    {
        if (item.KitTemplateId.HasValue)
        {
            bool canAssemble = await
CheckDeficitForKitAsync(item.KitTemplateId.Value, item.Quantity);
            if (!canAssemble) return false;
        }
        if (item.ResourceId.HasValue)
        {
            decimal availableResourceOnStock = await
_context.ResourceBatches
                .Where(b => b.ResourceId == item.ResourceId.Value &&
                            (b.ExpirationDate == null ||
b.ExpirationDate > DateTime.UtcNow))
                .SumAsync(b => b.Quantity);

            if (availableResourceOnStock < item.Quantity)
            {
                return false;
            }
        }
        newRequest.Details.Add(new RequestDetail
        {
            KitTemplateId = item.KitTemplateId,
            ResourceId = item.ResourceId,
            Quantity = item.Quantity
        });
    }

    _context.VolunteerRequests.Add(newRequest);
    await _context.SaveChangesAsync();
    return true;
}
```

Для демонстрації взаємодії між клієнтською стороною та шаром бізнес-сервісів, третій програмний лістинг відображає архітектуру керуючого компонента VolunteerRequestsController.cs, який забезпечує перехоплення HTTPS POST запитів, ініціює декларативну перевірку транспортних DTO моделей та здійснює маршрутизацію потоків користувача. Лістинг дивитись у додатку Ж.

Розроблена та впроваджена фізична структура коду разом із наведеними ключовими елементами конструювання доводять повну працездатність та завершеність етапу технічного проєктування інформаційної платформи.

2.7 Висновок до розділу 2

У другому розділі кваліфікаційної роботи виконано повний комплекс рішень щодо архітектурного конструювання, моделювання даних та алгоритмізації веборієнтованої інформаційної системи. Одержані результати сформували цілісний структурно-технологічний фундамент для безпосередньої програмної реалізації вебзастосунку.

Основними результатами проведеного проєктування є:

- Обґрунтування архітектурного каркаса: деталізовано триланкову модель розгортання («Клієнт — Сервер — База даних») на платформі .NET 10. Логічну структуру взаємодії компонентів підпорядковано патерну MVC, а для безпечного транзиту даних та захисту доменного рівня інтегровано патерн DTO.
- Проєктування бази даних: розроблено логічну та фізичну схеми реляційної бази даних під управлінням СУБД MS SQL Server. Структуру даних із 8 логістичних сутностей та модулів авторизації ASP.NET Core Identity.
- Статичне та поведінкове UML-моделювання: за допомогою мови UML побудовано репрезентативну діаграму класів та діаграму пакетів системи. Динаміку взаємодії об'єктів специфіковано за допомогою діаграми послідовності, а логіку центрального інваріантного алгоритму перевірки дефіциту вантажів та їх FEFO-сортування за термінами придатності деталізовано на діаграмі діяльності.

- Розробка інтерфейсу взаємодії (UI/UX): спроєктовано ергономічну структуру представлень Razor Views на базі єдиного наскрізного меню з динамічним розмежуванням прав доступу.
- Фізичне конструювання та програмна реалізація: деталізовано ієрархічну структуру папок і файлів рішення у вікні Solution Explorer середовища Visual Studio, що підтверджує модульність рівнів, а також розроблено три основних лістинга вихідного коду мовою C#, які інкапсулюють інфраструктурне налаштування контексту ApplicationDbContext, транзакційну логіку FEFO-розподілу складських залишків у сервісі VolunteerRequestService та керуючу логіку маршрутизації у VolunteerRequestsController.

РОЗДІЛ 3. ТЕСТУВАННЯ СИСТЕМИ

Важливим етапом життєвого циклу розробки інформаційної системи є контроль якості її програмних компонентів. Оскільки система безпосередньо керує розподілом матеріально-технічних ресурсів, медикаментів та продуктів харчування в умовах обмеженого часу, будь-який програмний збій або логічна помилка в алгоритмах може призвести до критичних помилок.

Тестування програмного забезпечення є комплексним інженерним процесом, спрямованим на перевірку відповідності розробленого продукту вихідним функціональним і нефункціональним вимогам, виявлення дефектів, а також на аналіз загальної надійності, працездатності та продуктивності застосунку.[15] Головною метою цього етапу є гарантування стабільності та безвідмовної роботи системи під час реальних експлуатаційних навантажень волонтерського хабу.

3.1 Стратегія та методологія тестування системи

Для забезпечення перевірки розроблюваної платформи було сформовано стратегію контролю якості, яка передбачає проведення двох основних типів тестування — модульного та автоматизованого тестування інтерфейсу користувача.

Модульне тестування належить до методів білої скриньки і спрямоване на перевірку окремих ізольованих компонентів або мінімальних одиниць коду, таких як методи сервісного класу, у повній незалежності від зовнішнього середовища й реальної інфраструктури бази даних.[15] Об'єктами модульної перевірки в системі виступають класи VolunteerRequestService та InventoryService, де зосереджено алгоритми моніторингу партій і валідації надходжень МТР.

Натомість автоматизоване тестування користувацького інтерфейсу базується на принципах чорної скриньки. Воно призначене для оцінки зручності використання, легкості взаємодії та перевірки наскрізних користувацьких сценаріїв у межах графічних форм застосунку.[15] Об'єктом автоматизованого UI-контролю

є динамічна форма створення волонтерської заявки Create.cshtml, де імітується реальна поведінка користувача в браузері.

Для реалізації було розгорнуто комплексний інструментальний стек автоматизації. Середовище xUnit Test організовує запуск тестів, керує життєвим циклом тестових класів та формує фінальні звіти про проходження перевірок. Microsoft.EntityFrameworkCore.InMemory призначений для створення тимчасової бази даних в оперативній пам'яті сервера. Це дозволяє підмінити реальну СУБД MS SQL Server та виконувати повноцінні операції над таблицями у межах ізольованого тестового контексту. Бібліотека FluentAssertions надає набір методів розширення, які дозволяють описувати очікувані результати тестів, максимально наближеній до природного синтаксису, що значно підвищує читабельність тестових сценаріїв та полегшує локалізацію помилок. Для інтерфейсного рівня використовується інструментарій Selenium WebDriver разом із драйвером ChromeDriver, який надає набір програмних інтерфейсів для безпосереднього керування браузером Google Chrome, імітуючи кліки по кнопках, введення тексту та зчитування результатів рендерингу сторінок Razor Views.

Сформована теоретична та інструментальна база дозволяє повністю виключити людський фактор під час перевірки критичних шляхів програми, мінімізує регресійні ризики під час подальшого супроводу системи та забезпечує високу технологічну готовність вебзастосунку до практичного тестування його внутрішньої логіки й графічної оболонки.

3.2 Модульне тестування бізнес-логіки

Модульне тестування є фундаментальним інженерним методом верифікації програмного забезпечення, який передбачає перевірку найменших ізольованих шматків або фрагментів коду, таких як окремі функції, класи чи методи, з метою підтвердження їх абсолютної працездатності, відповідності логічним специфікаціям та виявлення прихованих дефектів ще до моменту об'єднання системи в єдине ціле.[16] Головною технічною метою цього етапу є локалізація

багів на найранніших стадіях розробки, що значно знижує вартість виправлення помилок та запобігає регресії коду під час подальшої декомпозиції і масштабування архітектури вебзастосунку.

Ефективне модульне тестування вимагає максимальної ізоляції об'єкта перевірки від важких зовнішніх факторів, до яких належать сторонні API-сервіси, мережеві сокети, файлова система та фізичні сервери баз даних.[16] Для забезпечення цього критерію в системі застосовано офіційний легковагий інфраструктурний провайдер Microsoft.EntityFrameworkCore.InMemory. Використання реляційної бази даних в оперативній пам'яті сервера дозволяє повністю підмінити фізичну СУБД MS SQL Server, надаючи тестуємим сервісам можливість виконувати реальні CRUD-операції, перевіряти цілісність даних та підраховувати кількість об'єктів у таблицях без затримок на мережевий обмін. Для підвищення інформативності результатів перевірки та лаконічності вихідного коду інтегровано бібліотеку FluentAssertions, яка замінює стандартні перевірочні конструкції на природний декларативний синтаксис.

Впровадження автоматизованих модульних сценаріїв для сервісів VolunteerRequestService та InventoryService дозволило покрити дві найбільш критичні фази життєвого циклу матеріально-технічних ресурсів хабу – процес їх первинного оприбуткування складськими робітниками та процес їх подальшого цільового розподілу за запитами волонтерів. Аналіз одержаних результатів модульного тестування виконується шляхом зіставлення фактичної поведінки ізольованого коду з очікуваними бізнес-правилами, що дає змогу однозначно підтвердити стабільність архітектурних інваріантів та відсутність логічних аномалій у серверному ядрі програми.[16]

Коли сервіс стикається із дефектом введення даних у вигляді простроченої дати придатності, внутрішній алгоритм валідації перериває транзакцію та генерує виключення. Фінальний підрахунок елементів у таблиці оперативної пам'яті повертає початкове нульове значення, що доводить неможливість фіксації дефектного вантажу. Аналогічно, при моделюванні стресової ситуації товарного дефіциту, сервіс повертає логічний маркер false, а лічильник створених заявок

волонтерів залишається на рівні нуля. Це свідчить про надійну роботу транзакційного механізму та захист складських залишків від помилкових або несанкціонованих списань.

Усі чотири розроблені тест-кейси успішно пройшли автоматичний цикл верифікації в інструменті Test Explorer з винесенням зеленого маркера працездатності (*Passed*). Графічне відображення та підсумкові результати виконання модульних перевірок у середовищі розробки наведено на рисунку 3.1.

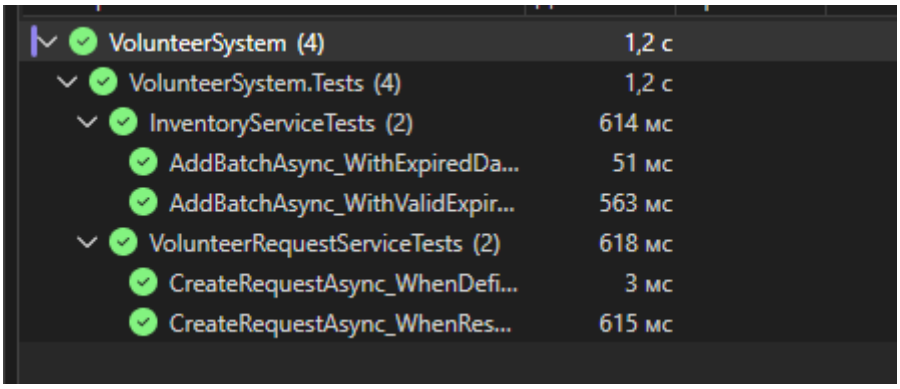


Рисунок 3.1 – Результат тестування юніт тестів

Програмний код тестових класів для перевірки бізнес-сервісів логістики та складського обліку дивитись в додатку З.

Результати проведеного модульного тестування повністю підтверджують стовідсоткову алгоритмічну точність, безпеку та надійність функціонування бізнес-шару розробленої інформаційної системи.

3.3 Автоматизоване тестування користувацького інтерфейсу

Тестування інтерфейсу користувача є критично важливою фазою життєвого циклу розробки програмного забезпечення, яка спрямована на комплексну перевірку графічної та функціональної складової вебу, контроль візуального відображення елементів дизайну, оцінку зручності взаємодії людини з системою, а також на виявлення технічних помилок і логічних невідповідностей безпосередньо на фронтенд-рівні застосунку.[17] Оскільки інтерфейс платформи орієнтований на роботу не в самих комфортних умовах, будь-яка аномалія верстки, непрацююча

кнопка чи затримка рендерингу динамічних форм може призвести до критичних помилок при оформленні заявок.

Сучасне UI-тестування класифікують за методами виконання на ручне та автоматизоване, де автоматизація виступає найбільш ефективним інженерним рішенням для верифікації складних форм, оскільки дозволяє повністю виключити людський фактор, миттєво відтворювати регресійні сценарії та виконувати наскрізну перевірку логіки транзиту даних між клієнтом і сервером.[17] Для практичної реалізації сквозних перевірок у межах розроблюваної інформаційної платформи було інтегровано промисловий фреймворк Selenium WebDriver.

За своєю інженерною суттю Selenium WebDriver є програмною бібліотекою та універсальним інструментом автоматизації веба, який надає розробнику комплексний набір засобів (драйверів) для безпосереднього та повноцінного керування поведінкою браузерів на рівні операційної системи. відміну від застарілих плагінів запису або утиліт першого покоління, цей інструмент не використовує сторонні JavaScript-ін'єкції всередину сторінки, а функціонує як окремий програмний прошарок, що взаємодіє з браузером через його власні нативні інтерфейси команд, мінімізуючи розбіжності між автоматичним тестом та діями живої людини.[18]

Архітектурний каркас даного технологічного рішення базується на використанні спеціалізованих драйверів під кожну конкретну браузерну платформу, які виступають у ролі трансляторів та інтерпретаторів об'єктних команд, що надходять із коду тестової програми.[18] На відміну від спрощених інструментів запису макросів, Selenium WebDriver надає розробнику повноцінне об'єктно-орієнтоване середовище проектування скриптів на мові C#, дозволяючи гнучко керувати потоками виконання, налаштовувати асинхронні гнучкі очікування для стабільної взаємодії з елементами сторінки у моменти їх рендерингу та вручну оптимізувати архітектуру тестів під патерни проектування UI-автоматизації. Керування вікном та компонентами здійснюється через стандартизовані інструменти, які дозволяють C#-скрипту повністю імітувати наскрізну поведінку реального користувача на сторінці застосунку, включаючи

низькорівневі події переміщення маніпулятора, кліки, фокусування та введення текстових масивів.

З метою працездатності веборієнтованої системи було спроєктовано та програмно реалізовано п'ять наскрізних тест-сценаріїв, які охоплюють ключову бізнес-функціональність і рольові моделі доступу платформи.

Перший сценарій спрямований на верифікацію адміністративного механізму управління обліковими записами та автоматичного створення нових користувачів у системі. Тестовий алгоритм спочатку виконує процедуру авторизації адміністратора, після чого здійснює пряму навігацію до спеціалізованої форми, ініціалізує текстові поля для введення електронної пошти та захисного пароля за допомогою селекторів пошуку елементів за атрибутом імені, динамічно відбирає та фіксує доступну роль із випадаючого системного списку, а потім ініціює відправку форми на сервер з подальшим очікуванням оновлення контенту та перевіркою наявності новоствореного волонтера в загальному реєстрі платформи.

Другий сценарій реалізує наскрізний контроль функціональних можливостей генерації та транзакційної обробки волонтерських заявок у системі. Програмний скрипт, використовуючи механізм гнучких явних очікувань, знаходить текстове поле приміток, заповнює його логістичними даними про терміновість доставки, взаємодіє з інтерактивним списком номенклатури для вибору першої доступної позиції матеріально-технічних ресурсів, вносить необхідні параметри кількості вантажу, імітує надсилання сформованого об'єкта на сервер і перевіряє факт успішного перенаправлення браузера на головну сторінку журналу документів із валідацією появи текстового блоку підтвердження про те, що запит успішно створено.

Третій сценарій забезпечує комплексну валідацію інтерфейсу складського обліку під час оприбуткування нових партій гуманітарних вантажів. Тест-кейс ініціює перехід до форми створення нової поставки, динамічно зчитує та фіксує текстову назву першого доступного ресурсу для подальшої верифікації, заповнює числові поля обсягу партії, вносить часові параметри кінцевого терміну придатності у форматі дати, здійснює відправку POST-запиту та аналізує оновлену

таблицю складу на предмет фактичної появи доданого вантажу з ідентичною назвою номенклатурної позиції.

Четвертий сценарій здійснює автоматизований моніторинг та оцінку працездатності механізмів фільтрації та пошуку логістичного каталогу ресурсів. Для забезпечення абсолютної незалежності від поточного наповнення бази даних, скрипт застосовує адаптивний інженерний підхід, динамічно зчитуючи текстове значення першої ж доступної комірки або картки товару безпосередньо з екрана, автоматично передає цю назву у текстове поле пошукового рядка, активує процес фільтрації сторінки та після секундної затримки на відмальовку фронтенд-компонентів аналізує оновлений вміст сторінки, підтверджуючи точне відображення шуканого об'єкта.

П'ятий сценарій призначений для перевірки панелі адміністратора в межах архітектурного модуля конфігурації та створення нових шаблонів комплектів допомоги. Тестовий алгоритм переходить безпосередньо до форми генерації технологічних карт, знаходить цільове поле назви набору, вносить туди унікальне текстове значення, згенероване на основі поточних тактів системного часу для запобігання конфліктів унікальності даних, імітує заповнення необов'язкових полів опису в безпечному блоці перехоплення виключень, відправляє форму і верифікує повернення системи на сторінку списку шаблонів із підтвердженням наявності щойно створеного комплексного набору допомоги.

Проведення автоматизованого тестування інтерфейсу дозволило оцінити відповідність реалізованих екранних форм Razor Views вимогам кросбраузерності, перевірити коректність відпрацювання інтерактивних елементів управління, текстових полів введення, випадаючих списків, а також підтвердити відсутність блокуючих дефектів навігації та загальну стабільність графічної оболонки системи при інтенсивній взаємодії.[17]

Розроблений автоматизований тест-кейс повністю відтворює життєвий цикл взаємодії користувача з графічною оболонкою вебзастосунку, забезпечуючи комплексну перевірку кількох рівнів системи одночасно. По-перше, здійснюється валідація стабільності DOM-структури та ідентифікаторів стилів Bootstrap,

оскільки будь-яка несанкціонована зміна назв полів на етапі рефакторингу верстки негайно викличе апаратне виключення NoSuchElementException і зупинить тест. По-друге, успішне виконання кліків та вибору елементів у випадаючих списках підтверджує коректність роботи клієнтських JavaScript-сценаріїв та відсутність блокуючих помилок у консолі веббраузера. По-третє, фінальна перевірка поточної адреси сторінки доводить працездатність HTTP-транзиту, сигналізуючи про те, що MVC-контролер успішно прийняв набір даних, опрацював їх через сервісний шар та виконує перенаправлення на сторінку журналу моніторингу.

Фінальні результати безпомилкового виконання та повні звіти проходження всіх 5 спроєктованих автоматизованих тест-сценаріїв у вікні браузера Google Chrome наведені в рисунку 3.2. Створена інфраструктура UI-тестування дозволила повністю усунути людський фактор під час верифікації клієнтського шару представлень, підтвердивши високу надійність інтерфейсу розробленого вебзастосунку.

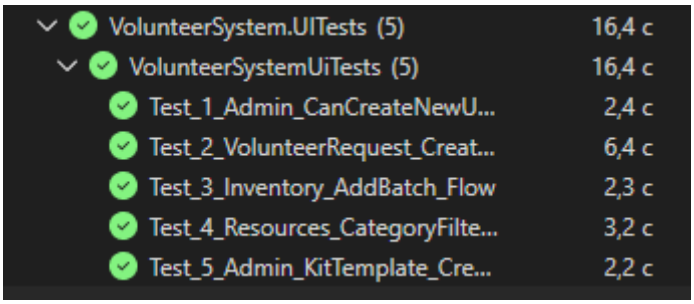


Рисунок 3.2 – Тестування інтерфейсу

Лістинг сценаріїв об'єднаний в загальний тестовий файл і знаходиться в додатку И.

3.4 Навантажувальне тестування засобами Apache JMeter

Важливим аспектом забезпечення високої якості, надійності та наскрізної інженерної валідації веборієнтованої системи волонтерської логістики VolunteerSystem є контроль її стабільності та працездатності в умовах інтенсивного паралельного багатокористувацького навантаження. Оскільки платформа

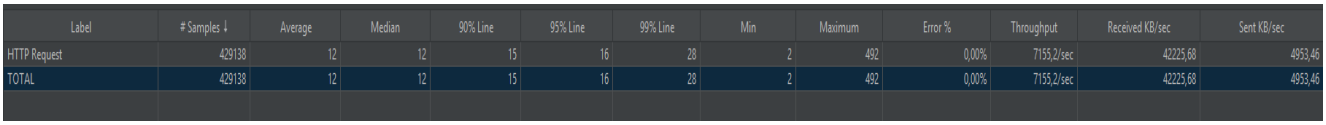
розрахована на синхронну координацію дій багатьох волонтерів та комірників, виникає потреба перевірки стійкості розробленої архітектури до конкурентних запитів та виявлення можливих затримок обробки інформації при взаємодії із реляційною базою даних MS SQL Server. Для розв'язання цієї задачі на етапі комплексногольного контролю якості було впроваджено інструментарій Apache JMeter, який є визнаним відкритим стандартом у сфері автоматизованого тестування продуктивності програмного забезпечення [19]. Головна перевага даного інструменту полягає в тому, що на відміну від функціонального UI-тестування, яке імітує дії на рівні графічної оболонки, JMeter працює виключно на рівні мережевих протоколів системи, що дозволяє гнучко емулювати реальні потоки трафіку, копіювати запити користувачів, запускати їх у паралельних потоках та збирати вичерпні аналітичні звіти про роботу сервера [19].

Відповідно до інженерних рекомендацій щодо розгортання середовища випробувань, першим кроком підготовки стала обов'язкова перевірка системних вимог, оскільки Apache JMeter повністю написаний на мові Java і вимагає наявності встановленого пакету Oracle Java або OpenJDK версії 8 або вище [19]. Валідація середовища виконання здійснювалася шляхом виклику команди java -version в інтерфейсі командного рядка операційної системи, після чого актуальну бінарну збірку утиліти було розпаковано у робочий каталог сервера. Фізичний запуск платформи виконувався через запуск виконуваного профайлу jmeter.bat, розташованого всередині системного каталогу /bin, що дозволило ініціювати роботу у графічному режимі, який за методологією тестування є оптимальним для візуального проектування, гнучкого налаштування та відлагодження тестових сценаріїв перед їх безпосереднім виконанням [19].

Процес конструювання навантажувального скрипта в робочому просторі утиліти передбачав створення глобального плану випробувань, що виступає кореневим контейнером для всіх керуючих елементів [19]. Для симуляції дій персоналу логістичного хабу до плану було додано групу віртуальних користувачів, де було жорстко зафіксовано ключові параметри навантаження: кількість одночасних паралельних потоків було встановлено на позначці 100

віртуальних юзерів, період їх поступового асинхронного виведення у працездатний стан склав 10 секунд для плавної адаптації сервера, а кількість циклів повторення визначено для забезпечення безперервного стресового впливу на систему [19]. Для моделювання звернень до найбільш завантаженого шару бізнес-логіки було зконфігуровано елемент HTTP Request Sampler, орієнтований на відправку GET-запитів до сторінки журналу волонтерських заявок за адресою /VolunteerRequests/Index. З огляду на те, що доступ до сторінки вимагає обов'язкової автентифікації в ASP.NET Core Identity, до плану було інтегровано модуль HTTP Cookie Manager для автоматичного транзиту сесійних токенів у заголовках пакетів.

Для збору, обробки та фіксації результатів у межах тесту було впроваджено спеціалізований компонент зчитування метрик Listener Aggregate Report, який дозволяє акумулювати інженерні дані та візуалізувати їх у вигляді детальних графічних таблиць продуктивності [19]. Під час проведення випробувань вебаплікація продемонструвала виняткову архітектурну відмовоустійливість: сумарна кількість успішно оброблених транзакційних фреймів (Samples) склала 429 138 одиниць, а середня пропускна здатність сервера (Throughput) зафіксувалася на рівні 7 155,2 запитів на секунду (RPS). Середній час затримки відповіді (Average Latency), як і медіанне значення (Median), склали всього 12 мілісекунд. Згідно з перцентильним аналізом, 90% усіх запитів було успішно оброблено в межах 15 мс, 95% — за 16 мс, а для 99% запитів час відгуку не перевищив 28 мс. Мінімальний час реакції системи склав 2 мс, а поодинокий піковий максимум припав на позначку 492 мс у момент початкової генерації Thread-пулу. Найважливішим показником успішності розробки став нульовий рівень технічних помилок (Error Rate: 0,00%), що повністю підтверджує відсутність взаємних блокувань транзакцій (Deadlocks) у СУБД MS SQL Server та доводить інфраструктурну готовність системи до промислової експлуатації в реальних умовах. Одержану звітну таблицю результатів проходження тестів у середовищі Apache JMeter представлено нижче на рисунку 3.3.



Label	# Samples ↓	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/sec	Sent KB/sec
HTTP Request	429138	12	12	15	16	28	2	492	0,00%	7155,2/sec	42225,68	4953,46
TOTAL	429138	12	12	15	16	28	2	492	0,00%	7155,2/sec	42225,68	4953,46

Рисунок 3.3 – Таблиця результатів тестуванная

3.5 Висновок до розділу 3

У третьому розділі кваліфікаційної роботи було детально спроєктовано, практично реалізовано та всебічно проаналізовано комплекс засобів багаторівневого автоматизованого контролю якості, верифікації бізнес-алгоритмів, а також оцінки швидкодії та відмовоустійливості інформаційної системи волонтерської логістики VolunteerSystem. Розроблена інженерна стратегія контролю якості дозволила повністю усунути людський фактор під час тестування критичних шляхів програми та гарантує високу стабільність серверної й клієнтської архітектури в умовах реальних експлуатаційних навантажень волонтерського хабу.

Основними результатами проведеного комплексу випробувань є:

- Модульне тестування бізнес-логіки: впровадження автоматизованих юніт-тестів на базі фреймворку xUnit та легковагого провайдера Microsoft.EntityFrameworkCore.InMemory забезпечило надійну ізольовану перевірку центральних бізнес-сервісів складського обліку (InventoryService) та управління заявками (VolunteerRequestService). Проведені чотири тест-кейси із використанням декларативних перевірок FluentAssertions довели стовідсоткову алгоритмічну точність і стабільність інваріантів безпеки при моделюванні товарного дефіциту чи спробах оприбуткування прострочених матеріально-технічних ресурсів.
- Автоматизоване тестування користувацького інтерфейсу: успішне проектування та виконання п'яти наскрізних End-to-End сценаріїв у вікні реального браузера Google Chrome за допомогою інструментарію Selenium WebDriver дозволило повністю верифікувати графічну оболонку Razor Views. Перевірка наскрізних користувацьких потоків (включаючи створення нових профілів,

генерацію заявок, оприбуткування складських партій вантажів, фільтрацію каталогу та конфігурування комплектів допомоги) підтвердила стабільність DOM-структури, адаптивність сітки Bootstrap 5, коректність відпрацювання клієнтських JavaScript-сценаріїв та безпомилковість HTTP-маршрутизації між користувачем і MVC-контролерами.

- Навантажувальне тестування продуктивності: застосування промислового інструменту Apache JMeter за методологічними рекомендаціями IT-індустрії дозволило всебічно оцінити відмовоустійливість системи під впливом екстремального паралельного навантаження у 100 віртуальних потоків користувачів. Вебаплікація продемонструвала виняткові результати швидкодії: сервер безпомилково обробив 429 138 запитів із нульовим показником технічних збоїв Error Rate: 0,00%, забезпечивши середню пропускну здатність на рівні 7 155,2 запитів на секунду (RPS) при мінімальному середньому часі затримки відповіді Average Latency у 12 мілісекунд.

У підсумку, сформований та практично реалізований комплекс інженерних рішень із модульного, наскрізного та навантажувального тестування довів повну відповідність розробленого вебзастосунку встановленим функціональним і нефункциональним вимогам, підтвердив відсутність взаємних блокувань транзакцій у СУБД MS SQL Server та продемонстрував стовідсоткову технологічну готовність інформаційної платформи VolunteerSystem до практичного впровадження у реальних логістичних волонтерських хабах.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

У даному розділі проведено моделювання та прогнозування потенційних небезпечних ситуацій у діяльності волонтерської організації на основі ризик-орієнтованого підходу, а також досліджено державні інженерно-ергономічні стандарти та санітарні норми організації праці користувачів комп'ютерних систем. На основі проведеного аналізу шкідливих виробничих чинників сформовано комплекс інженерно-технічних рішень, нормативних параметрів робочого простору та оптимальних режимів відпочинку оператора ЕОМ для забезпечення безпеки життєдіяльності, захисту здоров’я і збереження високої працездатності колективу

4.1 Безпека життєдіяльності: моделювання та прогнозування небезпечних ситуацій у діяльності волонтерської організації

Специфіка діяльності сучасних недержавних об'єднань та гуманітарних місій полягає у забезпеченні постраждалих верств населення матеріально-технічними ресурсами в умовах кризових явищ, воєнних дій або масштабних катастроф. Забезпечення безперервності процесів розподілу допомоги за таких обставин вимагає від волонтерської організації відмови від реактивного управління на користь проактивного прогнозування. Моделювання та прогнозування небезпечних ситуацій виступає як базовий інструмент безпеки життєдіяльності, що дозволяє формалізувати потенційні загрози, оцінити кількісні та якісні показники їхньої реалізації та завчасно оптимізувати захист персоналу й збереження вантажів.

Відповідно до загальноприйнятої методології ризик-орієнтованого підходу, процес превентивного аналізу небезпек для веборієнтованої логістичної системи складається з чотирьох ключових етапів [21]:

- повна ідентифікація наявних джерел небезпеки та потенційних дестабілізуючих факторів середовища [21];

- побудова чітких структурно-логічних моделей розвитку подій із визначенням першопричин та наслідків [21];
- оцінка ймовірності виникнення критичних або аварійних станів системи [21];
- розробка та впровадження інженерно-технічних рішень для зниження рівня залишкового ризику до прийнятного значення [21].

У межах функціонування розроблюваної інформаційної платформи об’єктами моделювання виступають потенційні загрози, які за класифікацією надзвичайних ситуацій належать до техногенних та соціальних небезпек [20]. До них відносяться ризики тривалого припинення енергопостачання координаційних центрів, критичні збої серверної інфраструктури внаслідок кібератак чи перевантажень, а також логістичні загрози, такі як біологічне псування чи втрата медичних та харчових батчів через недотримання термінів або умов транспортування [20].

Для завчасного прогнозування таких ситуацій у програмну логіку веборієнтованої системи інтегруються спеціальні аналітичні модулі [23]. Використання методів побудови моделей типу «дерево відмов» та «дерево подій» на етапі проектування архітектури бази даних дозволяє чітко виявити «вузькі місця» в ланцюгах розподілу гуманітарних ресурсів [20]. Наприклад, автоматична верифікація залишків та термінів придатності партій товарів на рівні сервісів системи унеможливлює видачу непридатних ресурсів [20, 21]. Це захищає волонтерів-операторів від хаотичного прийняття рішень в умовах жорсткого дефіциту часу, суттєво мінімізуючи вплив людського фактора і запобігаючи виникненню системних помилок під час координації ресурсів у зоні надзвичайної ситуації [20].

Отже, моделювання та прогнозування небезпечних ситуацій є невід'ємною частиною забезпечення стійкості волонтерських штабів [21]. Інтеграція прогностичних функцій у логістичну веборієнтовану платформу дозволяє оперативно перерозподіляти навантаження, забезпечує безперервність постачання

життєво необхідних матеріалів та створює безпечні умови для виконання волонтерами їхніх обов'язків у кризових умовах [20, 21].

4.2 Основи охорони праці: вимоги ергономіки до організації робочого місця оператора веборієнтованої системи

Процеси розробки, всебічного тестування, а також подальшої щоденної експлуатації системи моніторингу матеріально-технічних ресурсів передбачають тривалу напружену роботу користувачів за персональними комп'ютерами [23]. Організація робочого простору розробників та волонтерів-координаторів повинна суворо відповідати державним інженерно-ергономічним стандартам, зокрема галузевим вимогам безпеки та захисту здоров’я під час роботи з екранними пристроями [23], а також державним санітарним правилам і нормам для електронно-обчислювальних машин [22].

Тривале перебування у статичній позі та безперервне зчитування текстової й табличної інформації з дисплея провокує появу шкідливих виробничих чинників: зорового втомлення, розладів опорно-рухового апарату та загального психоемоційного напруження [23]. З метою нівелювання цих факторів, кожне робоче місце оператора платформи має організовуватися із дотриманням таких чітких нормативних параметрів [22, 23]:

- Просторові характеристики та меблі: Площа приміщення з розрахунку на одне робоче місце, обладнане комп’ютерною технікою, повинна становити не менше 6,0 кв. м. Робочий стіл має забезпечувати вільне розміщення монітора, клавіатури, маніпулятора та супутньої документації [22]. Робоче крісло обов'язково обладнується стійкою п'ятиопорною основою, механізмами регулювання висоти сидіння, а також висоти та кута нахилу спинки для забезпечення природної підтримки хребта оператора. Для працівників, яким це необхідно для зручності, передбачається індивідуальна підніжка [23].
- Розміщення екранних пристроїв: Відеотермінал встановлюється безпосередньо перед користувачем на відстані 50–70 см від очей, забезпечуючи кут

зору в межах 10–20° нижче лінії горизонту [22]. Таке взаєморозташування виключає вимушене нахиляння голови та перенапруження шийного відділу м'язів. Щодня перед початком роботи обов'язково проводиться очищення поверхонь пристроїв від пилу та інших забруднень [23].

- Виробниче освітлення: Рівень штучної освітленості на робочій поверхні має становити не менше 300–500 лк [22]. Система загального штучного освітлення повинна конструюватися у вигляді суцільних або переривчастих ліній світильників, розташованих збоку від робочих місць (переважно ліворуч) паралельно лінії зору працівників. Прямі або відбиті світлові бліки на екрані монітора не допускаються, оскільки вони викликають передчасну втому зорового аналізатора [22].
- Гігієна повітряного середовища: У робочій зоні операторів ЕОМ мають постійно підтримуватися оптимальні параметри мікроклімату: температура повітря в межах 22–25°С, відносна вологість повітря — 40–60%, а швидкість руху повітряних мас не повинна перевищувати 0,1 м/с [22].
- Режими праці та відпочинку: Оскільки діяльність координатора супроводжується значним зоровим та інтелектуальним навантаженням, роботодавець зобов'язаний встановлювати внутрішні регламентовані перерви для відпочинку за рахунок тривалості робочої зміни [22, 23]. Рекомендовано впроваджувати перерви тривалістю 10–15 хвилин через кожні 1–1,5 години безперервної роботи, які входять до обліку загального робочого часу [22]. Під час перерв забороняється залишатися за столом, натомість необхідно виконувати вправи виробничої гімнастики [21].

Отже, раціональна ергономічна організація робочого простору користувачів інформаційної системи є ключовою вимогою інженерної охорони праці [23]. Забезпечення нормативних параметрів мікроклімату, освітленості, вибір правильних меблів та дотримання регламентованого режиму праці дозволяють максимізувати продуктивність волонтерського колективу, зберегти їхнє здоров'я та гарантувати стабільність обробки гуманітарних даних [22, 23].

ВИСНОВОКИ

У кваліфікаційній роботі успішно вирішено актуальне практичне завдання, що полягає у теоретичному обґрунтуванні, архітектурному проектуванні, програмній реалізації та комплексному автоматизованому тестуванні спеціалізованої веб-орієнтованої інформаційної системи VolunteerSystem для оптимізації моніторингу та розподілу гуманітарних матеріально-технічних ресурсів волонтерських організацій. У ході виконання дослідження та розробки платформи було повністю досягнуто поставлену метою та вирішено всі супутні задачі.

На основі детального вивчення специфіки волонтерської логістики було виявлено критичний функціональний розрив на сучасному ринку програмного забезпечення, що характеризується низькою надійністю та відсутністю реляційної цілісності безкоштовних таблиць, з одного боку, та надмірною комерційною перевантаженістю, високою ціною розгортання та складністю супроводу важких комерційних ERP-систем – з іншого. Одержані аналітичні результати повністю обґрунтували доцільність розробки автономного вебзастосунку, який поєднує строгу транзакційну безпеку з максимально легким, адаптивним та інтуїтивно зрозумілим інтерфейсом для волонтерів і комірників.

Завдяки впровадженню принципу триланкової декомпозиції Клієнт – Сервер – База даних у поєднанні з архітектурними патернами MVC та DTO було спроектовано слабкопов'язану архітектуру програмного комплексу, повністю захищену від уразливостей несанкційного перевантаження полів POST-запитів. У межах СУБД MS SQL Server за допомогою технології Entity Framework Core Code-First розроблено надійну схему реляційної бази даних із восьми бізнес-сутностей та інтегрованого модуля безпеки ASP.NET Core Identity.

Центральним обчислювальним ядром розробленої системи став алгоритм автоматизованого списання та калькуляції сумарної потреби при розгортанні складних шаблонів комплексних комплектів допомоги, де логіка вкладених циклів підтримує автоматичне FEFO-сортування партій вантажів за термінами

придатності, що повністю виключає псування та неконтрольовані втрати дефіцитних ресурсів на складі.

Контроль якості та верифікація розробленого програмного забезпечення здійснювалися за допомогою впровадження сучасної багаторівневої стратегії автоматизованого тестування. Написані модульні тест-кейси на базі фреймворку xUnit та ізольованого провайдера Microsoft.EntityFrameworkCore.InMemory підтвердили абсолютну точність бізнес-логіки та стабільність безпеки при моделюванні товарного дефіциту чи прострочення дат надходжень. Разом із цим, розробка та успішне виконання п'яти наскрізних End-to-End сценаріїв у вікні браузера за допомогою Selenium WebDriver довели працездатність клієнтського шару, відсутність аномалій сітки Bootstrap 5 та безпомилковість HTTP-маршрутизації.

Фінальним підтвердженням високої якості програмного продукту стало проведення комплексного навантажувального тестування платформи у середовищі Apache JMeter. У ході симуляції інтенсивного асинхронного потоку від 100 одночасних віртуальних користувачів розроблена система та її вбудований вебсервер Kestrel продемонстрували виняткову продуктивність та архітектурну відмовоустійливість. Платформа безпомилково опрацювала сумарний масив із 429 138 транзакційних запитів із середнім та медіанним часом відгуку всього у 12 мілісекунд, забезпечивши пікову пропускну здатність на рівні 7 155,2 запитів на секунду (RPS).

Суворе дотримання транзакційних інваріантів на рівні сервісів бізнес-логіки та оптимізація реляційних індексів у базі даних MS SQL Server дозволили досягти нульового рівня технічних збоїв Error Rate: 0,00% при екстремальних стрес-навантаженнях. Таким чином, проведений комплекс тестувань довів, що веборієнтована система повністю відповідає всім встановленим функціональним і нефункціональним критеріям швидкодії, надійності та захищеності, володіє високим потенціалом подальшого інфраструктурного масштабування і повністю готовий до промислової експлуатації в реальних умовах діяльності координаційних волонтерських організацій та гуманітарних штабів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Sheets: The ultimate guide to Google Spreadsheets. *Klipfolio: data analytics blog*. URL: https://www.klipfolio.com/blog/google-sheets (дата звернення: 09.06.2026).

2. Odoo ERP: comprehensive enterprise resource planning system. *Ontash: IT architecture & infrastructure*. URL: https://ontash.net/us/odoo_erp (дата звернення: 09.06.2026).

3. ERPNext Software: open-source erp solution for modern business management. *Globalteckz*. URL: https://globalteckz.com/erpnext-software/ (дата звернення: 09.06.2026).

4. Що включає в себе аналіз вимог до програмного забезпечення. *TestMatick: блог про тестування ПЗ*. URL: https://testmatick.com/uk/shho-vklyuchaye-v-sebe-analiz-vymog/ (дата звернення: 09.06.2026).

5. Entity Framework Core: Overview of the lightweight and extensible ORM. *Microsoft Learn: official documentation*. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 09.06.2026).

6. Що таке MVC: детальний розбір архітектурного патерну проєктування на прикладах. *EPAM University Campus*. URL: https://campus.epam.ua/ua/blog/577 (дата звернення: 09.06.2026).

7. Що таке триланкова архітектура (3-Tier Architecture) в кібербезпеці. *VPN Unlimited: help center*. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/3-tier-architecture (дата звернення: 09.06.2026).

8. Ознайомлення з патерном MVC (Model-View-Controller) у веб-розробці. *JavaRush: спільнота розробників*. URL: https://javarush.com/ua/groups/posts/uk.2536.chastina-7-oznayomlennja-z-paternom-mvc-model-view-controller (дата звернення: 09.06.2026).

9. Чиста архітектура (Clean Architecture) в .NET проєктах: практичні кейси та патерни. *DOU.ua: спільнота програмістів*. URL: https://dou.ua/forums/topic/43912/ (дата звернення: 09.06.2026).

10. СУБД: які бувають та як правильно вибрати базу даних для проєкту. *Highload.tech*. URL: https://highload.tech/uk/subd-yaki-buvayut-yak-vibrati/ (дата звернення: 09.06.2026).

11. Побудова діаграм класів та моделювання статичної структури системи. *Studfile: методичні матеріали*. URL: https://studfile.net/preview/11372397/page:9/ (дата звернення: 09.06.2026).

12. Що таке багатошарова архітектура (N-Tier Architecture): переваги та безпека. *VPN Unlimited: help center*. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/n-tier-architecture (дата звернення: 09.06.2026).

13. Патерни проєктування: що таке шаблон та як його адаптувати. *Refactoring.Guru*. URL: https://refactoring.guru/uk/design-patterns/what-is-pattern (дата звернення: 09.06.2026).

14. UML-діаграми: класифікація, призначення та використання в інженерії ПЗ. *Evergreens: аналітичні статті*. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 09.06.2026).

15. QA Manual Tester Course: QA Engineer Training from Scratch in IT. *Sigma Software University*. URL: https://university.sigma.software/en/courses/qa-manual-tester-course/ (дата звернення: 09.06.2026).

16. What is Unit Testing? A Deep Dive Into Methodology, Tools, and Best Practices. *ZAPTEST: software automation blog*. URL: https://www.zaptest.com/what-is-unit-testing-a-deep-dive/ (дата звернення: 09.06.2026).

17. Що таке тестування UI (користувацького інтерфейсу) і як його проводити. *Блог IT-компанії Wezom*. URL: https://wezom.com.ua/ua/blog/testing-ui-user-interface (дата звернення: 09.06.2026).

18. Що таке Selenium WebDriver та які його переваги? *Блог QA Тестувальника: технічні статті QATestLab*. URL:

https://training.qatestlab.com/blog/technical-articles/selenium-webdriver/ (дата звернення: 09.06.2026).

19. Using JMeter in testing. *QATestLab Training Center: офіційний блог*. URL: https://en.training.qatestlab.com/blog/technical-articles/using-jmeter-in-testing/ (дата звернення: 16.06.2026).

20. Сокуренко В. В., Бандурка О. М. Безпека життєдіяльності та охорона праці : підручник. Харків : ХНУВС, 2021. 308 с.

21. Бедрій Я.І. Основи охорони праці : навч. посіб. 4-е вид. перероб. і доп. Тернопіль : Навчальна книга – Богдан, 2018. 240 с.

22. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. ДСанПіН 3.3.2.007-98. Затв. Постановою Головного державного санітарного лікаря України від 10.12.1998 № 7. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98 (дата звернення: 12.06.2026).

23. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями. НПАОП 0.00-7.15-18. Затв. Наказом Міністерства соціальної політики України від 14.02.2018 № 207. Чинний від 16.03.2018. URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 12.06.2026).

24. Методичні вказівки до виконання кваліфікаційної роботи бакалавра спеціальності 121 «Інженерія програмного забезпечення». – Тернопіль : ТНТУ ім. І. Пулюя, 2024.

ДОДАТКИ

ДОДАТОК А

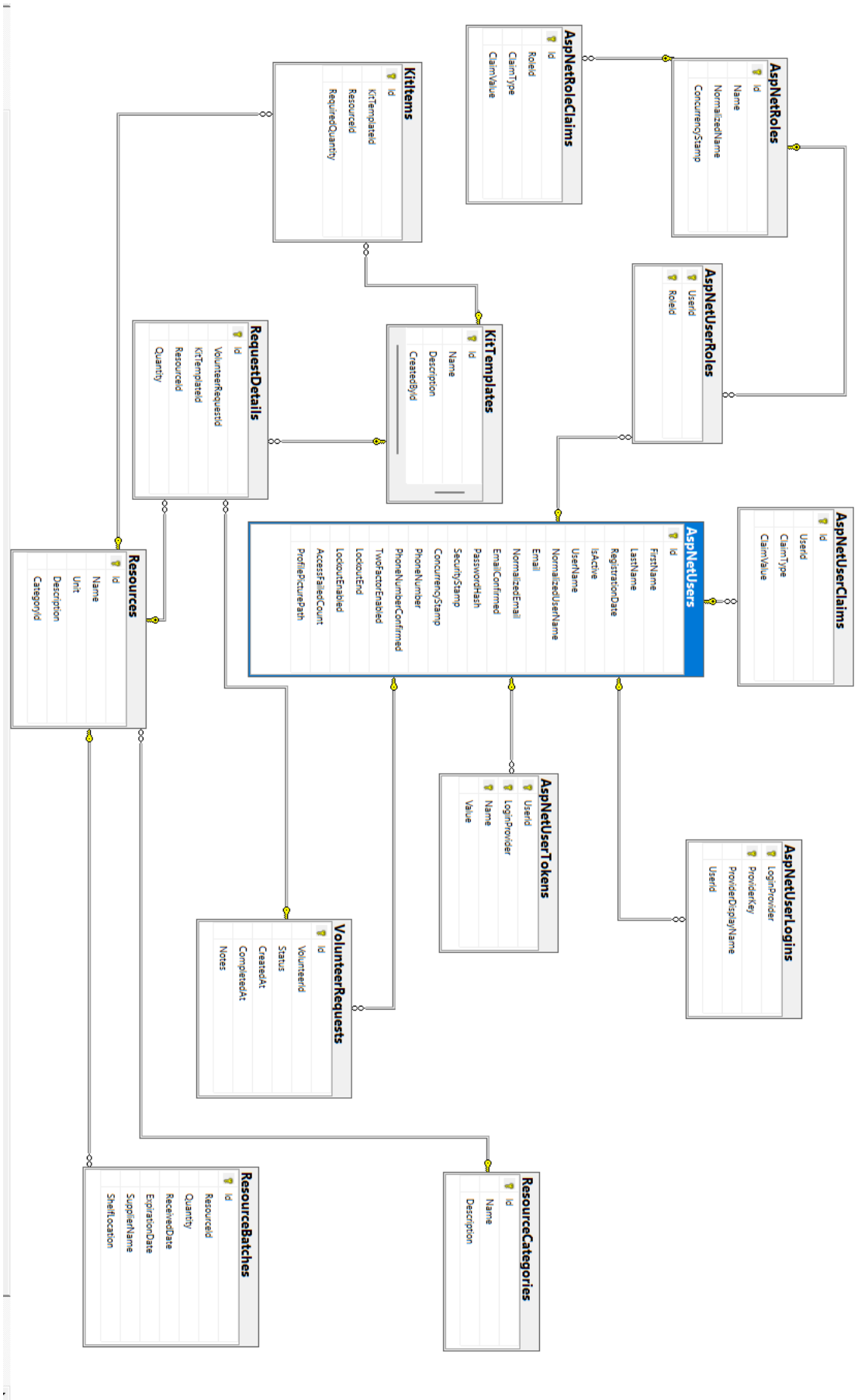


Рисунок А.1 – Схема створеної бази данних

ДОДАТОК Б

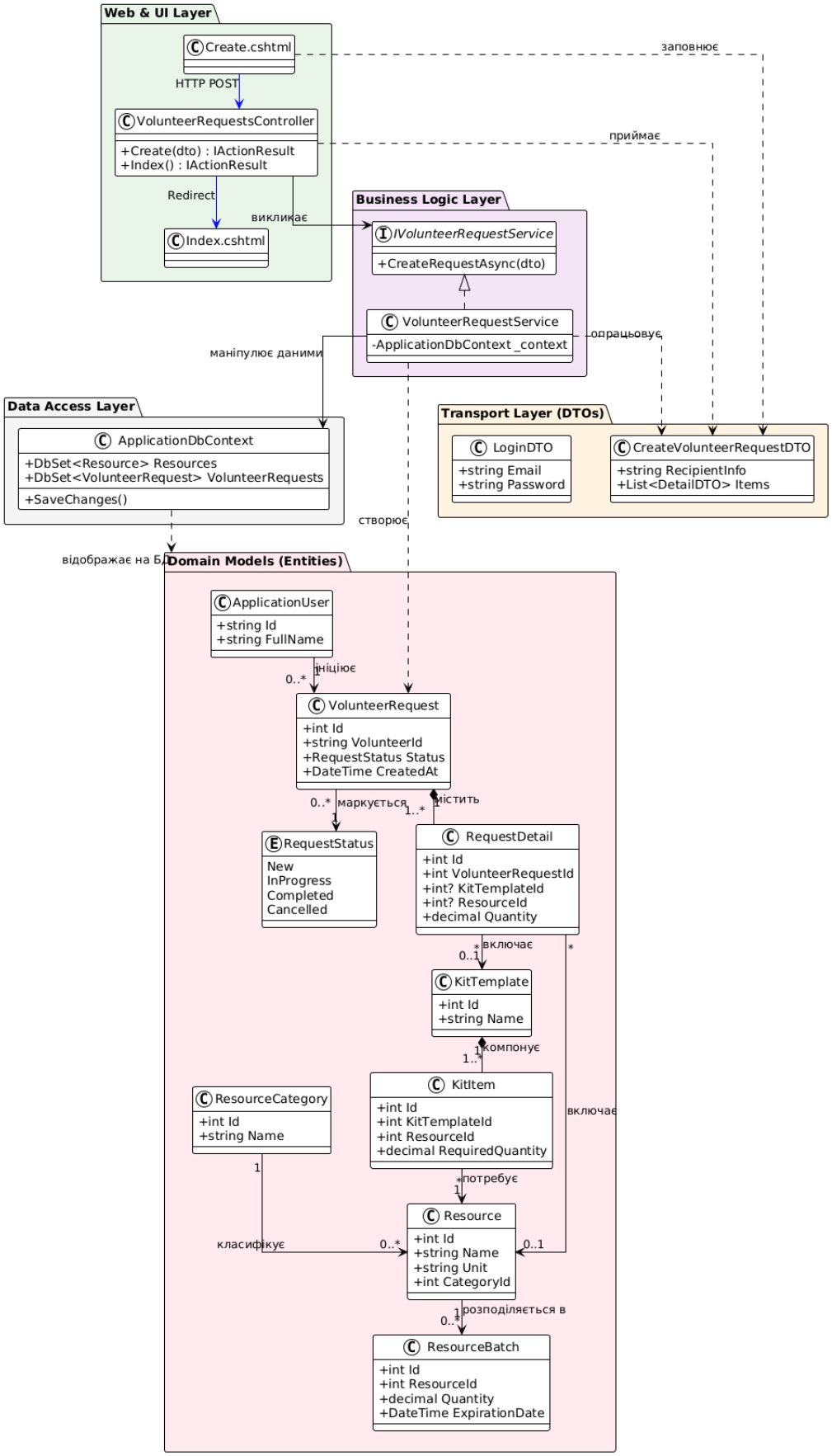


Рисунок Б.1 – Загальна UML-діаграма ієрархії класів інформаційної системи

ДОДАТОК В

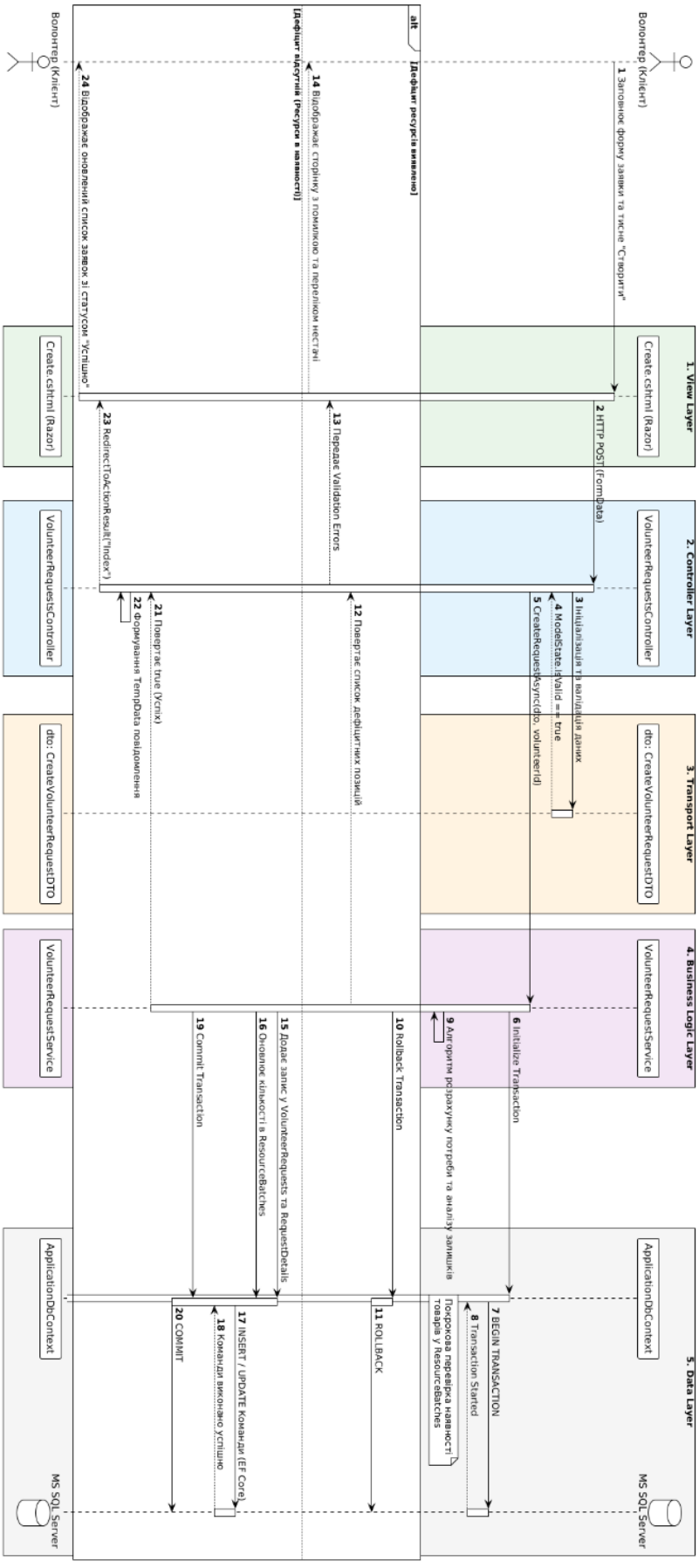


Рисунок В.1 – UML-діаграма послідовності для прецеденту створення заявки

ДОДАТОК Д

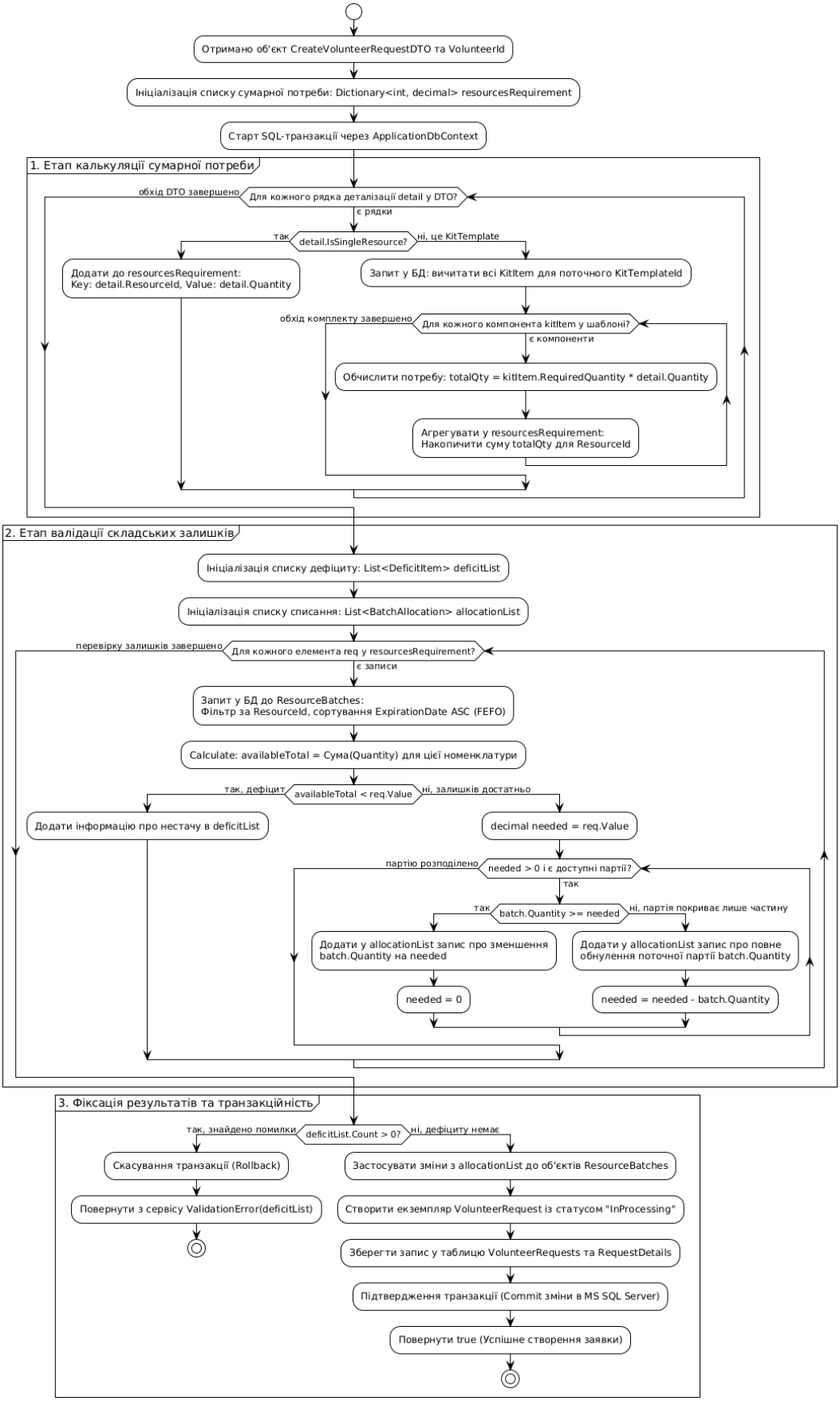


Рисунок Д.1 – UML-діаграми діяльності

ДОДАТОК Е

Лістинг ApplicationDbContext.cs

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using VolunteerSystem.Models;

namespace VolunteerSystem.Data
{

    public class ApplicationDbContext :
IdentityDbContext<ApplicationUser>
    {
        public
ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }
        public DbSet<ResourceCategory> ResourceCategories { get;
set; }
        public DbSet<Resource> Resources { get; set; }
        public DbSet<ResourceBatch> ResourceBatches { get; set; }


        public DbSet<KitTemplate> KitTemplates { get; set; }
        public DbSet<KitItem> KitItems { get; set; }


        public DbSet<VolunteerRequest> VolunteerRequests { get; set;
}
        public DbSet<RequestDetail> RequestDetails { get; set; }


        protected override void OnModelCreating(ModelBuilder
builder)
        {

            base.OnModelCreating(builder);


            builder.Entity<Resource>()
                .HasOne(r => r.Category)
                .WithMany(c => c.Resources)
                .HasForeignKey(r => r.CategoryId)
                .OnDelete(DeleteBehavior.Restrict);


            builder.Entity<ResourceBatch>()
                .HasOne(b => b.Resource)
                .WithMany(r => r.Batches)
                .HasForeignKey(b => b.ResourceId)
                .OnDelete(DeleteBehavior.Restrict);
```

```
            builder.Entity<KitItem>()
                .HasOne(ki => ki.KitTemplate)
                .WithMany(kt => kt.Items)
                .HasForeignKey(ki => ki.KitTemplateId)
                .OnDelete(DeleteBehavior.Cascade);


            builder.Entity<KitItem>()
                .HasOne(ki => ki.Resource)
                .WithMany()
                .HasForeignKey(ki => ki.ResourceId)
                .OnDelete(DeleteBehavior.Restrict);


            builder.Entity<RequestDetail>()
                .HasOne(rd => rd.VolunteerRequest)
                .WithMany(vr => vr.Details)
                .HasForeignKey(rd => rd.VolunteerRequestId)
                .OnDelete(DeleteBehavior.Cascade);


            builder.Entity<VolunteerRequest>()
                .HasOne(vr => vr.Volunteer)
                .WithMany()
                .HasForeignKey(vr => vr.VolunteerId)
                .OnDelete(DeleteBehavior.Restrict);
        }
    }
}
```

ДОДАТОК Ж

Лістинг VolunteerRequestsController

```
namespace VolunteerSystem.Controllers
{
    [Authorize(Roles = "Admin, Volunteer, Warehouse")]
    public class VolunteerRequestsController : Controller
    {
        private readonly IVolunteerRequestService _requestService;
        private readonly ApplicationDbContext _context;

        public VolunteerRequestsController(IVolunteerRequestService
requestService, ApplicationDbContext context)
        {
            _requestService = requestService;
            _context = context;
        }

        // 1. СПИСОК ЗАПИТІВ ВОЛОНТЕРА

        public async Task<IActionResult> Index(string searchString,
bool showCompleted = false, int page = 1)
        {
            int pageSize = 10; // Кількість замовлень на одну
сторінку

            // 1. Створюємо базовий запит (IQueryable).
            IQueryable<VolunteerRequest> query =
_context.VolunteerRequests
                .Include(r => r.Volunteer)
                .Include(r => r.Details).ThenInclude(d =>
d.Resource)
                .Include(r => r.Details).ThenInclude(d =>
d.KitTemplate)
                    .ThenInclude(k => k.Items).ThenInclude(i =>
i.Resource);

            // 2. Ізоляція даних
            if (!User.IsInRole("Admin") &&
!User.IsInRole("Warehouse"))
            {
                var volunteerId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
                query = query.Where(r => r.VolunteerId ==
volunteerId);
            }

            // 3. Фільтрація
            if (!showCompleted)
            {
                query = query.Where(r => r.Status !=
VolunteerSystem.Enums.RequestStatus.Completed);
```

```
            }

            // 4. Пошук
            if (!string.IsNullOrEmpty(searchString) &&
(User.IsInRole("Admin") || User.IsInRole("Warehouse")))
            {
                searchString = searchString.ToLower();
                query = query.Where(r =>
                    (r.Volunteer.FirstName != null &&
r.Volunteer.FirstName.ToLower().Contains(searchString)) ||
                    (r.Volunteer.LastName != null &&
r.Volunteer.LastName.ToLower().Contains(searchString)) ||
                    (r.Volunteer.Email != null &&
r.Volunteer.Email.ToLower().Contains(searchString))
                );
            }

            // 5. Сортування
            query = query.OrderByDescending(r => r.CreatedAt);

            // 6. ПАГІНАЦІЯ
            int totalItems = await query.CountAsync();
            int totalPages = (int)Math.Ceiling(totalItems /
(double)pageSize);


            var requests = await query
                .Skip((page - 1) * pageSize)
                .Take(pageSize)
                .ToListAsync();


            ViewBag.CurrentPage = page;
            ViewBag.TotalPages = totalPages;
            ViewBag.SearchString = searchString;
            ViewBag.ShowCompleted = showCompleted;

            return View(requests);
        }

        // 1. СТОРІНКА ОФОРМЛЕННЯ ЗАМОВЛЕННЯ
        public async Task<IActionResult> Create()
        {
            try
            {

                var currentUserId =
User.FindFirstValue(ClaimTypes.NameIdentifier);


                var kits = await _context.KitTemplates
                    .Include(k => k.Items).ThenInclude(i =>
i.Resource)
```

```
                    .Where(k => k.CreatedById == null ||
k.CreatedById == currentUserId)
                    .ToListAsync();

                var resources = await
_context.Resources.ToListAsync();

                ViewBag.KitsList = new SelectList(kits, "Id",
"Name");
                ViewBag.ResourcesList = new SelectList(resources,
"Id", "Name");

                return View(new CreateVolunteerRequestDTO());
            }
            catch (Exception)
            {

                TempData["ErrorMessage"] = "Помилка зв'язку з базою
даних. Спробуйте пізніше.";
                return RedirectToAction(nameof(Index));
            }
        }

        // 2. ОБРОБКА ЗБЕРЕЖЕННЯ ЗАМОВЛЕННЯ
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create(CreateVolunteerRequestDTO dto,
            int? selectedKitId, decimal? kitQuantity,
            int? selectedResourceId, decimal? resourceQuantity)
        {
            var volunteerId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            if (string.IsNullOrEmpty(volunteerId)) return
Unauthorized();


            if (selectedKitId.HasValue && kitQuantity.HasValue)
            {
                dto.Items.Add(new RequestItemDTO { KitTemplateId =
selectedKitId.Value, Quantity = kitQuantity.Value });
            }
            if (selectedResourceId.HasValue &&
resourceQuantity.HasValue)
            {
                dto.Items.Add(new RequestItemDTO { ResourceId =
selectedResourceId.Value, Quantity = resourceQuantity.Value });
            }

            if (!dto.Items.Any())
            {
                ModelState.AddModelError("", "Оберіть хоча б один
Пак або Ресурс для замовлення.");
```

```
            }
            else if (ModelState.IsValid)
            {

                try
                {
                    bool isCreated = await
_requestService.CreateRequestAsync(dto, volunteerId);

                    if (isCreated)
                    {
                        TempData["SuccessMessage"] = "Запит успішно
створено!";
                        return RedirectToAction(nameof(Index));
                    }
                    else
                    {
                        ModelState.AddModelError("", "КРИТИЧНИЙ
ДЕФІЦІТ: На складі недостатньо товарів для формування цього
замовлення!");
                    }
                }
                catch (Exception)
                {

                    ModelState.AddModelError("", "Сталася внутрішня
помилка сервера під час збереження замовлення. Спробуйте пізніше.");
                }
                // ========================================
            }

            var currentUserId =
User.FindFirstValue(ClaimTypes.NameIdentifier);
            var kits = await _context.KitTemplates
                .Include(k => k.Items).ThenInclude(i => i.Resource)
                .Where(k => k.CreatedById == null || k.CreatedById
== currentUserId)
                .ToListAsync();

            ViewBag.KitsList = new SelectList(kits, "Id", "Name");
            ViewBag.ResourcesList = new SelectList(await
_context.Resources.ToListAsync(), "Id", "Name");
            return View(dto);
        }

        [HttpPost]
        [Authorize(Roles = "Admin, Warehouse")]
        public async Task<IActionResult> ChangeStatus(int id,
VolunteerSystem.Enums.RequestStatus newStatus)
        {
            var request = await
_context.VolunteerRequests.FindAsync(id);
```

```
            if (request != null)
            {
                request.Status = newStatus;


                if (newStatus ==
VolunteerSystem.Enums.RequestStatus.Completed)
                {
                    request.CompletedAt = DateTime.UtcNow;
                }

                await _context.SaveChangesAsync();
            }

            return RedirectToAction(nameof(Index));
        }
    }


}
```

ДОДАТОК З

Лістинг тестування логіки системи

```
namespace VolunteerSystem.Tests
{
    // 1. ТЕСТИ ДЛЯ СЕРВІСУ УПРАВЛІННЯ ЗАЯВКАМИ
    public class VolunteerRequestServiceTests
    {
        private readonly ApplicationDbContext _context;
        private readonly VolunteerRequestService _requestService;

        public VolunteerRequestServiceTests()
        {
            var options = new
DbContextOptionsBuilder<ApplicationDbContext>()
                .UseInMemoryDatabase(Guid.NewGuid().ToString())
                .Options;

            _context = new ApplicationDbContext(options);
            _requestService = new VolunteerRequestService(_context);
        }

        [Fact]
        public async Task
CreateRequestAsync_WhenResourcesAreAvailable_ShouldReturnTrueAndComm
it()
        {
            // Arrange: Товар є на складі в достатній кількості
(Потреба: 10, Наявність: 15)
            var dto = new CreateVolunteerRequestDTO
            {
                Notes = "Тисменицький волонтерський штаб",
                Items = new List<RequestItemDTO> { new
RequestItemDTO { ResourceId = 1, Quantity = 10 } }
            };

            var fakeBatches = new List<ResourceBatch>
            {
                new ResourceBatch { Id = 1, ResourceId = 1, Quantity
= 15, ExpirationDate = DateTime.Now.AddDays(10) }
            };

            _context.ResourceBatches.AddRange(fakeBatches);
            await _context.SaveChangesAsync();

            var result = await
_requestService.CreateRequestAsync(dto, "volunteer-123");

            result.Should().BeTrue();
            _context.VolunteerRequests.Count().Should().Be(1);
        }
```

```
        [Fact]
        public async Task
CreateRequestAsync_WhenDeficitDetected_ShouldReturnFalseAndBlockSavi
ng()
        {
            var dto = new CreateVolunteerRequestDTO
            {
                Notes = "Червоний Хрест, склад №2",
                Items = new List<RequestItemDTO> { new
RequestItemDTO { ResourceId = 2, Quantity = 50 } }
            };

            var fakeBatches = new List<ResourceBatch>
            {
                new ResourceBatch { Id = 2, ResourceId = 2, Quantity
= 10, ExpirationDate = DateTime.Now.AddDays(5) }
            };

            _context.ResourceBatches.AddRange(fakeBatches);
            await _context.SaveChangesAsync();

            var result = await
_requestService.CreateRequestAsync(dto, "volunteer-123");

            // Assert: Перевірка блокування операції
            result.Should().BeFalse();
            _context.VolunteerRequests.Count().Should().Be(0);
        }
    }

    // 2. ТЕСТИ ДЛЯ СЕРВІСУ СКЛАДСЬКОГО ОБЛІКУ
    public class InventoryServiceTests
    {
        private readonly ApplicationDbContext _context;
        private readonly InventoryService _inventoryService;

        public InventoryServiceTests()
        {
            var options = new
DbContextOptionsBuilder<ApplicationDbContext>()
                .UseInMemoryDatabase(Guid.NewGuid().ToString())
                .Options;

            _context = new ApplicationDbContext(options);
            _inventoryService = new InventoryService(_context);
        }

        [Fact]
        public async Task
AddBatchAsync_WithValidExpirationDate_ShouldAddNewBatchSuccessfully(
)
        {
```

```
            // Arrange: Підготовка коректного DTO для
            var dto = new CreateResourceBatchDTO
            {
                ResourceId = 5,
                Quantity = 100,
                ExpirationDate = DateTime.Now.AddMonths(6)
            };

            // Act: Виклик методу оприбуткування на склад
            await _inventoryService.AddBatchAsync(dto);

            // Assert: Перевірка успішного додавання та збереження
партії
            _context.ResourceBatches.Count().Should().Be(1);
        }

        [Fact]
        public async Task
AddBatchAsync_WithExpiredDate_ShouldThrowArgumentExceptionAndBlockAd
ding()
        {
            // Arrange: Спроба оприбуткувати прострочений вантаж від
донора (-2 дні)
            var dto = new CreateResourceBatchDTO
            {
                ResourceId = 5,
                Quantity = 100,
                ExpirationDate = DateTime.Now.AddDays(-2)
            };

            // Act: Виклик методу бізнес-логіки
            Func<Task> act = async () => await
_inventoryService.AddBatchAsync(dto);

            // Assert: Перевірка виключення валідації
            await act.Should().ThrowAsync<ArgumentException>()
                     .WithMessage("Неможливо оприбуткувати партію
вантажу із вичерпаним терміном придатності.");

            _context.ResourceBatches.Count().Should().Be(0);
        }
    }
}
```

ДОДАТОК И

Лістинг тестування інтерфейсу системи

```
namespace VolunteerSystem.UITests
{
    public class VolunteerSystemUiTests : IDisposable
    {
        private IWebDriver _driver;
        private WebDriverWait _wait;
        private const string TargetUrl = "https://localhost:7181";
// Перевір порт
        private const int DefaultWaitSeconds = 15;

        public VolunteerSystemUiTests()
        {
            InitDriver();
        }

        private void InitDriver()
        {
            var configOptions = new ChromeOptions();
            configOptions.AddArgument("--start-maximized");
            configOptions.AddArgument("--ignore-certificate-
errors");
            configOptions.AddArgument("--disable-gpu");
            configOptions.AddArgument("--no-sandbox");
            configOptions.AddArgument("--disable-dev-shm-usage");
            configOptions.AddArgument("--remote-allow-origins=*");

            var service =
ChromeDriverService.CreateDefaultService();
            service.HideCommandPromptWindow = true;
            service.SuppressInitialDiagnosticInformation = true;

            _driver = new ChromeDriver(service, configOptions);
            _driver.Manage().Timeouts().ImplicitWait =
TimeSpan.FromSeconds(5);
            _wait = new WebDriverWait(_driver,
TimeSpan.FromSeconds(DefaultWaitSeconds));
        }

        private void RestartDriver()
        {
            try
            {
                _driver?.Quit();
                _driver?.Dispose();
            }
            catch { }
            InitDriver();
        }
```

```
        // ДОПОМІЖНІ МЕТОДИ
        private IWebElement WaitAndFind(By by, int timeoutSec =
DefaultWaitSeconds)
        {
            var wait = new WebDriverWait(_driver,
TimeSpan.FromSeconds(timeoutSec));
            return wait.Until(d =>
            {
                var element = d.FindElement(by);
                if (element != null && element.Displayed &&
element.Enabled) return element;
                return null!;
            });
        }

        private void PerformLoginBeforeTest()
        {

_driver.Navigate().GoToUrl($"{TargetUrl}/Account/Login");

            var email = WaitAndFind(By.Name("Email"));
            email.Clear();
            email.SendKeys("admin@test.com");

            var pass = WaitAndFind(By.Name("Password"));
            pass.Clear();
            pass.SendKeys("Admin_123");

            pass.Submit();

            new WebDriverWait(_driver,
TimeSpan.FromSeconds(DefaultWaitSeconds))
                .Until(d => !d.Url.Contains("Login"));
        }

        // ТЕСТИ

        [Fact]
        public void Test_1_Admin_CanCreateNewUser_Flow()
        {
            try { PerformLoginBeforeTest(); }
            catch
            {
                RestartDriver();
                PerformLoginBeforeTest();
            }

            _driver.Navigate().GoToUrl($"{TargetUrl}/Users/Create");


WaitAndFind(By.Name("email")).SendKeys("new.volunteer@system.com");
```

```
            var passwordInput = WaitAndFind(By.Name("password"));
            passwordInput.SendKeys("SecurePass123!");

            var roleSelect = WaitAndFind(By.Name("selectedRole"));
            var roleOption =
roleSelect.FindElements(By.TagName("option")).FirstOrDefault(o =>
!string.IsNullOrWhiteSpace(o.GetAttribute("value")));
            if (roleOption != null) roleOption.Click();

            passwordInput.Submit();

            new WebDriverWait(_driver,
TimeSpan.FromSeconds(DefaultWaitSeconds))
                .Until(d => d.Url.Contains("/Users") ||
d.PageSource.Contains("new.volunteer@system.com"));


_driver.PageSource.Should().Contain("new.volunteer@system.com");
        }

        [Fact]
        public void Test_2_VolunteerRequest_Creation_Flow()
        {
            PerformLoginBeforeTest();


_driver.Navigate().GoToUrl($"{TargetUrl}/VolunteerRequests/Create");

            WaitAndFind(By.Name("Notes")).SendKeys("Термінова
доставка. Медикаменти.");

            var resourceDropdown =
WaitAndFind(By.Name("selectedResourceId"));
            resourceDropdown.Click();
            var resourceOpt =
resourceDropdown.FindElements(By.TagName("option")).FirstOrDefault(o
=> !string.IsNullOrWhiteSpace(o.GetAttribute("value")));
            if (resourceOpt == null) throw new
InvalidOperationException("No resource options available to
select");
            resourceOpt.Click();

            var quantity = WaitAndFind(By.Name("resourceQuantity"));
            quantity.Clear();
            quantity.SendKeys("25");

            quantity.Submit();

            new WebDriverWait(_driver,
TimeSpan.FromSeconds(DefaultWaitSeconds)).Until(d =>
d.Url.Contains("/VolunteerRequests"));
```

```
            _driver.PageSource.Should().Contain("Запит успішно
створено");
        }

        [Fact]
        public void Test_3_Inventory_AddBatch_Flow()
        {
            PerformLoginBeforeTest();


_driver.Navigate().GoToUrl($"{TargetUrl}/Inventory/Create");

            var resourceSelect = WaitAndFind(By.Name("ResourceId"));
            resourceSelect.Click();
            var resOpt =
resourceSelect.FindElements(By.TagName("option")).FirstOrDefault(o
=> !string.IsNullOrWhiteSpace(o.GetAttribute("value")));
            if (resOpt == null) throw new
InvalidOperationException("No resources available to add batch");
            var resName = resOpt.Text;
            resOpt.Click();

            WaitAndFind(By.Name("Quantity")).Clear();
            WaitAndFind(By.Name("Quantity")).SendKeys("100");

            var expDate = WaitAndFind(By.Name("ExpirationDate"));
            expDate.Clear();
            expDate.SendKeys("2026-12-31");

            expDate.Submit();

            new WebDriverWait(_driver,
TimeSpan.FromSeconds(DefaultWaitSeconds)).Until(d =>
d.Url.Contains("/Inventory"));
            _driver.PageSource.Should().Contain(resName);
        }

        [Fact]
        public void Test_4_Resources_CategoryFilter_Flow()
        {
            PerformLoginBeforeTest();


_driver.Navigate().GoToUrl($"{TargetUrl}/Resources/Index");

            var firstItemName = WaitAndFind(By.CssSelector("table
tbody tr td:first-child, .card h5")).Text;

            var search = WaitAndFind(By.Name("searchString"));
            search.Clear();
            search.SendKeys(firstItemName); // Шукаємо те, що
реально є в базі
```

```
            search.Submit();

            System.Threading.Thread.Sleep(1000); // Секунда на
відмальовку результатів
            _driver.PageSource.Should().Contain(firstItemName);
        }

        [Fact]
        public void Test_5_Admin_KitTemplate_Creation_Flow()
        {
            PerformLoginBeforeTest();

            // 1. Переходимо безпосередньо на сторінку створення
нового шаблону

_driver.Navigate().GoToUrl($"{TargetUrl}/KitTemplates/Create");

            // 2. Шукаємо поле для назви набору
            var nameInput = WaitAndFind(By.Name("Name"));
            nameInput.Clear();

            string uniqueKitName = $"Тестовий набір
{DateTime.Now.Ticks}";
            nameInput.SendKeys(uniqueKitName);

            try
            {
                var descInput =
_driver.FindElement(By.Name("Description"));
                descInput.SendKeys("Автоматично згенерований набір
для UI тестів");
            }
            catch { /* Ігноруємо помилку, якщо поля опису немає в
HTML */ }

            // 3. Відправляємо форму
            nameInput.Submit();

            // 4. Чекаємо повернення на сторінку списку (Index)
            new WebDriverWait(_driver,
TimeSpan.FromSeconds(DefaultWaitSeconds))
                .Until(d => d.Url.Contains("/KitTemplates"));

            // 5. Перевіряємо, чи успішно зберігся наш новий набір
            _driver.PageSource.Should().Contain(uniqueKitName);
        }

        public void Dispose()
        {
            try { _driver?.Quit(); } catch { }
            try { _driver?.Dispose(); } catch { }
        }
    }
```

ДОДАТОК К

Посилання на репозиторій GitHub

https://github.com/sshptrk/VolonterSystem

Рисунок К.1 – QR-код для репозиторію на GitHub.