**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**

| | |
|---|---|
| **ПОГОДЖЕНО** | **ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** |
| **Декан факультету** | **Завідувач кафедри комп'ютерних наук** |
| _____ **Ігор БОЛБОТ** | _____ **Белла ГОЛУБ** |
| (підпис) | (підпис) |
| «___» _____ 2026 р. | «___» _____ 2026 р. |

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Фронтенд частина системи обліку взаємодії волонтерів і військових»**
Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

**Гарант освітньої програми**
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

**Керівник бакалаврської кваліфікаційної роботи** _____ **Матвій ГОРБАЧ**
(підпис)

**Консультант бакалаврської кваліфікаційної роботи** _____ **Белла ГОЛУБ**
к.т.н., доцент (підпис)

**Виконав** _____ **Олександр БАЧИНСЬКИЙ**
(підпис)

**Київ-2026**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри комп'ютерних наук**

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

# ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Бачинському Олександру Анатолійовичу
(прізвище, ім'я, по-батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи

«Фронтенд частина системи обліку взаємодії волонтерів і військових»

затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): опис організації та змісту взаємодії волонтерів і військових

Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області
2. Проєктування інформаційної системи
3. Розробка програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації

Дата видачі завдання «19» грудня 2025 р.

**Керівник бакалаврської кваліфікаційної роботи** _____ **Матвій ГОРБАЧ**
(підпис)

**Завдання взяв до виконання** _____ **Олександр БАЧИНСЬКИЙ**
(підпис)

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

**API –** Application Programming Interface

**CSS –** Cascading Style Sheets

**CRUD –** Create Read Update Delete

**DOM –** Document Object Model

**HTML –** HyperText Markup Language

**IP –** Internet Protocol

**CSS –** Cascading Style Sheets

**JWT –** JSON Web Token

**HTTP –** HyperText Transfer Protocol

**HTTPS –** HyperText Transfer Protocol Secure

**JS –** JavaScript

**JSON –** JavaScript Object Notation

**JWT –** JSON Web Token

**REST –** Representational State Transfer

**SQL –** Structured Query Language

**TCP –** Transmission Control Protocol

**UI –** User Interface

**UML –** Unified Modeling Language

**UX –** User Experience

**XSS –** Cross-Site Scripting

**СУБД –** Система Управління Базами Даних

# ВСТУП

**Актуальність.** В сучасних умовах нашої країни, дуже важливими є облік та ефективна координація взаємодії між військовими підрозділами та волонтерськими організаціями. Існує критична необхідність максимально швидкого реагування на поточні запити сил оборони, а також забезпечення прозорості під час їх виконання. Наразі такі процеси здебільшого відбуваються через різноманітні месенджери або соціальні мережі, що може призводити до дублювання запитів, ускладнення верифікації обох сторін взаємодії, що в свою чергу значно підвищує ризики шахрайства. Всі ці проблеми спричинені відсутністю централізованої системи, яка б дозволила забезпечити швидкий доступ до необхідної структурованої інформації та прозорість всіх процесів взаємодії між військовими і волонтерами за допомогою ведення обліку та обов'язкового звітування обох сторін.

**Мета розробки.** Метою цієї бакалаврської роботи є розробка фронтенд-частини єдиної системи для обліку та координації взаємодії волонтерів і військових. Розроблювана вебсистема призначена для пришвидшення та полегшення відповідних процесів співпраці зацікавлених сторін, за допомогою створення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів з будь-яким рівнем технічних навичок, а також забезпечення високої продуктивності та адаптивності для стабільної роботи з системою на різних типах пристроїв, зокрема у складних умовах в яких перебувають військовослужбовці.

**Методи та технології.** Попереднє проєктування UX/UI дизайну та створення інтерактивного прототипу було здійснено за допомогою інструментарію Figma. Під час розробки програмної системи було використано компонентний підхід, який надав можливість спростити розробку та подальше масштабування. Фронтенд частина розроблена з використанням базових веб-технологій, HTML5 для семантичної верстки, CSS3 для стилізування та адаптивності за допомогою Flexbox та CSS Grid та мови програмування JavaScript для динамічності. Було прийнято рішення не використовувати важкі

фреймворки, для забезпечення максимальної продуктивності. Автентифікація та безпека передачі даних здійснюється за допомогою технології JWT, санітизація введених даних запобігає XSS-атакам, а для обміну даними з серверною частиною використовується Fetch API для надсилання асинхронних запитів.

**Апробація програмного продукту.** Результати розробки були опубліковані у вигляді тез на тему «Фронтенд частина системи обліку взаємодії волонтерів і військових» на VII Всеукраїнській науково-практичній конференції студентів та аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем '2026»

**Структура записки.** Обсяг записки становить 74 сторінки, 28 джерел та 4 додатки. Робота складається із чотирьох розділів:

- Перший розділ містить аналіз та моделювання предметної області, розгляд існуючих аналогів та здійснення постановки завдання;
- Другий розділ стосується процесу проєктування інформаційної системи, опису вебзастосунку, його складових та архітектури, логічної організації системи, а також механізмів взаємодії та обміну даними;
- Третій розділ присвячений розробці програмного забезпечення. У ньому виконано моделювання структури та взаємодії об'єктів, описано компонентну структуру системи, обґрунтовано вибір інструментів та технологій розробки, а також наведено алгоритми роботи основних програмних модулів;
- Четвертий розділ містить рекомендації щодо впровадження та експлуатації системи, опис процесу тестування, визначення вимог до апаратного та програмного забезпечення, а також формування інсталяційного пакет.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис взаємодії волонтерів і військових як предметної області

Взаємодія волонтерів і військових охоплює багато різноманітних процесів, які є дуже поширеними та важливими для нашої держави в сучасних умовах. Основними з них є комунікація, формування та опрацювання запитів на матеріальну чи фінансову допомогу, логістика та звітування про виконану роботу [1]. У цих процесах беруть участь військові підрозділи, які потребують допомоги та формують запити, волонтерські організації або окремі волонтери, які займаються збором та передачею необхідних ресурсів, контролюючі органи, які здійснюють перевірку учасників та моніторинг виконуваних процесів, а також фізичні особи, які добровільно долучаються до зборів коштів.

Одним із найважливіших етапів взаємодії військових і волонтерів є створення та отримання запитів на допомогу. Військовий підрозділ, у якого виникають потреби у певних ресурсах, формує запит, в якому мають детально описуватись потреби, кількість та терміни виконання, після чого розповсюджує його через різні канали зв’язку. Волонтери, у свою чергу, побачивши запит, який вони мають можливість виконати, надсилають відповідь і розпочинають комунікацію.

Опрацювання запиту волонтером передбачає перевірку його актуальності, верифікацію військового підрозділу, встановлення зворотного зв’язку для уточнення деталей потреб та логістичних питань, далі волонтер оцінює власні можливості і, за потреби, організовує додатковий збір коштів та здійснює пошук постачальників. Наступними кроками є закупівля ресурсів, перевірка їх якості та відповідності вимогам запиту, а також організація безпечного транспортування до пункту призначення. Важливою частиною є прозорість, кожен з вищезазначених етапів має фіксуватись, для запобігання шахрайству чи недоброчесності.

Звітування є останнім, але не менш важливим етапом взаємодії. Після надання допомоги волонтери повинні створити звіт, який містить детальний опис переданих ресурсів, копії пов’язаних документів тощо. Військовий підрозділ, у свою чергу, після отримання ресурсів формує підтвердження у вигляді звіту з детальним описом та фото. Створені звіти надсилаються контролюючому органу та перевіряються на відповідність отриманої допомоги первинному запиту. При виявленні розбіжностей, здійснюється запит пояснень до обох сторін. Звітування надає можливість забезпечити повну прозорість взаємодії та мінімізувати корупційні ризики.

Зазвичай розповсюдження зборів коштів та запитів на допомогу відбувається через багато різних соціальних мереж та месенджерів. Такий підхід сильно ускладнює пошук, верифікацію та обробку потрібної інформації. Для пришвидшення цих процесів та уникнення дублювання, важливим є забезпечення доступу у режимі реального часу до актуальної та структурованої інформації з можливістю швидкого пошуку.

Вся інформація про збори коштів, запити на допомогу та їх виконання у формі звітів має обліковуватись та зберігатись у відкритому доступі. Це дозволяє забезпечити повну прозорість взаємодії військових та волонтерів, для уникнення випадків корупції, шахрайства та нецільового використання коштів або ресурсів. Таким чином гарантується безпека обох сторін, а також підвищується рівень суспільної довіри до роботи волонтерських організацій та військових підрозділів. Також, відкритий доступ до даних дозволяє проводити статистичний аналіз ефективності роботи волонтерів та виявляти найбільш критичні потреби військових.

Отже, взаємодія волонтерів і військових – це велика кількість взаємопов’язаних процесів, ефективність яких безпосередньо залежить від швидкості пошуку та обміну інформацією, а також надійності та прозорості співпраці обох сторін. Саме тому виникає потреба у створенні централізованої системи, яка надасть можливість військовим та волонтерам розповсюджувати інформацію про актуальні збори коштів та запити на допомогу, забезпечить

швидкий та структурований доступ до неї та автоматизує її облік. Це дозволить уникнути поточних проблем, які виникають через використання розрізнених соціальних мереж та месенджерів для взаємодії.

## 1.2 Моделювання предметної області

Моделювання предметної області – це один із ключових аспектів для розробки якісного та ефективного програмного продукту, який базується на побудові моделей для візуалізації, аналізу та проєктування інформаційних систем. Модель визначає та детально описує її основні елементи, їхні характеристики та взаємозв’язки, принципи взаємодії, логіку процесів тощо. Така візуалізація набагато краще сприймається замовниками та значно полегшує взаємодію між ними та розробниками. Це дозволяє пришвидшити процес аналізу та визначення вимог до майбутньої системи та зменшити кількість непорозумінь чи помилок під час розробки.

Для предметної області взаємодії волонтерів та військових за допомогою моделювання важливо визначити основних акторів та які функції вони можуть виконувати, описати послідовність передачі інформації між ними, а також покроково деталізувати, як виконуватимуться ключові процеси.

Для моделювання вищенаведених аспектів використовується мова UML (Unified Modeling Language) [2]. Вона надає можливість візуалізувати та спроєктувати архітектуру майбутньої системи за допомогою відповідних діаграм, кожна з яких відображає певну сторону предметної області. Далі в розділі будуть детально описані та представлені: діаграма прецедентів, діаграма послідовності та діаграма активності.

**1.2.1 Діаграма прецедентів.** На початку моделювання одною із найважливіших є діаграма прецедентів, адже вона дозволяє описати основних акторів взаємодії, що саме вони зможуть виконувати і які результати отримати у межах предметної області [3]. Вона використовується для визначення загальних вимог до поведінки майбутньої системи, фокусуючись на потребах користувачів різних ролей.

Побудова цієї діаграми на ранніх етапах проєктування надає можливість чітко описати контекст та межі предметної області. Графічна візуалізація полегшує документацію та взаємодію із замовниками для узгодження ключових сценаріїв роботи. Діаграма прецедентів слугує основою для подальшої деталізації предметної області.

Основними компонентами діаграми є:

- Актори – це сутності для відображення зовнішніх ролей, які можуть виконувати процеси в межах предметної області. Актором може бути людина, організація, технічний засіб тощо, що має свій чіткий перелік функцій та цілей;
- Прецеденти – відображають окремі процеси або функції, які можуть виконуватись акторами для досягнення певного результату. Кожен прецедент описує конкретну закінчену дію;
- Зв’язки – це логічні залежності, які відображають взаємодію певного актора з відповідним прецедентом. Окрім звичайної асоціації, в діаграмі прецедентів існують наступні типи відношень:
    - Включення (include) – показує те, що базовий прецедент обов’язково включає в себе функціонал іншого;
    - Розширення (extend) – описує додатковий функціонал прецеденту, який не є обов’язковим та виконується лише за певних умов;
    - Узагальнення – вказує, що між акторами або прецедентами може існувати ієрархія, де дочірній елемент успадковує властивості та поведінку батьківського.

Для предметної області взаємодії волонтерів і військових було побудовано діаграму прецедентів (рис. 1), на якій зображені основні актори та прецеденти, а також зв’язки між ними.

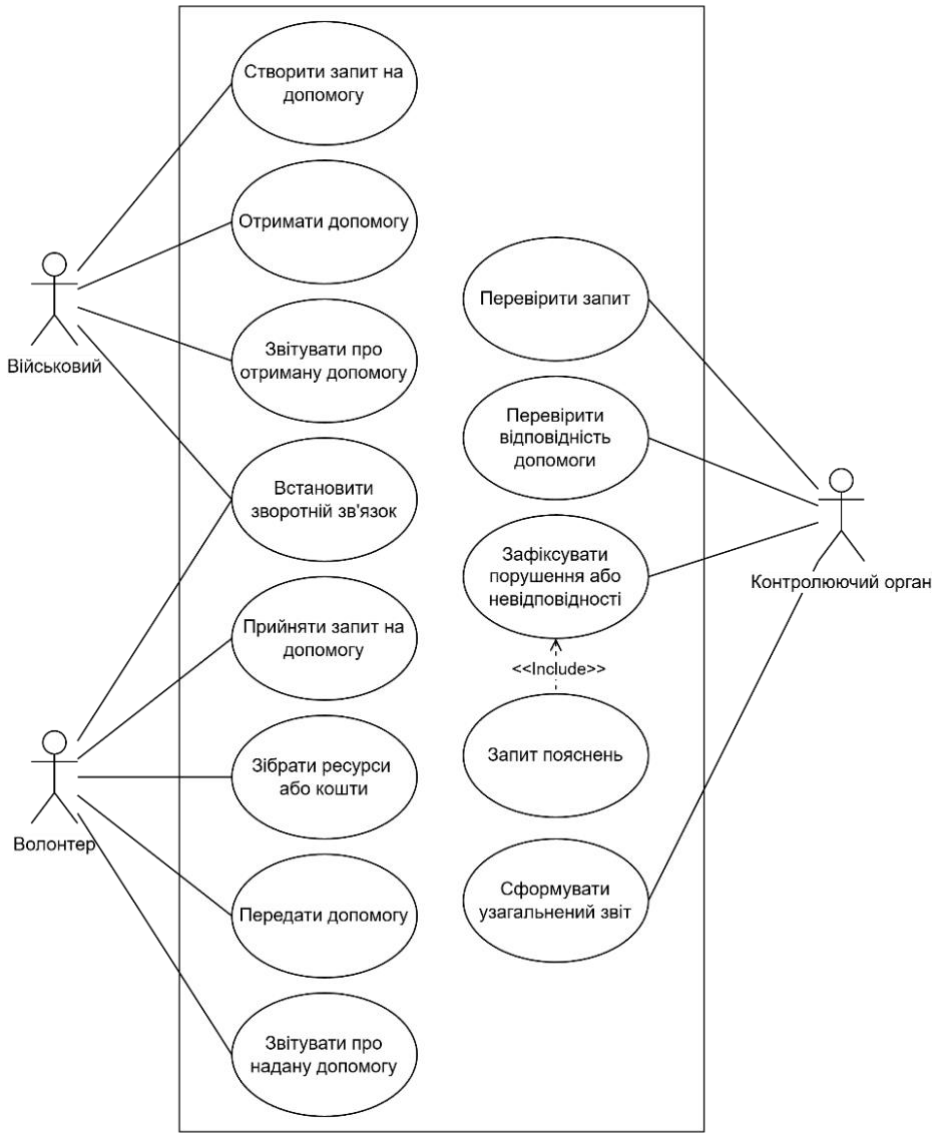



Рис. 1 Діаграма прецедентів

На діаграмі виділено три основних актори, які забезпечують виконання всіх потрібних процесів предметної області:

- Військовий – актор взаємодії, який формулює запити на допомогу згідно з потребами підрозділу, спілкується з волонтером для узгодження питань, а також отримує допомогу та звітує про це.
- Волонтер – опрацьовує та приймає бажані запити військових, встановлює зв’язок з ними, збирає ресурси, забезпечує логістику та звітує про виконану роботу.
- Контролюючий орган – виконує роль наглядача, забезпечує верифікацію даних, безпечність та прозорість взаємодії, перевіряє звіти та виявляє можливі порушення.

Кожен актор має певний перелік прецедентів, які він може виконувати.

Військовий:

- Створити запит на допомогу – формування запиту із детальним описом потреб, зазначенням кількості необхідних ресурсів та терміновості виконання запиту;
- Встановити зворотний зв’язок (з волонтером) – комунікація з волонтером для відстеження виконання запиту, а також уточнення деталей допомоги та логістики;
- Отримати допомогу – фізичне надходження та отримання ресурсів зазначених в запиті;
- Звітувати про отриману допомогу – створення детального звіту по отриманим ресурсам та їх відповідності вимогам запиту з фотопідтвердженням.

Волонтер:

- Прийняти запит на допомогу – перегляд актуальних запитів та прийняття їх до виконання;
- Встановити зворотний зв’язок (з військовим) – узгодження деталей потрібної допомоги, логістики та надання інформації про хід виконання запиту за потребою військового;
- Зібрати ресурси або кошти – організація закупівлі ресурсів або створення благодійних зборів для закриття потреб запиту;
- Передати допомогу – здійснення відправки або транспортування ресурсів військовому підрозділу у зазначене місце та час;
- Звітувати про надану допомогу – створення звіту після надання допомоги, з детальним описом виконаної роботи та фото підтвердженням.

Контролюючий орган:

- Перевірити запит – здійснення перевірки запиту на предмет підозрілості, дублювання, а також верифікація автора;

- Перевірити відповідність допомоги – зіставлення даних у звітах волонтера та військового для перевірки відповідності отриманої допомоги початковому запиту;
- Зафіксувати порушення або невідповідності – фіксування порушень або розбіжностей у наданих звітах. Прецедент включає в себе (за допомогою зв’язку <<include>>) обов’язковий запит пояснень у обох сторін для з’ясування та вирішення проблем;
- Сформувати узагальнений звіт – формування загального документу по конкретному запиту на основі отриманих звітів від волонтера та військового.

Побудована діаграма прецедентів чітко відображає ролі, відповідальності та можливості учасників процесу, а також взаємодії між ними в межах предметної області. Вона стає гарною основою для подальшого проєктування системи.

**1.2.2 Діаграма послідовності.** На відміну від діаграми прецедентів, яка визначає акторів та їхні можливості, діаграма послідовності відображає безпосередню взаємодію між об’єктами системи в динаміці. Вона зосереджена на тому, як саме актори співпрацюють між собою в часі для виконання відповідних процесів. На цій діаграмі взаємодія зображується як обмін повідомленнями між об’єктами, що дозволяє більш детально описати логіку конкретних сценаріїв.

Дана діаграма використовується для уточнення діаграми прецедентів, перетворюючи статичні дії на послідовні кроки. В ній чітко визначається час передачі інформації, послідовність подій та тривалість активності кожного об’єкта. Вона дозволяє спроєктувати логіку обміну даними в межах предметної області.

Діаграма послідовності складається з наступних компонентів:

- Об’єкт або актор – учасники які відповідають за ініціювання або прийняття повідомлень;
- Лінія життя об’єкта – відображає часовий період існування об’єкта;

- Фокус управління – позначає період часу протягом якого об’єкт безпосередньо виконує певну функцію;
- Повідомлення – відображає передачу інформації або дію від одного об’єкта до іншого;
- Реакція на повідомлення – відображає повернення результату виконаної дії;
- Завершення існування об’єкта – вказує на припинення активності об’єкта.

Далі наведено розроблену діаграму послідовності (рис. 2) для предметної області, яка деталізує процес отримання, опрацювання та виконання запиту на допомогу, а також звітування після його виконання з кінцевою перевіркою та формуванням загального звіту.

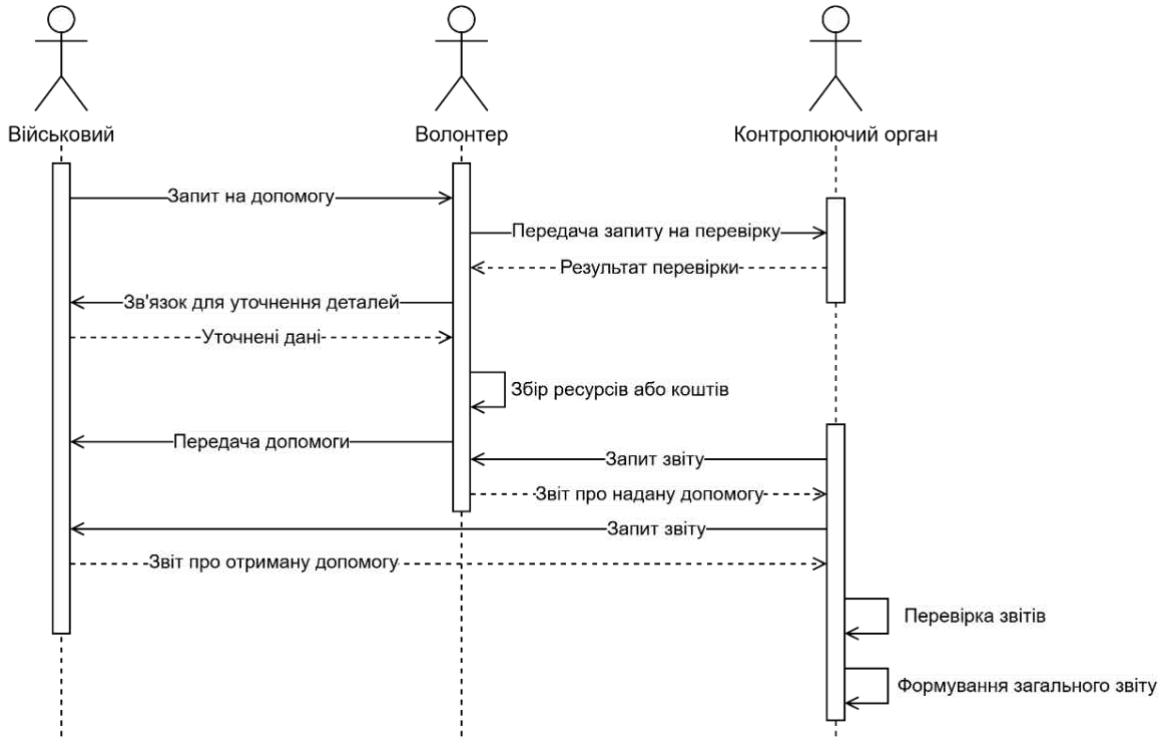



Рис. 2 Діаграма послідовності

Опис акторів:

- Військовий – ініціатор взаємодії, який надсилає вхідний запит, надає уточнюючі дані волонтеру, отримує допомогу та звітує;

- Волонтер – актор, який відповідає за опрацювання запиту, надсилає його на перевірку, встановлює зв’язок з військовим для уточнення деталей та відповідає за збір і передачу допомоги.
- Контролюючий орган – гарантує безпеку та прозорість взаємодії, здійснює верифікацію запиту, перевірку виконаної роботи та створення узагальненого звіту.

Опис сценарію взаємодії:

1. Отримання та перевірка запиту:
   - Військовий надсилає запит на допомогу волонтеру;
   - Волонтер, отримавши запит, переадресовує його контролюючому органу для перевірки;
   - Контролюючий орган опрацьовує дані та повертає результат перевірки.
2. Комунікація та виконання запиту:
   - Волонтер встановлює зворотний зв’язок з військовим для уточнення деталей запиту та логістики;
   - Військовий надає відповідь з уточненими даними;
   - Волонтер починає збір ресурсів або коштів відповідно потребам та можливостям;
   - Після закінчення збору необхідних ресурсів волонтер передає їх військовому шляхом транспортування у зазначене місце.
3. Звітування:
   - Контролюючий орган робить запит звітів у обох сторін взаємодії після завершення попередніх етапів;
   - Волонтер надсилає сформований звіт про надану допомогу, що містить детальну інформацію про витрати та виконану роботу;
   - Військовий надсилає звіт про отриману допомогу з фотопідтвердженням;
   - Контролюючий орган перевіряє отримані звіти на наявність порушень чи невідповідностей;

- Якщо ніяких проблем не виявлено, контролюючий орган формує загальний звіт по виконанню запиту на допомогу.

Побудована діаграма послідовності дозволила детально зобразити всі етапи виконання запиту на допомогу в часі та визначати чітку послідовність обміну повідомленнями між акторами. Це надає чітке розуміння динаміки процесів, що є важливим для правильного подальшого проєктування.

**1.2.3 Діаграма активності.** Діаграма активності використовується для моделювання динамічних аспектів під час взаємодії основних суб'єктів [4]. Вона зосереджена на зображенні алгоритмів та логіки виконання процесів, а саме візуалізує послідовність дій та переходів між ними, що відбуваються в межах предметної області.

Ця діаграма дає можливість детально описувати складні процеси, де можуть виникнути розгалуження за певних умов чи паралельні операції, шляхом розбиття взаємодії на зрозумілі кроки

Основні компоненти діаграми активності:

- Початковий вузол – позначає точку входу, з якої починається виконання всього процесу;
- Дія – описує конкретну операцію або крок, що виконується актором;
- Потік управління – стрілка, що вказує напрямок переходу від однієї дії до іншої;
- Об'єкт – дані або матеріальні ресурсі, які створюються, змінюються або передаються в процесі взаємодії;
- Вузол прийняття рішення – дозволяє спрямувати процес за певним сценарієм залежно від результату попереднього рішення або дії;
- Розгалуження та злиття – відображають паралельні процеси, що відбуваються одночасно;
- Доріжки – використовуються для групування дій відповідальних акторів у окремі зони, що дає можливість чітко бачити, хто виконує конкретну частину процесу;
- Фінальний вузол – позначає логічне завершення виконання процесу.

Для детального та покрокового відображення життєвого циклу взаємодії акторів у межах предметної області було розроблено діаграму активності, яку представлено у Додатку А. Вона візуалізує повний процес опрацювання запиту на допомогу: від моменту його формування військовим до створення загального звіту контролюючим органом.

Опис алгоритму діаграми:

- Військовий формує та надсилає запит на допомогу до волонтера;
- Волонтер приймає запит та передає його на перевірку контролюючому органу;
- Контролюючий орган здійснює перевірку запиту та верифікацію автора:
    - Якщо перевірка невдала, військовий повідомляється про помилку і має знову сформувати запит;
    - Якщо перевірка вдала, волонтер встановлює зворотний зв’язок з військовим для уточнення деталей.
- Військовий здійснює уточнення інформації;
- Волонтер розпочинає збір ресурсів або коштів;
- Після завершення збору, виконує передачу допомоги;
- Після передачі процес розділяється на паралельні дії:
    - Волонтер звітує про виконану роботу;
    - Військовий отримує допомогу та також надає звіт.
- Контролюючий орган отримує звіти та здійснює їх перевірку:
    - Якщо дані не збігаються, здійснюється запит пояснень до волонтера та військового;
    - Якщо дані збігаються контролюючий орган формує загальний звіт по запиту.

За допомогою діаграми активності було зображено повний життєвий цикл опрацювання запиту, з урахуванням розгалужень та паралельних операцій під час виконання процесу.  Її використання забезпечило чітке розуміння всіх

можливих сценаріїв розвитку подій, включаючи обробку помилок та виняткових ситуацій.

## 1.3 Огляд існуючих рішень

Аналіз існуючих аналогів є надзвичайно важливим етапом проєктування, він дозволяє визначити недоліки існуючих систем, потреби користувачів та сформувати вимоги до майбутньої системи, щоб забезпечити високу ефективність взаємодії та уникнути поширених проблем інших продуктів.

Наразі в Україні існує чимало різних цифрових рішень для організації волонтерської допомоги та забезпечення військових потреб. Найпоширенішими з таких інструментів, є різні фандрейзингові платформи та звичайні соціальні мережі або месенджери для розповсюдження відповідної інформації.

**People`s Project (Всеукраїнський центр волонтерів)**

People`s Project є однією з найдавніших та найбільш структурованих платформ в Україні, що спеціалізується на зборі коштів для конкретних військових та соціальних проєктів [5]. Вона працює за принципом прямого фінансування потреб.

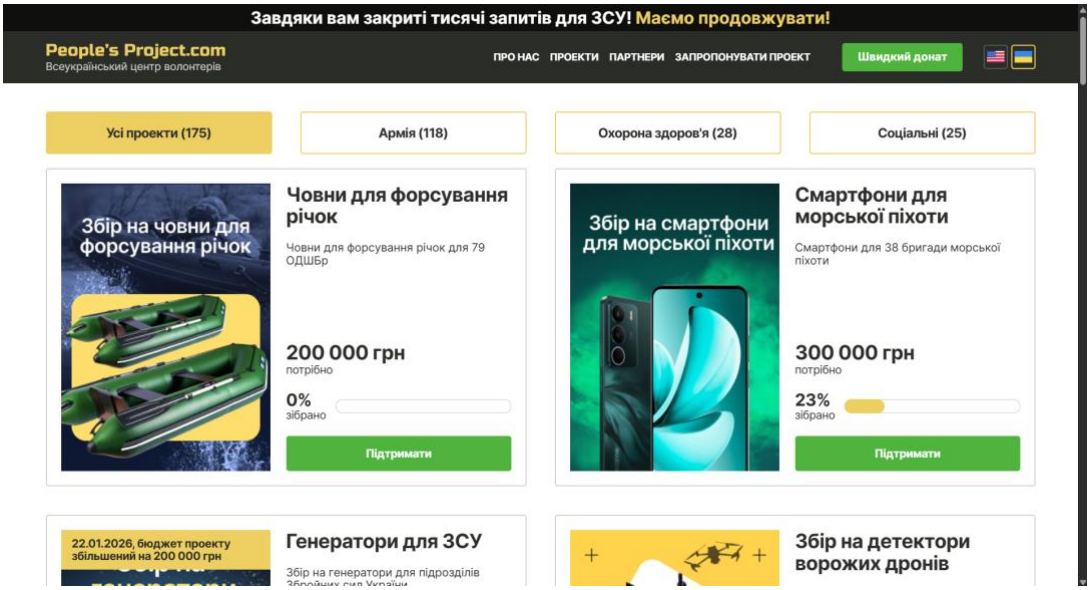



Рис. 3 Інтерфейс вебсайту People`s Project

**Переваги:**

- Висока довіра – багаторічна репутація та верифікація всіх проєктів;

- Прозорість – наявність детальної інформації про надходження та витрати коштів з відповідними документами по кожному збору у відкритому доступі;

**Недоліки:**

- Вузька спеціалізація – система орієнтована лише на збори коштів, що виключає можливість передачі існуючих у волонтерів ресурсів;
- Відсутність прямої взаємодії – Платформа не передбачає можливості прямого зв’язку між волонтерами та військовими;
- Відсутність вільної публікації – немає можливості вільно публікувати збори, можна лише запропонувати проєкт через форму;
- Орієнтованість на великі проєкти – система займається здебільшого великими зборами коштів для масштабних проєктів, через що дрібні потреби окремих підрозділів часто пропускаються.

**Фонд «Повернись живим»**

Професійна організація, яка фокусується на стратегічній допомозі армії забезпечуючи підрозділи обладнанням, зброєю та проводячи навчання військових [6].

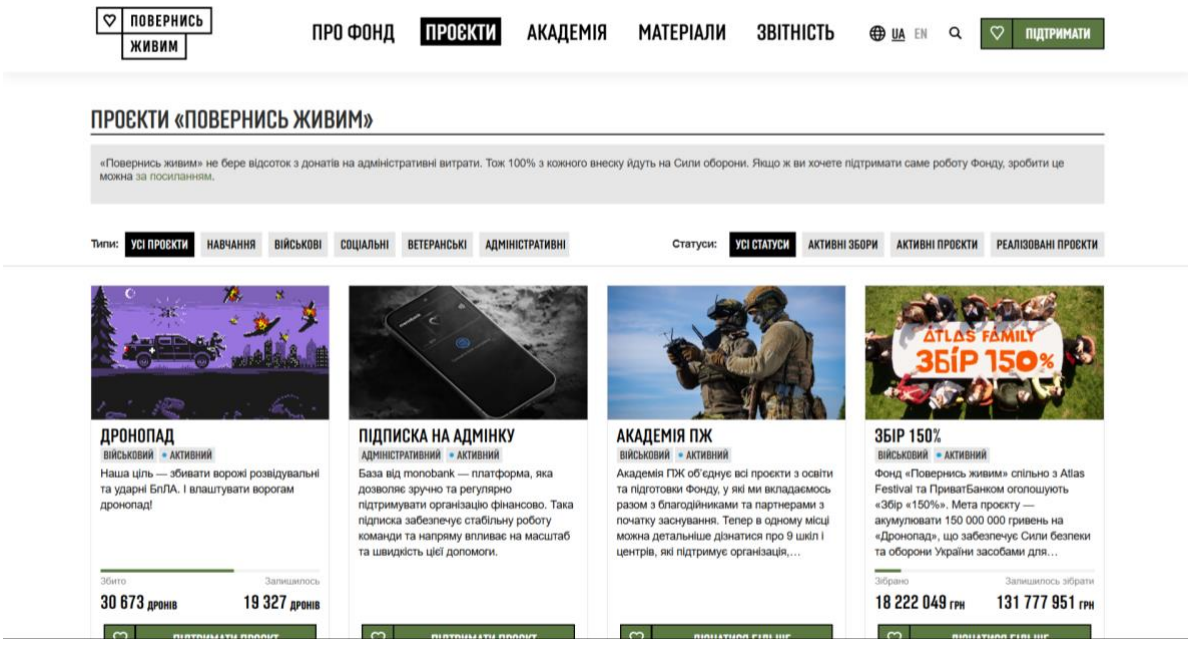



Рис. 4 Інтерфейс вебсайту фонду «Повернись живим»

**Переваги:**

- Публічна звітність – повна прозорість діяльності фонду з детальними фінансовими звітами, які знаходяться у відкритому доступі;
- Високий рівень безпеки – ретельна перевірка кожної потреби та забезпечення передачі ресурсів безпосередньо військовій частині.

**Недоліки:**

- Бюрократизація процесів – отримання допомоги можливе лише через офіційний запит від військової частини, що значно сповільнює надання допомоги;
- Лише масштабні закупівлі – фонд рідко розглядає дрібні або специфічні запити, які складають значну частину волонтерської роботи;
- Закрита структура – не передбачено інструментів для розповсюдження потреб сторонніми волонтерами чи самими військовими;

**Соціальні мережі**

На даний момент соціальні мережі є основними каналами для розповсюдження зборів та запитів на допомогу завдяки величезній кількості аудиторії.

**Переваги:**

- Охоплення – величезна аудиторія та можливість швидкого поширення інформації;
- Зручність – зручний формат створення дописів з фото та відео матеріалами;

**Недоліки:**

- Ризики шахрайства – відсутність перевірки та верифікації дописів дозволяє розповсюджувати фейкові збори;
- Велика кількість сторонньої інформації – в соціальних мережах поширюється безліч іншої інформації, через яку дуже важко просувати та здійснювати пошук потрібних зборів та запитів;

- Довільне звітування – звіти публікуються у довільній формі, що ускладнює перевірку.

**Месенджери**

Месенджери, як і соціальні мережі, є одними з основних каналів для поширення відповідної інформації серед населення та швидкої координації між учасниками волонтерського процесу.

**Переваги:**

- Швидкість – миттєвий обмін повідомленнями дозволяє ефективно розповсюджувати інформацію;
- Пряма комунікація – можливість напряму співпрацювати з іншими людьми через чати, групи або особисті повідомлення;
- Мобільність – зручність використання в будь-яких умовах, що є дуже важливими для військових.

**Недоліки:**

- Складність структурування – у великих групах та чатах важко структурувати та шукати потрібну інформацію, через постійне надходження нових повідомлень;
- Ризики безпеки та шахрайства – через відсутність перевірки та верифікації, можуть виникнути проблеми з фейками, а також з безпекою конфіденційної інформації;
- Обмежений функціонал – зазвичай можливо відправляти лише звичайні повідомлення.
- Відсутність централізації – інформація розкидана між великою кількістю різних чатів та груп, що ускладнює пошук та аналіз потреб.

Аналіз існуючих аналогів показав, що вони не забезпечують весь потрібний для вирішення поточних проблем функціонал. Спеціалізовані системи надають гарну структурованість інформації та прозорість волонтерської роботи, проте вони є занадто закритими, орієнтованими лише на масштабні фінансові збори та не надають можливості прямої комунікації. Соціальні мережі та месенджери, в свою чергу, забезпечують швидку та зручну взаємодію, а також

легке розповсюдження потрібної інформації, але мають недоліки у впорядкованості даних, безпеці, обліку та звітуванні. Аналіз надав можливість виявити переваги та недоліки існуючих рішень, що допомогло сформувати чіткі потреби потенційних користувачів та визначити вимоги до майбутнього продукту.

## 1.4 Постановка завдання

Метою розробки є створення клієнтської частини спеціалізованої інформаційної системи для обліку взаємодії між волонтерами та військовими. Система повинна мати інтуїтивно зрозумілий та адаптивний інтерфейс, що забезпечує швидкий і структурований доступ до актуальної інформації, надає зрозумілий інструментарій для легкого формування та розповсюдження зборів і запитів на допомогу, гарантує безпеку всіх процесів шляхом перевірок та верифікації для мінімізації ризиків шахрайства, а також забезпечує прозорість співпраці через облік даних та звітність у відкритому доступі

**Вхідні дані, які повинні оброблятись у системі:**

- Дані користувачів – роль (волонтер або військовий), ім’я або назва організації чи підрозділу, електронна пошта, РНОКПП, пароль;
- Дані дописів – тип допису (збір коштів або запит на допомогу), основна текстова інформація, що включає заголовок та детальний опис потреб, теги для фільтрації та зображення;
- Звітна інформація – детальний опис наданої або отриманої допомоги, фотопідтвердження та копії документів про виконання роботи;
- Відповіді на запит – повідомлення від волонтера до військового про можливість надати допомогу;
- Фільтри та параметри пошуку – ключові слова та обрані теги для більш глибокого пошуку інформації.

**Вихідні дані, які повинні генеруватись у системі:**

- Статус запиту – візуальне відображення поточного стану відповідного запиту;

- Звіти – структуровані документи про отримання або надання допомоги, з даними про запит та інформацією наданою автором звіту;
- Системні сповіщення – візуальні повідомлення про успішне виконання дій або помилки під час роботи.

**Функціональні вимоги:**

- Автентифікація – створення форм авторизації та реєстрації з вибором ролі та автоматичною валідацією полів на стороні клієнта;
- Сторінки з інформацією – окремі сторінки для структурованого відображення актуальних зборів коштів, запитів на допомогу, а також списків зареєстрованих волонтерів та військових;
- Пошук та фільтрація – забезпечення швидкого пошуку потрібної інформації по відповідній сторінці вебзастосунку, з можливістю фільтрації за допомогою тегів;
- Управління дописами – зручні форми для створення та редагування дописів з підтримкою додавання зображень та вибору тегів;
- Обробка запитів – форма для відгуку на запит, сторінка з відображенням всіх прийнятих запитів з можливістю зміни статусу та генерування звіту;
- Особистий профіль – сторінка для керування персональними даними та дописами;
- Сторінка звітів – інтерфейс для перегляду всіх згенерованих звітів по відповідним запитам з можливістю перегляду та експорту;

**Нефункціональні вимоги:**

- Безпека – Використання JWT-токенів для захисту сесій та запобігання крадіжці доступу через скрипти. Захист від XSS-атак через санітизацію введеного користувачем тексту, для блокування вставки шкідливого коду;
- Продуктивність – швидка робота інтерфейсу з використанням Lazy Loading для завантаження відповідних елементів лише тоді, коли

вони потрапляють у зону видимості користувача, мінімізація затримок при переході між сторінками та обробці запитів, а також можливість роботи застосунку з великою кількістю одночасних користувачів;

- Доступність – Адаптивний дизайн для зручної роботи на будь-яких пристроях. Інтерфейс має бути максимально простим та інтуїтивним, щоб користувачі з будь-яким рівнем технічних навичок могли миттєво зорієнтуватись в системі без додаткового навчання;
- Кросбраузерність – коректна робота застосунку у всіх сучасних версіях браузерів;
- Стабільність – Збереження цілісності даних шляхом використання механізмів кешування, щоб введений текст не зникав при розриві з’єднання чи інших збоях.

Отже, розроблювана клієнтська частина системи зосереджена на забезпеченні максимально зручної, швидкої та безпечної взаємодії між волонтерами та військовими в будь-яких умовах. Це реалізовано шляхом створення адаптивного та інтуїтивно зрозумілого інтерфейсу, що не потребує тривалого навчання, структуруванням інформації для забезпечення ефективного пошуку потрібних даних, а також впровадженням надійних механізмів безпеки для захисту користувачів та їх даних.

# 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1 Вебзастосунок та його складові

Вебзастосунок – це програмна система, що надає користувачам можливість взаємодіяти з даними через інтерфейс у браузері без потреби встановлювати додаткове програмне забезпечення на пристрій [7]. Він орієнтований на забезпечення високого рівня інтерактивності, виконання конкретних функцій та обробку динамічної інформації в режимі реального часу. Структура будь-якого сучасного вебзастосунку складається з двох тісно пов’язаних складових, а саме фронтенду та бекенду, кожна з яких виконує окрему роль у роботі системи.

Фронтенд – це частина застосунку, яка відповідає за візуалізацію контенту та взаємодію користувача з системою [8]. Його основною метою є створення інтуїтивно зрозумілого середовища, яке забезпечує швидке реагування на дії користувача та коректне відображення на екранах різних розмірів. Фронтенд забезпечує реалізацію всіх візуальних аспектів вебсистеми, наприклад: структури та стилізації графічного інтерфейсу, навігаційних елементів, форм введення даних, логіки відображення контенту та багато іншого. Для цього зазвичай використовуються мови HTML, CSS, JavaScript, а також можливе використання сучасних бібліотек та фреймворків, наприклад, React або Vue, які значно спрощують створення адаптивних та інтерактивних інтерфейсів. Якісний фронтенд безпосередньо впливає на загальний досвід користувача під час роботи із системою.

Бекенд – це частина вебзастосунку, що залишається прихованою від кінцевого користувача та відповідає за реалізацію всієї бізнес-логіки системи [8]. Він забезпечує взаємодію з базою даних для обробки та збереження інформації, а також передачі її на фронтенд-частину для подальшого використання. Окрім цього, бекенд здійснює автентифікацію користувачів і гарантує цілісність та безпеку даних. Для розробки бекенду використовуються різні мови програмування, такі як JavaScript, PHP, Python та спеціалізовані бібліотеки і

фреймворки, наприклад, Node.js, Express, Django чи Laravel, за допомогою яких реалізуються всі алгоритми роботи системи та забезпечується її стабільне функціонування навіть при значних навантаженнях.

Таким чином, фронтенд і бекенд є невід'ємними частинами сучасного вебзастосунку, які забезпечують його повноцінне функціонування. Їх взаємодія дозволяє реалізувати весь необхідний функціонал системи та забезпечити зручний користувацький інтерфейс для роботи з нею.

## 2.2 Взаємодія фронтенд та бекенд частин системи

**2.2.1 Логічна організація взаємодії.** В процесі проєктування інформаційної системи для обліку взаємодії волонтерів та військових було прийнято рішення реалізовувати вебзастосунок відповідно до ідеології клієнт-серверної архітектури, згідно з якою система поділяється на дві основні частини: клієнтську та серверну [9]. У межах розроблюваного програмного продукту фронтенд виступає як клієнтська частина, що забезпечує взаємодію користувача із застосунком та відображення інформації, а бекенд, в свою чергу, реалізує серверну частину, яка відповідає за обробку запитів, виконання бізнес-логіки та взаємодію з базою даних.

Такий підхід є одним із найпоширеніших для сучасних вебзастосунків і дає можливість чітко розділити відповідальність між окремими частинами системи та забезпечити злагоджену взаємодію між ними. Розподіл вебзастосунку на клієнтську та серверну частини дозволяє більш структуровано та ефективно організувати процес розробки, адже ці дві частини можуть створюватись та оновлюватись незалежно одна від одної. Також це полегшує тестування та підтримку системи, підвищує її гнучкість та спрощує майбутнє масштабування функціоналу. Крім того, це гарантує кращу безпеку та надійність вебзастосунку, оскільки серверна частина, яка контролює доступ до даних та виконує важливі функції, залишається прихованою від користувача.

Взаємодія між клієнтською та серверною частинами здійснюється шляхом обміну запитами та відповідями через мережеві протоколи. Клієнтська частина

формує запит на основі дій та даних користувача, після чого передає його до серверної частини для подальшої обробки. У відповідь сервер виконує необхідні операції, а саме: звертається до бази даних для роботи з інформацією, виконує дії над нею згідно запиту та формує результат, який повертається до клієнтської частини у відповідному форматі. Отримані від сервера дані використовуються для оновлення інтерфейсу користувача. Такий підхід забезпечує динамічну взаємодію користувача із застосунком без необхідності перезавантаження сторінок та дозволяє реалізувати асинхронну обробку даних для безперервної роботи із застосунком під час виконання запитів. Обмін інформацією здійснюється у зрозумілому для обох частин системи форматі, що забезпечує коректну інтерпретацію даних та стабільність взаємодії.

Логічна організація системи базується на поділі складових клієнтської та серверної частин на окремі модулі, кожен з яких містить певні функціональні компоненти, виконує відповідні завдання і взаємодіє з іншими модулями через визначені інтерфейси. Це дозволяє впорядкувати внутрішню структуру вебзастосунку та його програмного коду, а також спростити навігацію всередині проєкту.

**2.2.2 Механізми взаємодії та обміну даними.** У процесі роботи інформаційної системи дуже важливими є механізми взаємодії між клієнтською та серверною частинами, оскільки вони забезпечують передачу, обробку та оновлення даних у вебзастосунку. Правильна організація обміну інформацією дозволяє забезпечити ефективну роботу системи, швидке реагування на дії користувача, коректне відображення даних та стабільність застосунку. Для реалізації такої взаємодії використовується програмний інтерфейс REST API, який забезпечує можливість комунікації між фронтендом та бекендом.

REST (Representational State Transfer) – це архітектурний стиль організації програмних інтерфейсів, який базується на використанні HTTP-протоколу для обміну даними між різними частинами системи [10]. Основними перевагами REST є простота реалізації, універсальність та можливість незалежної роботи клієнтської і серверної частин. У межах розроблюваної системи REST API

використовується для надсилання запитів від клієнтської частини до серверної та отримання відповідей із необхідною інформацією. Кожен ресурс системи має відповідну адресу (endpoint), через яку здійснюється взаємодія з даними.

Обмін даними між клієнтом та сервером реалізується за допомогою HTTP-запитів. Для виконання різних операцій над даними використовуються відповідні HTTP-методи:

- GET – для отримання інформації із сервера;
- POST – для створення нових записів;
- PUT – для оновлення існуючих даних;
- DELETE – для видалення даних.

Після отримання запиту серверна частина виконує необхідну обробку, взаємодіє з базою даних та формує відповідь, яка повертається на клієнтську частину разом із кодом стану HTTP. Використання таких кодів дозволяє автоматично визначати результат виконання запиту та реагувати на нього відповідним чином.

Для передачі інформації між клієнтською та серверною частинами використовується формат JSON (JavaScript Object Notation) [11]. Даний формат є одним із найпоширеніших стандартів обміну структурованими даними у вебзастосунках завдяки своїй зрозумілості та простоті обробки. JSON дозволяє передавати різні дані у текстовому вигляді, що забезпечує правильну інтерпретацію інформації на обох частинах системи. Вибір цього формату зумовлений тим, що він базується на мові JavaScript, яка використовується для реалізації застосунку.

Для забезпечення безперервної взаємодії користувача із системою обмін даними виконується асинхронно. Надсилання запитів та отримання відповідей відбувається без повного перезавантаження вебсторінки. Такий підхід дозволяє покращити швидкодію застосунку та забезпечити кращий користувацький досвід. Асинхронна взаємодія реалізується за допомогою технології Fetch API, яка дозволяє виконувати HTTP-запити у фоновому режимі та динамічно

оновлювати окремі елементи інтерфейсу після отримання відповіді від сервера [12].

Після отримання відповіді від сервера клієнтська частина виконує оновлення стану застосунку відповідно до отриманих даних. Це може включати зміну вмісту сторінок, оновлення списків дописів, відображення повідомлень про успішне виконання операцій або повідомлень про помилки. Управління станом дозволяє підтримувати актуальність інформації у користувацькому інтерфейсі та забезпечує коректну взаємодію між всіма компонентами системи.

Отже, використання REST API, HTTP-запитів, формату JSON та асинхронного обміну даними забезпечує ефективну взаємодію між клієнтською та серверною частинами системи. Це дозволяє організувати стабільний, гнучкий та масштабований механізм обробки інформації, який є невід'ємною частиною для повного функціонування вебзастосунку.

## 2.3 Архітектура системи

Для відображення архітектури та внутрішньої структури інформаційної системи доцільно використовувати діаграму пакетів.

Діаграма пакетів – це один із видів структурних діаграм мови UML, який використовується для відображення архітектури системи [13]. Основною метою такої діаграми є представлення структури програмного проєкту у вигляді окремих пакетів, що об'єднують пов'язані між собою компоненти, модулі або підсистеми відповідно до їх функціонального призначення. Вона широко використовується під час проєктування великих інформаційних систем, оскільки дозволяє розділити складний програмний продукт на окремі логічні частини. Це надає можливість зрозуміло представити ієрархію майбутнього проєкту та визначити зони відповідальності кожного модуля, що спрощує організацію командної розробки.

Основними елементами діаграми є:

- Пакет – основний структурний елемент, який виступає контейнером для групування пов'язаних компонентів системи. Усередині пакетів

можуть розміщуватись класи, модулі, підсистеми, інші пакети або різні структурні елементи проєкту;

- Залежність – використовується для відображення взаємозв’язків між пакетами та показує, як саме окремі частини системи взаємодіють одна з одною. В діаграмі пакетів можуть використовуватись наступні типи залежності:
  - «import» – дозволяє імпортувати публічні елементи одного пакета в інший, завдяки чому вони можуть використовуватись без прямого звернення до базового пакета;
  - «use» – показує, що один пакет використовує функціонал або сервіси іншого для виконання певних операцій;
  - «access» – надає можливість залежному пакету отримувати доступ до всіх публічних елементів базового, без їх імпортування;
  - «trace» – відображає історичний розвиток одного елемента в інший, більш розвинений варіант;
  - «merge» – використовується для доповнення або розширення елементів базового та залежного пакетів шляхом їх об’єднання.

На рис. 5 зображено діаграму пакетів, побудовану для системи обліку взаємодії волонтерів та військових, яка відображає розподіл основних пакетів та типи залежностей між ними.

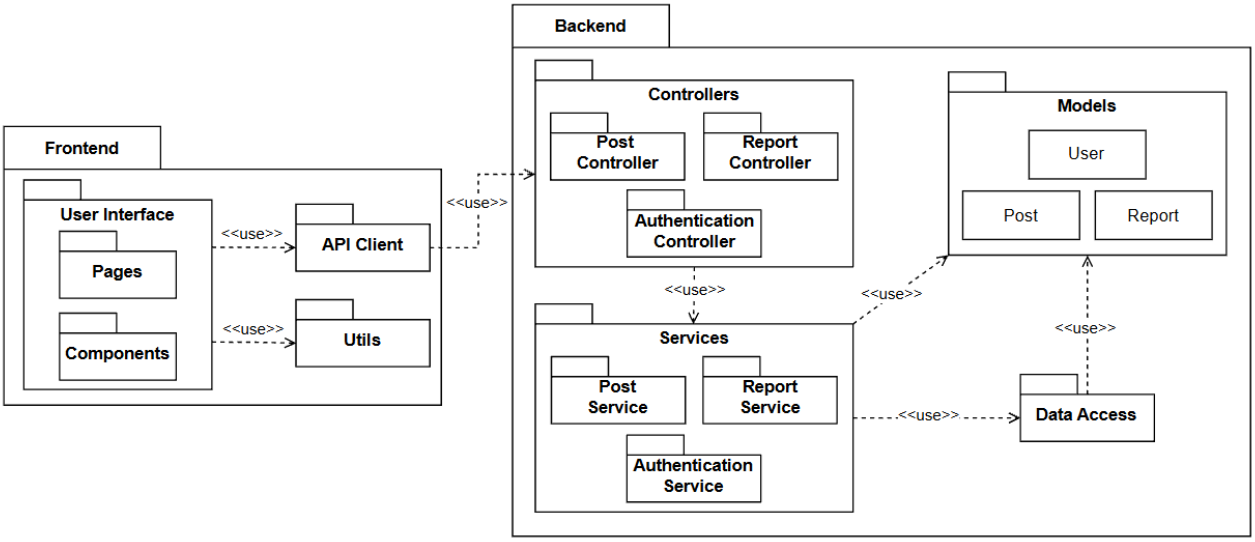



Рис. 5 Діаграма пакетів

**Опис пакетів діаграми:**

**1. Пакет Frontend (Клієнтська частина):**

- User interface – пакет фронтенду, що відповідає за візуальну частину системи. Містить наступні підпакети:
  - Pages – сторінки застосунку, кожна з яких відповідає за конкретний функціонал (наприклад, сторінка зборів коштів, запитів на допомогу, особистий профіль);
  - Components – містить окремі повторно використовувані елементи з яких складається інтерфейс (кнопки, модальні вікна, картки дописів);
- API Client – пакет, який відповідає за мережеву взаємодію з бекендом, забезпечує формування та передачу HTTP-запитів до серверної частини;
- Utils – містить допоміжні функції, які використовуються іншими пакетами фронтенду, наприклад для форматування даних, обробки подій або керування окремими елементами інтерфейсу;

**2. Пакет Backend (Серверна частина):**

- Controllers – забезпечує обробку вхідних запитів на сервері. Кожен контролер приймає відповідний запит, викликає потрібний сервіс та

повертає відповідь користувачу. На діаграмі відображені наступні контролери:

- Post Controller – оброблює запити на створення, редагування або видалення дописів;
- Report Controller – забезпечує обробку запитів на створення та керування звітами про результати виконаної роботи;
- Authentication Controller – керує запитами на автентифікацію користувачів та їх доступом до операцій у системі;

- Services – пакет, який містить основну бізнес-логіку системи для реалізації вхідних запитів та обробки даних. До нього входять:
  - Post Service – реалізує логіку керування дописами;
  - Report Service – відповідає за генерацію, перегляд та керування звітами;
  - Authentication Service – забезпечує автентифікацію користувачів, хешування паролів, генерацію та верифікацію JWT-токенів.
- Data Access – пакет для виконання операцій з базою даних. Виконує команди відповідних сервісів для отримання, обробки та виконання дій над потрібними даними;
- Models – містить опис структур даних, якими оперує вся система. На діаграмі описані наступні моделі даних:
  - User – модель користувача системи з його правами, роллю та реєстраційними даними;
  - Post – описує структуру допису з відповідним типом, детальним описом потреб, зображеннями та тегами;
  - Report – структура звіту про отримання або передачу допомоги з інформацією про запит та вхідними даними користувача, а також фотопідтвердженням.

**Опис зв’язків між пакетами:**

- Зв’язок Frontend та Backend – клієнтська частина звертається через пакет API Client до пакету Controllers на серверній частині за допомогою протоколу HTTP;
- Зв’язки внутрішніх пакетів Frontend:
  - User Interface та API Client – користувацький інтерфейс використовує методи API Client для надсилання мережевих запитів;
  - User Interface та Utils – сторінки та компоненти використовують допоміжні функції з пакета Utils для виконання відповідних операцій над елементами інтерфейсу;
- Зв’язки внутрішніх пакетів Backend:
  - Controllers та Services – контролери передають отримані від клієнтської частини запити до відповідних сервісів, які їх реалізують;
  - Services та Data Access – сервіси звертаються до Data Access для виконання CRUD-операцій (створення, читання, оновлення та видалення) у базі даних;
  - Services та Models – сервіси використовують моделі даних для керування об’єктами, зміни їх станів та передачі інформації між різними модулями системи;
  - Data Access та Models – пакет доступу до даних використовує структури моделей для перетворення записів із бази даних у зрозумілі системі об’єкти.

# 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Абстракції предметної області

Абстракції використовуються для виділення найважливіших властивостей та обов’язків відповідних об’єктів, ігноруючи другорядні деталі реалізації, що є важливим кроком під час проєктування програмного забезпечення та полегшує безпосередню розробку.

Далі наведено основні абстракції предметної області (рис. 6), що беруть участь у взаємодії волонтерів і військових, описано їх ключові характеристики та призначення у цьому процесі.


Рис. 6 Абстракції предметної області

**Опис абстракцій**

- Військовий – Ініціює взаємодію. Його властивостями є ідентифікуюча інформація, контактні дані та список створених запитів. Основними обов’язками є формування запиту, встановлення зв’язку з волонтером для уточнення деталей, отримання допомоги та звітування;
- Волонтер – Відповідає за опрацювання та виконання запиту. Також має ідентифікуючу та контактну інформацію, а також список активних запитів. Приймає запити, передає їх на перевірку, встановлює зв’язок з військовим для уточнення деталей, збирає та передає ресурси або кошти і звітує;
- Контролюючий орган – забезпечує безпеку та прозорість. Має назву і контактні дані. Перевіряє запити та звіти, фіксує та уточнює порушення або невідповідності та генерує загальний звіт по виконаному запиту;

- Запит – головний об’єкт взаємодії, що зв’язує волонтера і військового. Містить інформацію про відправника, отримувача, опис потреб, статус та термін виконання;
- Допомога – описує ресурси або кошти які потрібно забезпечити в результаті виконання запиту. Характеризуються типом допомоги, описом та кількістю;
- Звіт – фінальний об’єкт взаємодії, що описує надання або отримання допомоги відповідною стороною. Описує тип звіту, дані про виконану роботу та фіксує час створення. Забезпечує експорт документу у зручному форматі.

Визначення абстракцій дозволило чітко окреслити зони відповідальності ключових учасників у процесі взаємодії волонтерів та військових, визначити структуру даних у системі та сформувати уявлення про майбутній дизайн інтерфейсу для кожної ролі користувачів.

## 3.2 Моделювання структури та взаємодії об’єктів

Моделювання структури та взаємодії об’єктів є дуже важливим етапом проєктування, на якому абстракції предметної області перетворюються на конкретні програмні компоненти. Цей етап зосереджений на описі внутрішньої архітектури системи, визначенні властивостей та функцій об’єктів, встановленні зв’язків між ними та формуванні принципів їх взаємодії для реалізації функціональних вимог. Такий підхід дозволяє створити основу для майбутнього застосунку та ще до початку розробки виявити можливі проблеми.

Для ефективної візуалізації структури та функціонування системи використовується набір діаграм мови UML, які дозволяють описати статичні та динамічні аспекти розроблюваного продукту. Діаграма класів забезпечує статичне моделювання, яке фокусується на постійних характеристиках та логіці системи. Діаграми кооперацій, в свою чергу, відносяться до динамічного моделювання, яке більш детально описує взаємодію між об’єктами системи в конкретних сценаріях.

**3.2.1 Діаграма класів.** Діаграма класів – це діаграма мови UML, яка використовується для відображення статичної структури програмної системи. Вона описує основні класи системи, їх атрибути, методи та відношення між ними [14]. Ця діаграма надає можливість сформувати загальне уявлення про внутрішню організацію програмного застосунку, розподіл функціональності між окремими об’єктами та способи їх взаємодії.

Основними компонентами діаграми класів є:

- Клас – основний елемент діаграми, який описує певну сутність або об’єкт програмної системи із визначеними характеристиками та функціями. Кожен клас містить ім’я, атрибути, які описують його властивості та дані, а також методи, які визначають функції та операції, що можуть виконуватись об’єктами класу. Атрибути та методи можуть мати різну видимість, яка визначає рівень доступу до них з інших класів:
  - public (+) –доступний будь-якому іншому класу;
  - private (-) – доступний лише всередині класу;
  - protected (#) – доступний класу та його нащадкам;
  - package (~) – доступний у межах одного пакету класів.
- Зв’язок – використовується для відображення взаємодії, залежності або ієрархії між класами системи. Зв’язки показують, яким чином окремі класи пов’язані між собою, як обмінюються даними або використовують функціональні можливості один одного. У діаграмі класів існують наступні типи зв’язків:
  - Асоціація – відображає наявність структурного зв’язку між об’єктами класів. Характеризується кратністю, що визначає кількість об’єктів кожного класу, які беруть участь у взаємодії;
  - Залежність – вказує на те, що зміна в одному класі може вплинути на інший;
  - Узагальнення – описує наслідування між класами, при якому дочірній клас успадковує атрибути та методи батьківського;

- Агрегація – показує, що один клас містить об’єкти іншого, але вони існують незалежно від нього. Кратність зв’язку вказує на кількість об’єктів що може входити до складу головного класу;
- Композиція – зв’язок, при якому об’єкти одного класу, не можуть існувати окремо від іншого і створюються та знищуються разом з ним. Також містить кратність зв’язку.

На рис. 7 зображено діаграму класів з асоціаціями для системи обліку взаємодії волонтерів і військових.

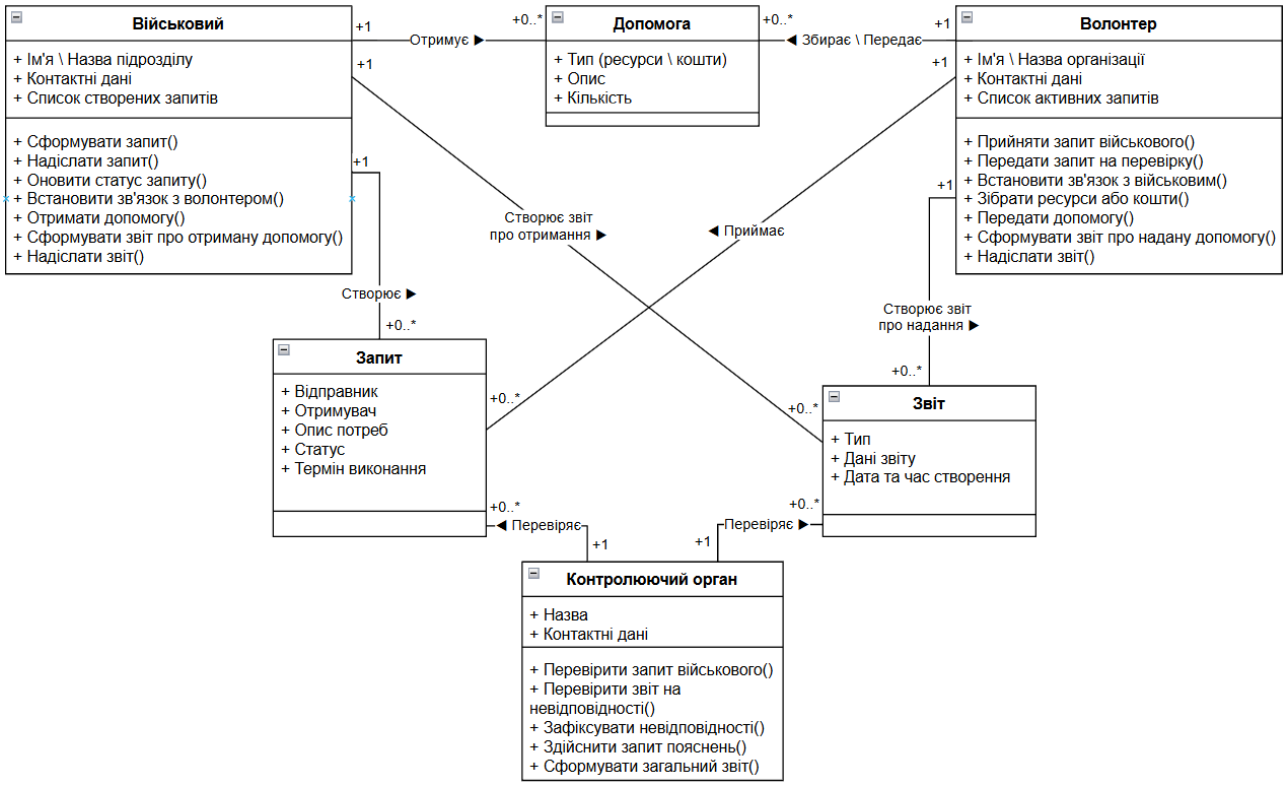



Рис. 7 Діаграма класів з асоціаціями

**Опис класів та зв’язків діаграми:**

- Клас «Військовий» – описує користувача системи, який потребує допомоги та ініціює початок взаємодії шляхом створення запиту. Даний клас містить основну інформацію про військового або підрозділ, а саме ім’я або назву підрозділу, контактні дані та список створених запитів. Методи класу забезпечують формування та надсилання запитів, оновлення їх статусу, встановлення зв’язку з

волонтером, отримання допомоги та створення звітів. Клас має наступні зв’язки:

- Зв’язок із класом «Запит» – військовий може створювати безліч запитів;
- Зв’язок із класом «Допомога» – відображає отримання необхідних ресурсів або коштів військовим;
- Зв’язок із класом «Звіт» – військовий генерує звіти про отримання допомоги.

- Клас «Волонтер» – описує користувача, який займається виконанням запитів військових та організацією передачі допомоги. Містить інформацію про ім’я волонтера або назву організації, контактні дані та список активних запитів. Методи класу дозволяють приймати запити військових, передавати їх на перевірку, встановлювати зв’язок з військовим для уточнення деталей, збирати ресурси або кошти, передавати допомогу та формувати звіт про виконану роботу. Клас має наступні зв’язки:
  - Зв’язок із класом «Запит» – показує прийняття та обробку запитів волонтером;
  - Зв’язок із класом «Допомога» – волонтер забезпечує збір та передачу ресурсів або коштів;
  - Зв’язок із класом «Звіт» – волонтер створює звіти з результатами надання допомоги.
- Клас «Контролюючий орган» – здійснює перевірку достовірності запитів та відповідності звітів. Містить назву та контактні дані. Методи класу забезпечують перевірку запитів військових, аналіз звітів на наявність невідповідностей, їх фіксацію з запитом пояснень і формує загальні звіти. Має наступні зв’язки:
  - Зв’язок із класом «Запит» – контролюючий орган перевіряє запити військових на достовірність;

- Зв’язок із класом «Звіт» – перевірка звітів на наявність невідповідностей, а також створення загального звіту по конкретному запиту.

- Клас «Запит» – описує потребу військового у певній допомозі. Містить інформацію про відправника та отримувача, детальний опис потреб, поточний статус та термін виконання.
- Клас «Допомога» – описує матеріальні або фінансові ресурси, які збираються волонтером для забезпечення потреб військового. Містить тип допомоги (ресурси або кошти), її опис та кількість.
- Клас «Звіт» – використовується для обліку результатів отримання або надання допомоги у вигляді структурованих звітів. Містить тип звіту, дані та дату і час його створення.

Побудована діаграма класів дозволила структурувати дані та функціональні обов’язки кожного учасника системи, визначивши перелік атрибутів, методів та зв’язків для кожного класу. Вона є основою для подальшої програмної реалізації застосунку, забезпечуючи відповідність встановленим функціональним вимогам.

**3.2.2 Діаграми кооперацій.** Діаграми кооперацій використовуються для візуалізації того, як саме окремі об’єкти відповідних класів системи співпрацюють між собою для реалізації конкретного сценарію. На відміну від діаграми класів, яка описує статичну структуру системи, кооперації зосереджені на динамічній взаємодії об’єктів. Під час побудови таких діаграм переважно використовуються типи зв’язків, що дозволяють детальніше показати логіку взаємодії об’єктів для реалізації окремих функціональних можливостей системи, а саме відношення залежності, узагальнення, агрегації та композиції.

На рис. 8 наведено діаграму кооперацій для процесу створення та виконання запиту.

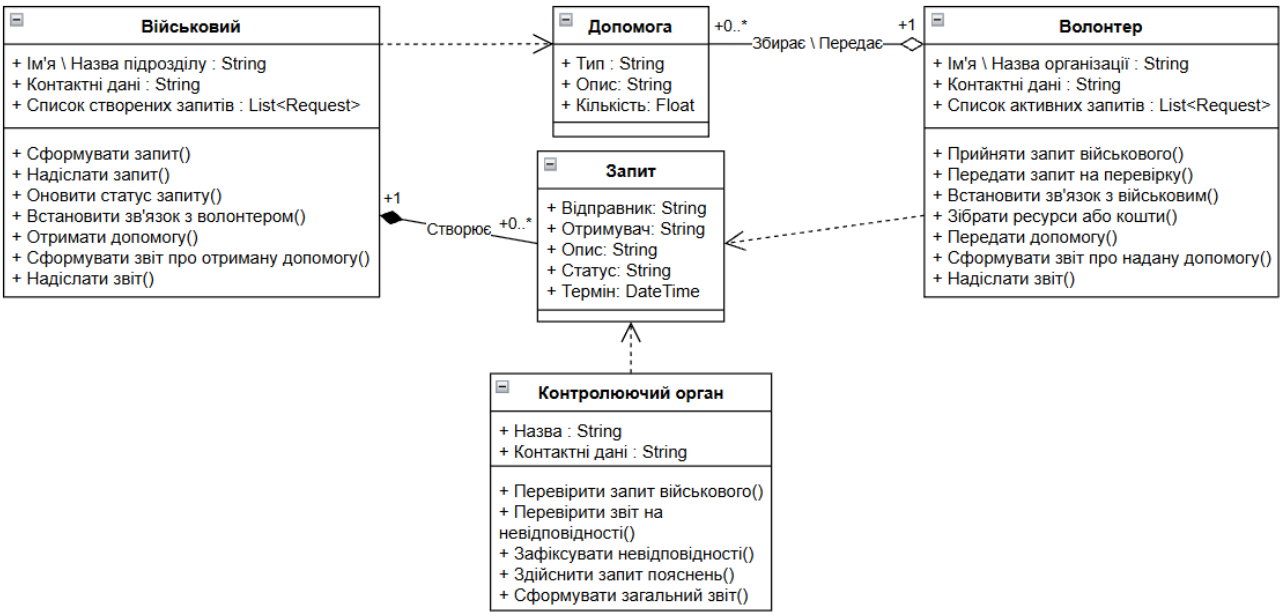



Рис. 8 Діаграма кооперацій: Створення та виконання запиту

Дана кооперація візуалізує взаємодію класів під час процесу виконання запиту на допомогу, від моменту його створення до отримання допомоги військовим.

**Опис зв’язків у межах кооперації:**

- Військовий та Запит – зв’язок композиції один до багатьох, який вказує на те, що військовий може створювати безліч запитів і вони не можуть існувати без нього, їх життєвий цикл повністю керується військовим;
- Волонтер та Запит – зв’язок залежності показує, що волонтер використовує інформацію прийнятого запиту для виконання своєї частини роботи;
- Контролюючий орган та Запит – зв’язок залежності демонструє, що контролюючий орган використовує запит для здійснення його перевірки та верифікації;
- Волонтер та Допомога – зв’язок агрегації один до багатьох, відображає що волонтер збирає потрібну допомогу для виконання запиту, але після здійснення її передачі військовому, вона вже існує незалежно від нього;

- Військовий та Допомога – зв’язок залежності який показує що військовий напряму залежить від отриманих ресурсів.

На рис. 9 зображено діаграму кооперацій для процесу звітування про виконану роботу.

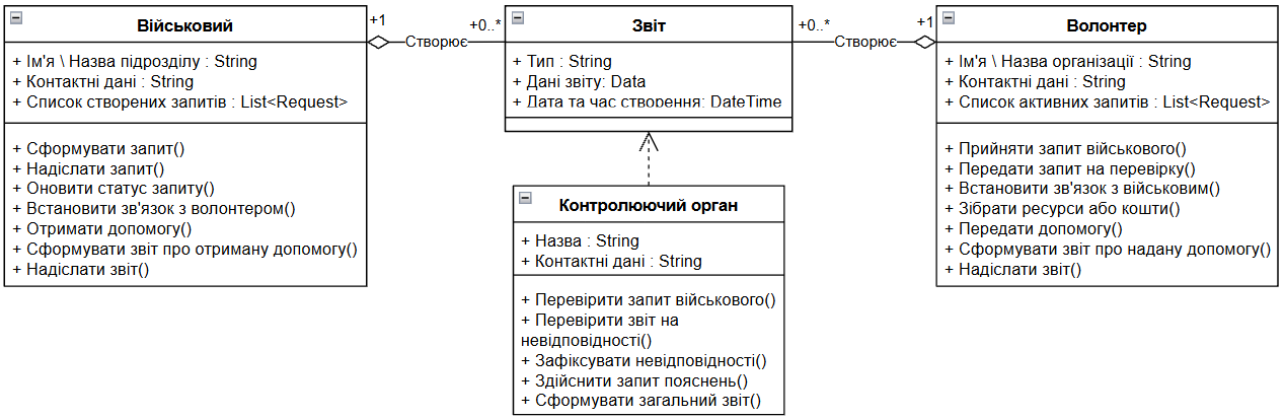



Рис. 9 Діаграма кооперацій: Звітування

Наведена кооперація візуалізує останній етап взаємодії волонтерів і військових, що відповідає за документування результатів, їх перевірку контролюючим органом та забезпечення прозорості діяльності обох сторін через систему звітності.

**Опис зв’язків у межах кооперації:**

- Військовий та звіт – зв’язок агрегації один до багатьох, що вказує на можливість військового формувати звіти про отриману допомогу, які після створення стають незалежним документом, який зберігається у системі для обліку;
- Волонтер та звіт – зв’язок агрегації один до багатьох, відображає процес створення волонтером звітів за результатами збору та передачі допомоги, які так само існують та зберігаються незалежно після їх формування;
- Контролюючий орган та Звіт – зв’язок залежності, який демонструє, що контролюючий орган використовує звіти для здійснення їх перевірки на наявність невідповідностей між наданою допомогою та заявленою у запиті військового, а також формування підсумкового звіту.

## 3.3 Компонентна структура системи

Після моделювання структури та взаємодії об’єктів наступним етапом є визначення фізичної структури програмного забезпечення та організація його окремих модулів. Для візуалізації фізичного поділу системи використовується діаграма компонентів мови UML.

Діаграма компонентів – це структурна діаграма UML, яка використовується для відображення фізичної організації та модульної структури програмної системи [15]. Вона показує, з яких реально існуючих програмних модулів, бібліотек, файлів, сервісів та інших, складається застосунок та як вони взаємодіють між собою.

**Основними елементами діаграми компонентів є:**

- Компонент – це основний елемент діаграми, який представляє окремий модуль або частину системи, що реалізує певний функціонал. Компонентами можуть бути файли програмного коду, бібліотеки, сервіси, підсистеми та інші фізичні елементи системи. Для уточнення призначення компонентів використовуються наступні стереотипи:
  - «executable» – визначає компонент, який може бути виконаний у застосунку;
  - «library» – програмна бібліотека, що містить набір допоміжних функцій;
  - «table» – таблиця бази даних;
  - «file» – окремий файл системи, який містить будь-які дані;
  - «source» – файл, який містить вхідний програмний код;
  - «document» – будь-який документ.
- Інтерфейс – визначає набір операцій або сервісів, які компонент надає іншим компонентам системи. Інтерфейси забезпечують взаємодію між окремими частинами системи. Один компонент може одночасно як надавати, так і використовувати декілька інтерфейсів;

- Залежність – використовується для відображення взаємозв'язків між компонентами та показує, що один компонент використовує функціональні можливості іншого.

На рис. 10 зображено діаграму компонентів системи обліку взаємодії волонтерів і військових, яка відображає основні програмні компоненти вебзастосунку та зв'язки між ними.

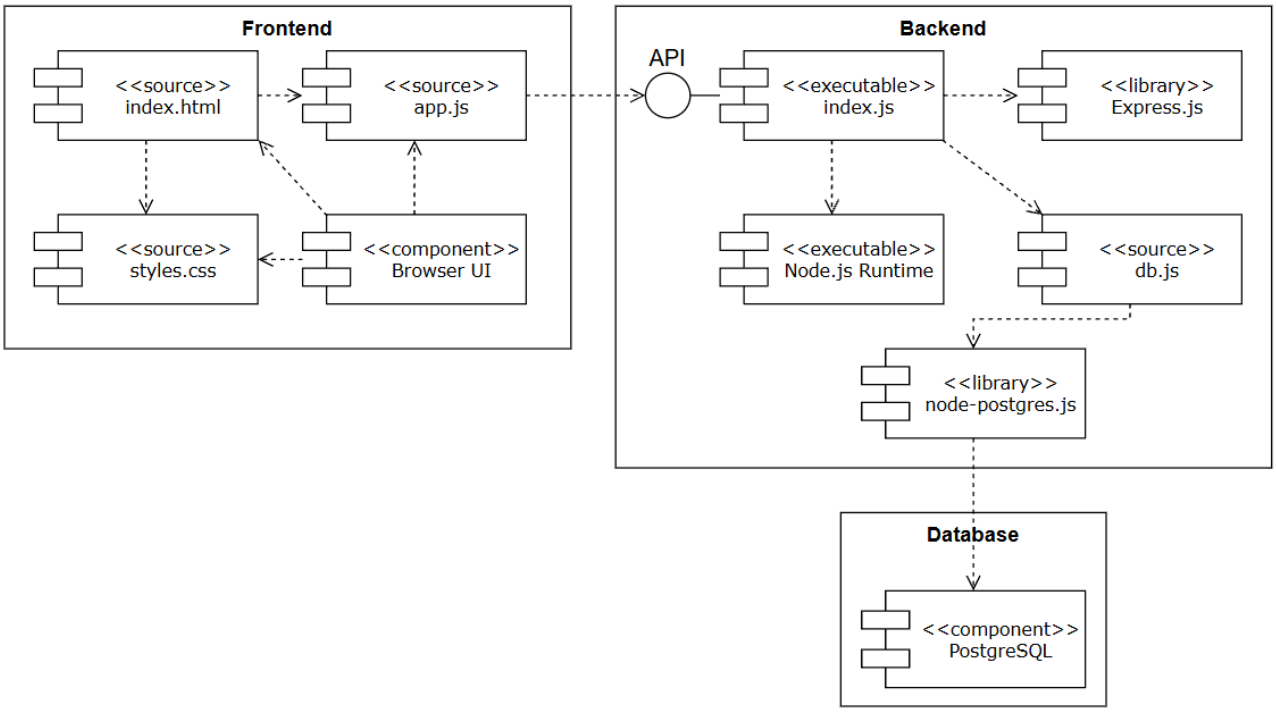



Рис. 10 Діаграма компонентів

Наведена діаграма компонентів складається з трьох основних вузлів: клієнтської частини (Frontend), серверної частини (Backend), та рівень зберігання даних (Database).

**Опис вузлів та їх компонентів:**

- Вузол Frontend – представляє клієнтську частину вебзастосунку, що виконується у браузері користувача. Його компонентами є:
  - index.html («source») – файл розмітки, що містить вхідний код структури інтерфейсу;
  - styles.css («source») – файл каскадних таблиць стилів, що відповідає за візуальне стилізування елементів;

- app.js («source») – основний файл логіки фронтенду, який забезпечує обробку дій та надсилання запитів користувача до API бекенду;
- Browser UI («component») – компонент користувацького інтерфейсу у браузері, що забезпечує відображення всіх елементів застосунку.

- Вузол Backend – частина застосунку, що забезпечує реалізацію бізнес-логіки системи, обробку вхідних запитів та взаємодію з базою даних. Складається з наступних компонентів:
  - API – інтерфейс, через який фронтенд частина застосунку взаємодіє з бекендом;
  - index.js («executable») – головний виконуваний файл бекенду, що забезпечує основний функціонал, а також отримання та обробку запитів;
  - Express.js («library») – фреймворк, який використовується для організації маршрутизації та спрощення обробки HTTP-запитів;
  - Node.js Runtime («executable») – середовище виконання, необхідне для інтерпретації та роботи всього JavaScript-коду на стороні сервера;
  - db.js («source») – файл, що містить вхідний код для забезпечення конфігурації та підключення до бази даних;
  - node-postgres.js («library») – бібліотека, що забезпечує відправку SQL-запитів та обмін даними між бекендом та СУБД PostgreSQL.
- Вузол Database – рівень збереження даних, який містить компонент PostgreSQL («component»), що відповідає за надійне зберігання всіх даних системи.

**Опис зв’язків між компонентами:**

- Внутрішні зв’язки Frontend:

- index.html та styles.css - файл розмітки залежить від таблиці стилів для відображення коректного візуального оформлення елементів;
- index.html та app.js – структура інтерфейсу залежить від файлу логіки фронтенду для забезпечення динамічної взаємодії користувача та обробки подій;
- Browser UI та всі інші компоненти вузла – відображає, що користувацький інтерфейс залежить від всіх файлів з вхідним кодом для коректного відображення та функціонування інтерфейсу у браузері.

● Внутрішні зв’язки Backend:

- index.js та Express.js – головний виконуваний файл бекенд частини залежить від фреймворку Express для маршрутизації та обробки запитів;
- index.js та Node.js Runtime – файл логіки залежить від середовища виконання для інтерпретації та запуску коду;
- index.js та db.js – виконуваний файл логіки звертається до файлу взаємодії з базою даних для передачі та отримання даних;
- db.js та node-postgres.js – файл з вхідним кодом для взаємодії з базою даних, використовує для цього можливості бібліотеки node-postgres.

● Зв’язок між вузлами Frontend та Backend: головний файл логіки фронтенд частини app.js передає запити користувача через інтерфейс API, який надається компонентом бекенд частини index.js.

● Зв’язок між вузлами Backend та Database: файл db.js забезпечує виконання SQL-запитів для роботи з даними в PostgreSQL за допомогою бібліотеки node-postgres.

Побудова діаграми компонентів дозволила визначити фізичну структуру системи та розподілити функціональність між окремими модулями. Визначені

компоненти, інтерфейси та залежності між ними створюють чіткий план для безпосередньої розробки програмного продукту.

## 3.4 Вибір інструментарію для створення прикладного програмного забезпечення

Процес розробки фронтенд частини системи обліку взаємодії волонтерів і військових ґрунтується на використанні сучасних інструментів та стандартів вебтехнологій, що забезпечують високу продуктивність, легкість коду та швидкість роботи системи. Вибір технологічного стеку зумовлений необхідністю створити стабільний продукт з адаптивним та інтуїтивно зрозумілим дизайном, що буде здатний ефективно працювати навіть у важких умовах, в яких перебувають військові.

**Основними інструментами та технологіями розробки є:**

- Visual Studio Code – основне середовище розробки, було обрано завдяки його зручності, гнучкості та наявності великої кількості розширень для роботи з JavaScript, HTML та CSS [16]. Використання вбудованих інструментів дебаггінгу та інтеграції з системами контролю версій дозволило ефективніше організувати розробку, пришвидшити написання та тестування коду;
- HTML5 – мова розмітки, що забезпечує створення семантичної структури сторінок та визначає розташування всіх елементів [17]. Використання семантичних тегів забезпечує кращу доступність вебзастосунку, правильну інтерпретацію контенту пошуковими системами та допоміжними технологіями [18].
- CSS3 – мова стилів, яка використовувалась для налаштування візуального оформлення елементів інтерфейсу, створення анімацій, переходів між станами елементів та загального покращення користувацького досвіду [19]. Для забезпечення гнучкості та адаптивності дизайну, активно використовувались можливості Flexbox та CSS Grid. Вони дозволили створити інтерфейс, який

автоматично підлаштовується під різні розміри екранів, що є критично важливим у польових умовах;

- JavaScript ES6+ – мова програмування, що забезпечує реалізацію динамічної поведінки інтерфейсу та взаємодії користувача [20]. За допомогою JS реалізовано обробку подій користувача, роботу з формами, валідацію введених даних, динамічне оновлення контенту сторінок, взаємодію з бекенд частиною застосунку та багато іншого. Під час розробки було прийнято рішення не використовувати важкі фреймворки, що дозволило зменшити навантаження на систему, підвищити швидкість роботи та забезпечити максимальну продуктивність. Також такий підхід надає можливість краще контролювати роботу фронтенду;
- Fetch API – вбудований у браузер інтерфейс JavaScript, що застосовується для здійснення асинхронних HTTP-запитів до бекенду [12]. Він дозволяє отримувати та надсилати дані у форматі JSON без перезавантаження сторінки, забезпечуючи плавність роботи інтерфейсу;
- JWT – технологія, що використовується на стороні фронтенду для безпечного обміну інформацією між користувачами. Токени зберігаються у браузері та додаються до заголовків запитів для автентифікації при спробі доступу до захищених ресурсів [21];
- Git – система контролю версій, яка дозволяє відстежувати зміни у коді, здійснювати паралельну розробку різних частин за допомогою гілок, фіксувати етапи розробки та забезпечувати цілісність проєкту;
- GitHub – платформа для віддаленого зберігання Git-репозиторіїв, що використовувалась для збереження програмного коду та спрощення організації командної роботи [22];
- Google Chrome DevTools – вбудований у браузер інструментарій, який використовується для відлагодження коду, тестування адаптивності та моніторингу мережевих запитів [23].

Обраний набір інструментів та технологій дозволив реалізувати всі функціональні та нефункціональні вимоги до системи, забезпечити ефективну розробку, підтримку та масштабування фронтенд частини вебзастосунку.

## 3.5 Алгоритмізація та програмування програмних модулів

Алгоритмізація використовується для визначення логіки виконання конкретних операцій в системі. Для цього використовується графічне представлення алгоритмів у вигляді блок-схем, що надає можливість відобразити послідовність дій користувача, процеси обробки даних, логіку розгалужень, циклічні операції та точки прийняття рішень. Використання таких схем дозволяє виявити можливі помилки, а також значно спростити написання коду [24]. У межах даного розділу буде представлено опис ключових алгоритмів системи, а саме авторизації та створення допису.

**Блок схеми мають наступні елементи:**

- Початок та кінець алгоритму (прямокутник із заокругленими кутами) – позначають старт та завершення виконання процесу;
- Процес (прямокутник) – відображає виконання певної дії;
- Введення або виведення даних (паралелограм) – використовується для позначення отримання або передачі інформації;
- Рішення (ромб) – позначає розгалуження у алгоритмі, де здійснюється вибір одного з двох варіантів результату;
- Потік керування (стрілка) – відображає послідовність кроків;
- Підпроцес (прямокутник із подвійними бічними лініями) – виклик допоміжного алгоритму.

На рис. 11 представлено блок-схему алгоритму авторизації в системі.

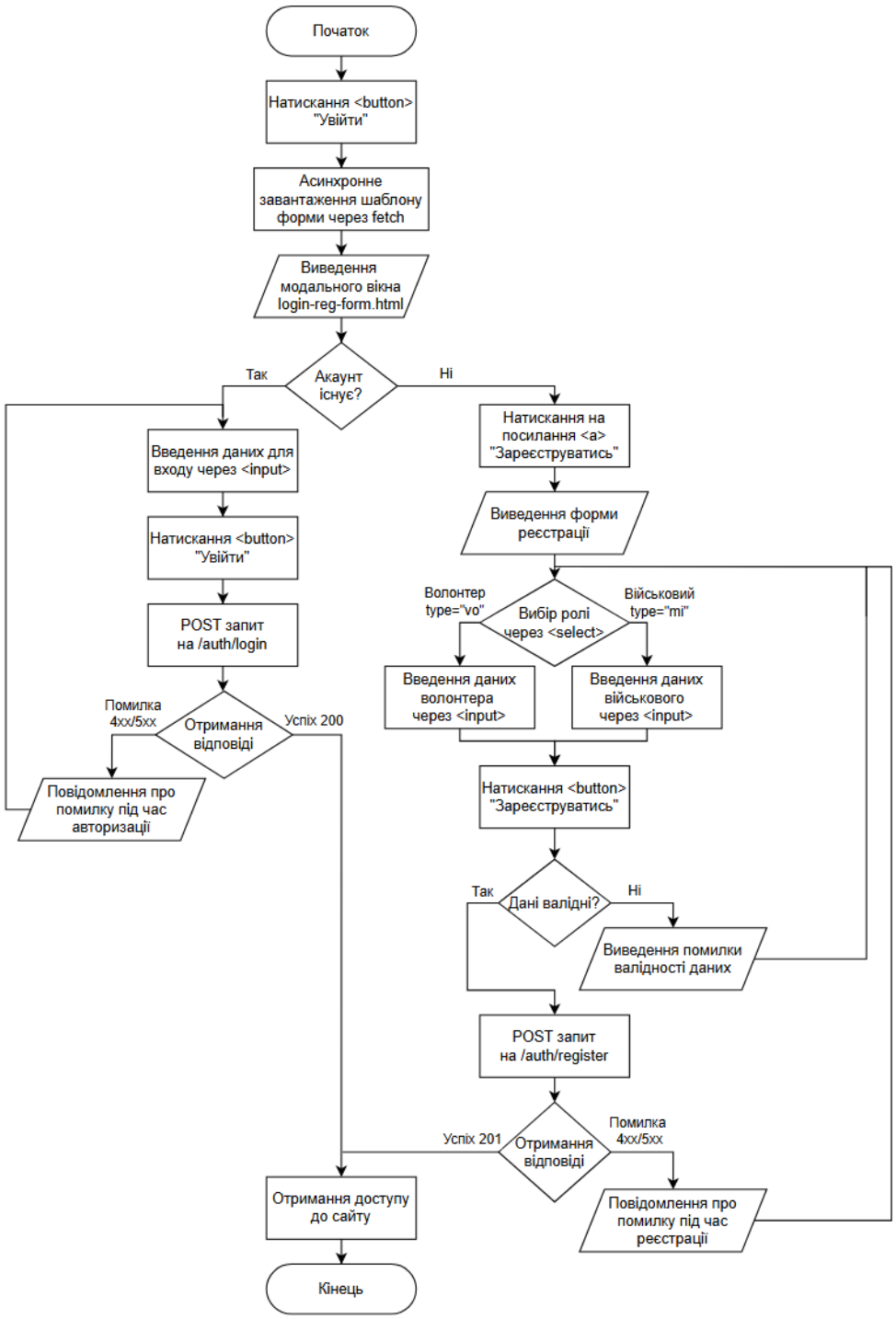



Рис. 11 Блок-схема алгоритму авторизації в системі

Алгоритм авторизації ініціюється натисканням кнопки «Увійти», що запускає асинхронне завантаження шаблону форми login-reg-form.html через fetch та його подальше виведення у модальному вікні. Система пропонує два сценарії взаємодії залежно від наявності акаунту в користувача:

1. Авторизація – користувач вводить облікові дані (email та пароль) через відповідні поля (<input>). Після натискання кнопки «Увійти»

виконується POST-запит на API-ендпоінт /auth/login. Далі відбувається отримання відповіді від бекенду:

- Помилка 4xx/5xx – система виводить відповідне повідомлення про помилку авторизації та повертає користувача до введення даних;
- Успіх 200 – користувач успішно авторизований у системі та отримує доступ до відповідного функціоналу сайту.

2. Реєстрація нового користувача – при відсутності акаунту, користувач натискає на гіперпосилання «Зареєструватись» і відбувається перемикання форми. Першим кроком реєстрації є вибір ролі волонтера або військового з випадаючого списку (<select>). Далі користувач вводить реєстраційні дані, а саме електронну пошту, ім’я або назву, РНОКПП, пароль та натискає кнопку «Зареєструватись», після чого відбувається валідація даних:
   - Дані невалідні – виводиться відповідна помилка (наприклад, паролі не збігаються або некоректний формат РНОКПП) і користувач повертається до редагування даних;
   - Дані валідні – здійснюється надсилання даних до бекенду через Fetch API методом POST на ендпоінт /auth/register.

   Останнім кроком є отримання та аналіз відповіді від бекенду:
   - Помилка 4xx/5xx – виводиться повідомлення про відповідну помилку реєстрації;
   - Успіх 201 – система повідомляє про успішну реєстрацію та користувач отримує доступ до застосунку.

В Додатку Б наведено HTML код розмітки форм авторизації та реєстрації користувача у системі.

**Основними елементами форми входу є:**

- Текстові поля різних типів:
  - <input type=”email”> – для введення електронної пошти з вбудованою валідацією формату;

- <input type=”password”> – для введення паролю з приховуванням символів.

- Кнопка toggle-password – дозволяє перемикати видимість пароля для перевірки його коректності під час входу;
- Посилання <a> – гіперпосилання на форму реєстрації у раз відсутності акаунту.

**Основними елементами форми реєстрації є:**

- Випадаючий список <select> – дозволяє обрати одну із ролей під час реєстрації:
  - type=”vo” – волонтер;
  - type=”mi” – військовий.
- Спеціалізовані текстові поля:
  - <input type=”text” minlength=10 maxlength=10 pattern=[0-9]*> – поле для введення РНОКПП з використанням обмежень та паттерну, що сприяє коректному введенню даних;
  - <input type=”password” oninput=”validatePassword()”> – для підтвердження введеного паролю з використанням функції validatePassword(), яка при введенні порівнює значення двох полів для запобігання помилкам при реєстрації.
- Посилання <a> – гіперпосилання для переходу до форми авторизації, якщо акаунт вже існує.

В Додатку В надано код для керування процесом авторизації, реєстрації та забезпечення взаємодії між фронтендом та API бекенду. Наведена функція забезпечує виконання наступних операцій:

- Динамічне перемикання між формами – обробники подій click на посиланнях regLink та loginLink змінюють форму шляхом додавання до потрібної CSS-класу active-modal та видалення у іншої;
- Процес авторизації – при натиску на кнопку «Увійти» викликається подія submit при якій введені дані збираються, очищаються від зайвих пробілів за допомогою функції trim(), серіалізуються в JSON-

об’єкт та надсилаються асинхронно через fetch на ендпоінт /auth/login/. При успішній відповіді (response.ok) користувач отримує доступ до системи, інакше виводиться відповідно помилка;

- Процес реєстрації – аналогічно до авторизації, дані форми реєстрації проходять обробку та серіалізацію і асинхронно надсилаються на ендпоінт /auth/register/. Після успішного створення акаунту користувач автоматично авторизується в системі;
- Обробка помилок – обидва процеси знаходяться в блоках try-catch, що дозволяє перехоплювати помилки мережі або недоступність сервера, відображаючи зрозумілі сповіщення через alert.

На рис. 12 представлено блок-схему алгоритму створення допису.

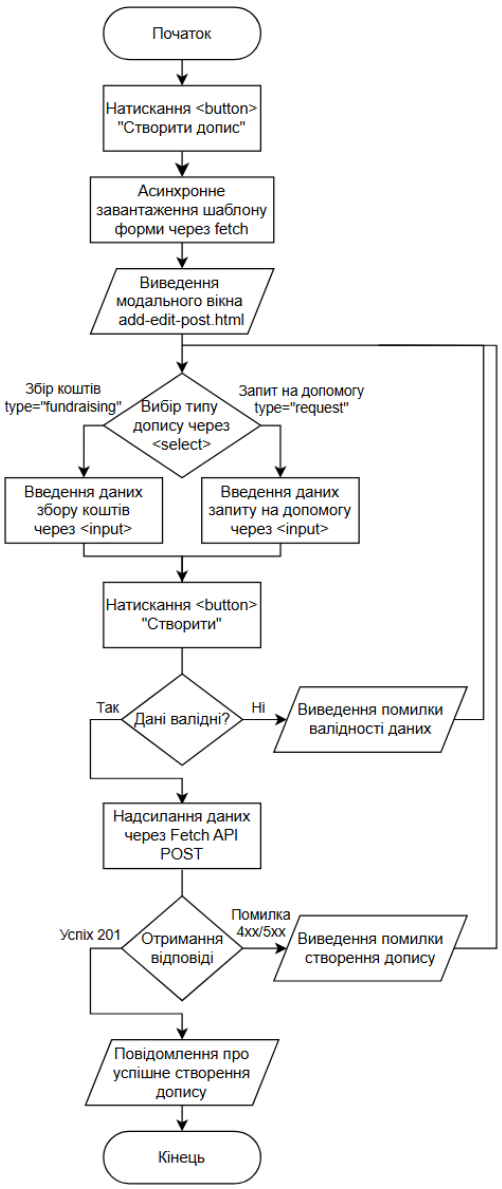


Рис. 12 Блок-схема алгоритму створення допису у системі

Алгоритм створення допису починається натисканням кнопки «Створити допис», далі асинхронно завантажується шаблон форми add-edit-post.html через fetch та виводиться у вигляді модального вікна. Наступним кроком є вибір типу публікації (збір коштів або запит на допомогу) через випадаючий список (<select>), після чого користувач вводить дані у відповідні поля (<input>, <textarea>). При натисканні кнопки «Створити» відбувається валідація форми:

- Виявлено некоректно заповнені або порожні поля – система виводить відповідне повідомлення та повертає користувача до редагування;
- Валідація успішна – здійснюється надсилання даних до бекенду за допомогою Fetch API методом POST.

Далі відбувається аналіз отриманої відповіді від бекенду:

- Помилка 4xx/5xx – система виводить відповідну помилку і повертає користувача до введення даних;
- Успіх 201 – система повідомляє про успішне створення допису.

На рис. 13 наведено HTML код розмітки форми для створення допису.

```
<form class="form active-modal" id="add-post-form">
    <div class="post-text-input select-wrapper">
        <label for="post-type" class="input-title">Тип</label>
        <select name="type" id="type">
            <option value="fundraising" selected>Збір коштів</option>
            <option value="request">Запит на допомогу</option>
        </select>
    </div>
    <div class="post-text-input">
        <label for="post-title" class="input-title">Заголовок</label>
        <input type="text" id="post-title" name="post-title" required>
    </div>
    <div class="post-text-input">
        <label for="post-tags" class="input-title">Теги</label>
        <div class="post-tags-entry-row">
            <input type="search" id="post-tags" name="post-tags" placeholder="Додати тег">
            <button type="button" class="add-tag-button" aria-label="Додати тег">
                <img src="/public/images/add-icon.png" alt="add tag">
            </button>
        </div>
        <div class="post-tags" id="user-post-tags">
        </div>
    </div>
    <div class="post-text-input post-description">
        <label for="post-text" class="input-title">Текст</label>
        <div class="textarea-wrapper">
            <textarea id="add-post-text" name="post-text" required></textarea>
            <div class="image-container" id="post-image-preview-container">
            </div>
            <hr class="textarea-divider">
            <label for="post-image-upload" class="image-upload-icon" title="Додати фото">
                <img src="/public/images/image-icon.png" alt="Додати фото">
            </label>
            <input type="file" class="image-upload" id="post-image-upload" accept="image/*" multiple hidden>
            <p class="image-upload-hint">Можна додати до 5 фото. Гіфки не підтримуються.</p>
            <p class="image-upload-error hidden" aria-live="polite"></p>
        </div>
    </div>
    <button type="submit" class="submit-button">Створити</button>
</form>
```

Рис. 13 HTML розмітка форми для створення допису

**Основні елементи форми:**

- Випадаючий список <select> – дозволяє обрати один з двох типів допису:
  - type=”fundraising” – збір коштів;
  - type=”request” – запит на допомогу.
- Текстові поля:
  - <input type=”text”> – для введення заголовку допису;
  - <textarea> – для введення детального опису.
- Система тегів – поле пошуку (<input type=”search”>) для додавання тегів, кожен з яких візуалізується у блоці ( );
- Завантажувач зображень (<input type=”file” accept=”image/*”> - для додавання зображень до допису;
- Кнопка відправки (<button type=”submit”>) – ініціює валідацію даних та їх передачу на API-ендпоінт бекенду.

На рис. 14 зображено обробник події click для кнопки addPostButton для відображення форми для створення допису.

```
addPostButton?.addEventListener('click', async () ⇒ {
  if (!document.getElementById('post-modal')) {
    const res = await fetch('/public/pages/components/modal-forms/add-edit-post-form/add-edit-post.html');
    const html = await res.text();
    modalContainer.innerHTML = html;
    bindSharedModalClose();
  }

  showAddEditPostModal('add');
  document.getElementById('add-post-form')?.classList.add('active-modal');
  document.getElementById('edit-post-form')?.classList.remove('active-modal');
});
```

Рис. 14 Код для обробки натискання кнопки «Створити допис»

Програмна логіка реалізована за допомогою асинхронної функції (async), яка перевіряє наявність модального вікна в DOM-дереві. Якщо елемент post-modal відсутній, скрипт виконує запит fetch до сервера для отримання файлу add-edit-post.html, зчитує його та динамічно вставляє в modalContainer. Після цього викликається функція відображення вікна та встановлюється активний стан для форми створення допису шляхом редагування CSS-класів.

На рис. 14 наведено реалізацію функції showAddEditPostModal, яка відповідає за процес відображення модального вікна.

```
function showAddEditPostModal(mode = 'add') {
  const overlay = document.getElementById('modal-overlay');
  const modal = document.getElementById('post-modal');
  const heading = modal?.querySelector('.modal-heading');

  if (!overlay || !modal || !heading) {
    return;
  }

  heading.textContent = mode === 'edit' ? 'Редагувати допис' : 'Створити допис';
  overlay.classList.remove('hidden');
  modal.classList.remove('hidden');
  document.body.classList.add('modal-open');

  const user = window.AuthState?.getUser?.();
  const roleCode = user?.role_code;
  const roleName = (user?.role_name || '').trim().toLowerCase();
  const isMilitary = roleCode === 'mi' || roleName === 'військовий';

  modal.querySelectorAll('select[name="type"]').forEach((select) => {
    const fundraisingOption = Array.from(select.options).find((option) => option.value === 'fundraising');
    const requestOption = Array.from(select.options).find((option) => option.value === 'request');
    if (!fundraisingOption || !requestOption) {
      return;
    }

    fundraisingOption.hidden = false;
    fundraisingOption.disabled = false;
    requestOption.hidden = !isMilitary;
    requestOption.disabled = !isMilitary;

    if (!isMilitary && select.value === 'request') {
      select.value = 'fundraising';
    }
  });
}
```

Рис. 14 Функція для відображення модального вікна форми

Функція приймає параметр mode, що визначає тип вікна: «Створення допису» або «Редагування допису». Основні дії функції включають перевірку існування необхідних DOM-елементів (overlay, modal, heading), потім видалення CSS-класу hidden для фонового шару та модального вікна, що робить їх видимими, а також додавання класу modal-open до тіла документу для блокування прокрутки основної сторінки під час створення допису. Далі функція отримує інформацію про поточного користувача ти визначає його роль для розмежування функціональних можливостей. Військовий може створювати запити будь-якого типу, а волонтер лише збори коштів.

# 4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

## 4.1 Тестування системи

Тестування – це один із найважливіших етапів розробки програмного забезпечення, адже воно дозволяє перевірити коректність роботи застосунку, відповідність функціональним та нефункціональним вимогам та виявити помилки до моменту впровадження в експлуатацію. Основною метою тестування фронтенд частини системи обліку взаємодії волонтерів і військових є перевірка правильності роботи користувацького інтерфейсу, його зручності та адаптивності під різні розміри екранів, коректного функціонування у різних браузерах та стабільності взаємодії між фронтендом та бекендом [25].

У процесі розробки використовувались різні види тестування, які спрямовані на перевірку окремих аспектів роботи системи:

- Функціональне тестування – використовується для перевірки відповідності розробленого функціоналу поставленим вимогам під час проєктування. Під час даного тестування перевірялась робота форм авторизації та реєстрації, створення дописів з можливістю додавати теги та зображення, створення звітів та керування особистим профілем;
- Модульне тестування – застосовувалось для перевірки окремих функцій та модулів фронтенду незалежно один від одного. Тестувалась робота JS-функцій, що відповідають за роботу з модальними вікнами, валідацію полів форм та обробку подій користувача;
- Інтеграційне тестування – спрямоване на перевірку взаємодії між фронтендом та бекендом системи. Здійснювалась перевірка правильності надсилання HTTP-запит через Fetch API, обробка

JSON-об’єктів, коректність роботи API-ендпоінтів та реакція інтерфейсу на різні статуси відповідей;

- Інтерфейсне тестування – використовувалось для перевірки відображення елементів користувацького інтерфейсу. Перевірялась коректність роботи всіх елементів, їх розташування та стилізація, а також навігація по застосунку;
- Метод «Чорної скриньки» – використовувався для тестування функціоналу системи без аналізу її програмного коду. Під час такого тестування перевірялась правильність реакції інтерфейсу на дії користувача на основі вхідних даних та отриманих результатів.
- Кросбраузерне тестування – застосовувалось для перевірки стабільної роботи вебзастосунку у різних браузерах, таких як Google Chrome, Microsoft Edge, Opera, Mozilla Firefox та Safari для забезпечення однакової роботи JS-коду та CSS-стилів;
- Тестування адаптивності – дозволило перевірити коректність відображення інтерфейсу на різних розмірах екранів. Перевірка здійснювалась на персональних комп’ютерах, ноутбуках, планшетах та мобільних пристроях;

Далі наведено приклад виконання тестування з використанням методу «Чорної скриньки», що дозволило оцінити коректність роботи застосунку з точки зору користувача. Основна увага приділяється перевірці реакції системи на відповідні дії та введені дані. Для кожного сценарію були визначені очікувані результати, які порівнювались з фактичною поведінкою системи.

**Опис етапів тестування:**

- Реєстрація користувача з роллю військовий – при відсутності акаунту користувач відкриває форму реєстрації, вибирає свою роль та вводить реєстраційній дані, а саме електронну пошту, ім’я або назву, РНОКПП, пароль та підтвердження. На рис. 15 зображена форма реєстрації з введеними даними. Якщо реєстрація успішна, користувач отримує доступ до системи;

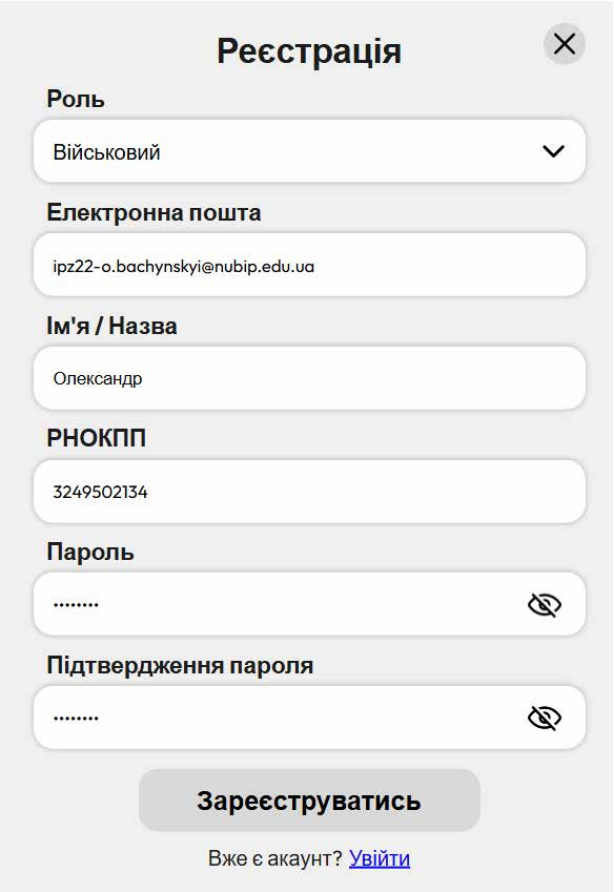



Рис. 15 Форма реєстрації

- Доступ до системи та перевірка створення акаунту – після успішної реєстрації користувач з’являється у відповідному списку військових або волонтерів, згідно його ролі, це продемонстровано на рис. 16;

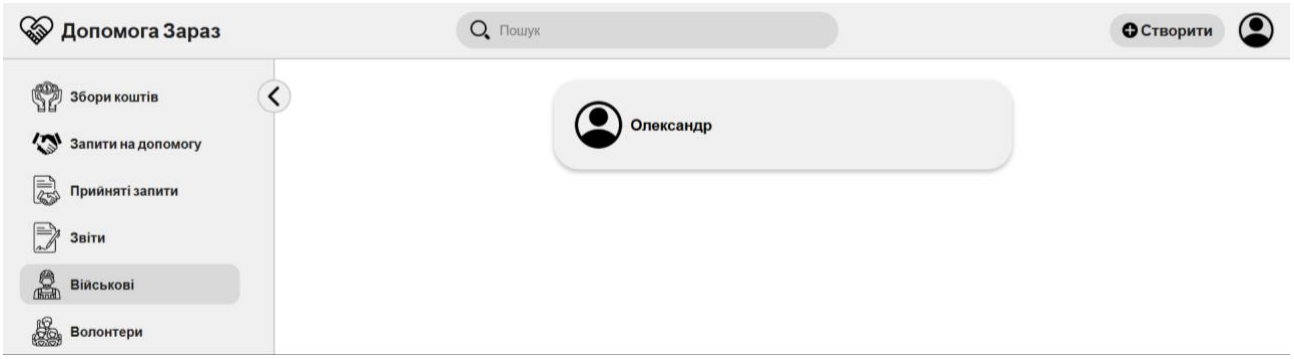



Рис. 16 Сторінка списків військових

- Створення нового запиту на допомогу та перевірка – для створення запиту користувач натискає на кнопку «Створити» у правому верхньому куті застосунку, після чого відкривається відповідна форма для створення допису (рис. 17), де потрібно обрати тип, ввести всі необхідні дані, а також за бажанням додати теги або зображення.

Після успішного створення запиту, його зможуть побачити усі користувачі застосунку на відповідній сторінці «Запити на допомогу» (рис. 18);

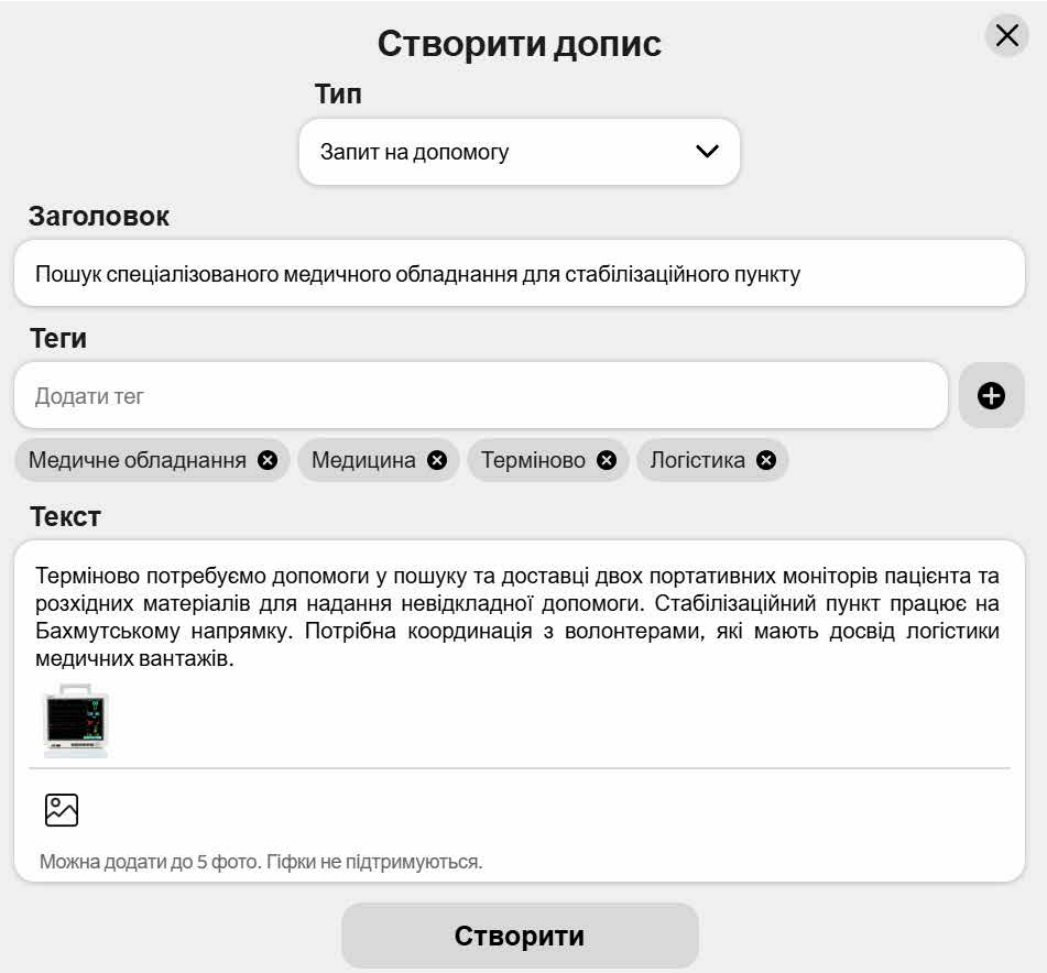



Рис. 17 Форма створення допису

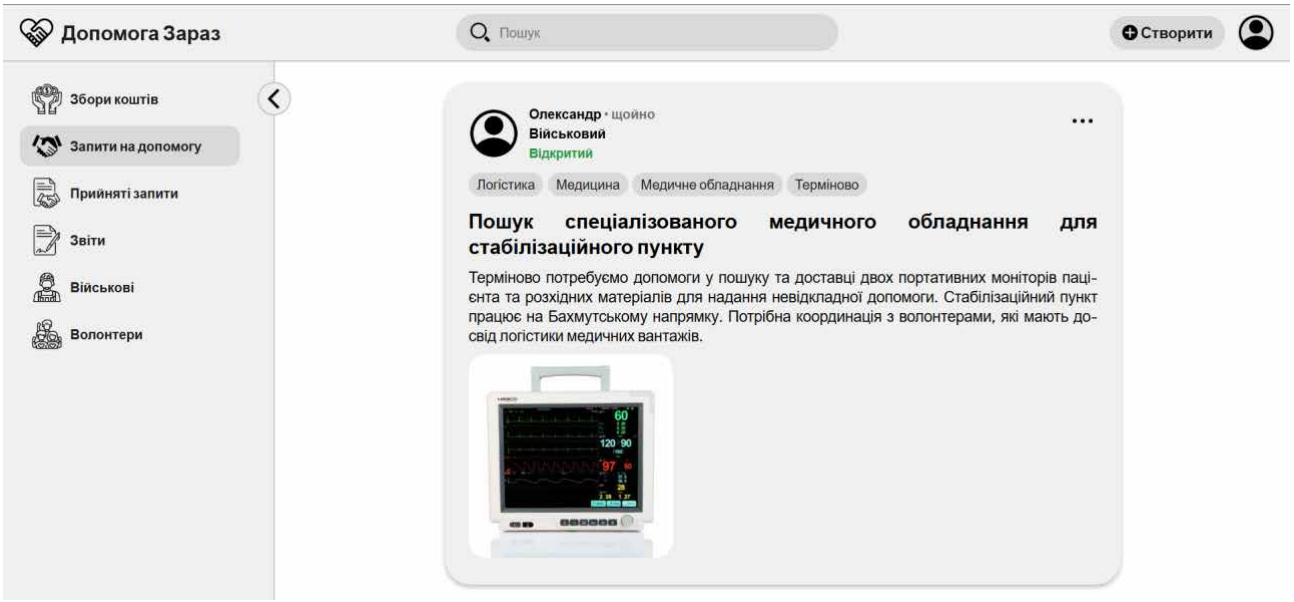



Рис. 18 Сторінка «Запити на допомогу» зі створеним дописом

- Вхід в акаунт волонтера – при наявності акаунту, користувачу потрібно авторизуватись в системі для доступу до функціоналу. Для цього потрібно натиснути на кнопку «Увійти», що відкриє форму входу (рис. 19), та заповнити в ній облікові дані.

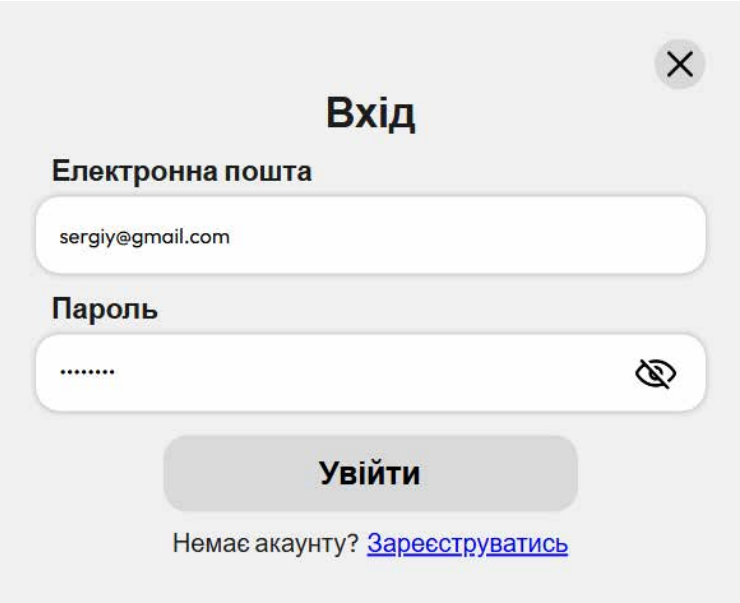



Рис. 19 Форма авторизації в системі

- Перегляд сторінок зборів коштів та запитів на допомогу – для пошуку та перегляду дописів відповідного типу використовуються окремі сторінки. На рис. 20 наведено приклад сторінки «Запити на допомогу» з точки зору волонтера, якому доступна можливість відгукуватись на запити;

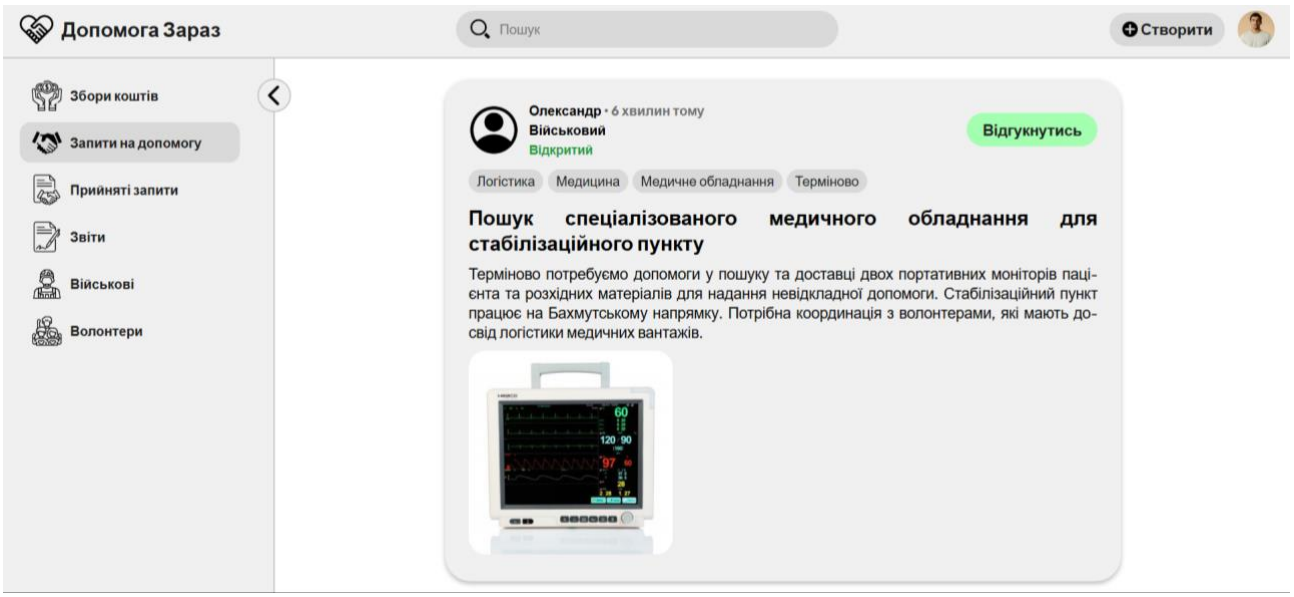



Рис. 20 Сторінка «Запити на допомогу» з точки зору волонтера

- Відгук на запит – на кожному запитів присутня кнопка «Відгукнутись», натиснувши яку відкривається форма для надсилання відгуку військовому (рис. 21), де потрібно ввести заголовок та текст відгуку, а також можна додати зображення;

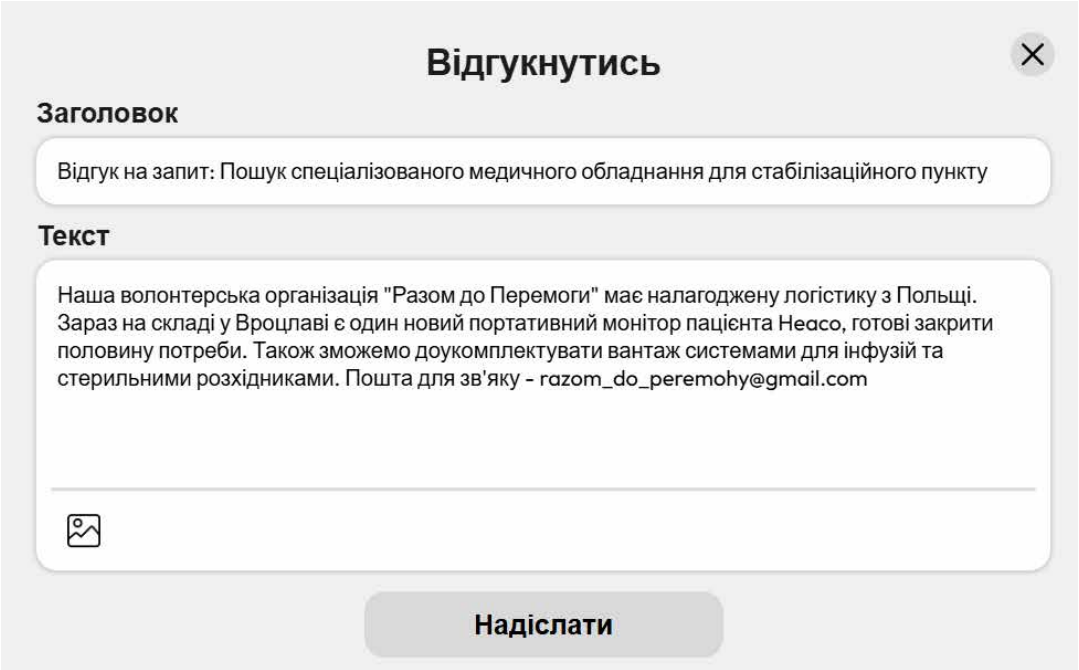



Рис. 21 Форма відгуку на запит

- Перевірка прийняття запиту – після успішного відгуку, запит позначається як прийнятий та відображається на сторінці «Прийняті запити» (рис. 23);

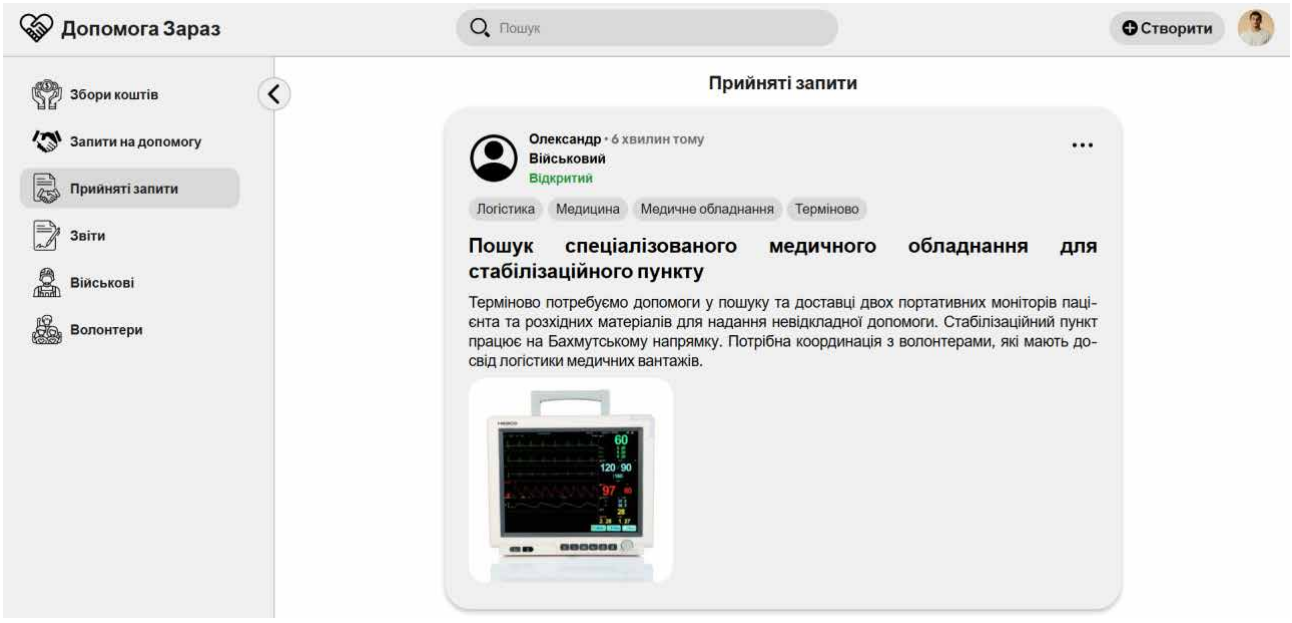



Рис. 22 Сторінка «Прийняті запити» з новим запитом у роботі

- Закриття запиту волонтером після надання допомоги та створення звіту – для цього потрібно натиснути на три крапки на картці запиту та натиснути на кнопку «Закрити», після чого відкриється форма заповнення даних для звіту про надання допомоги (рис. 23), де потрібно ввести текст звіту, а також додати зображення;

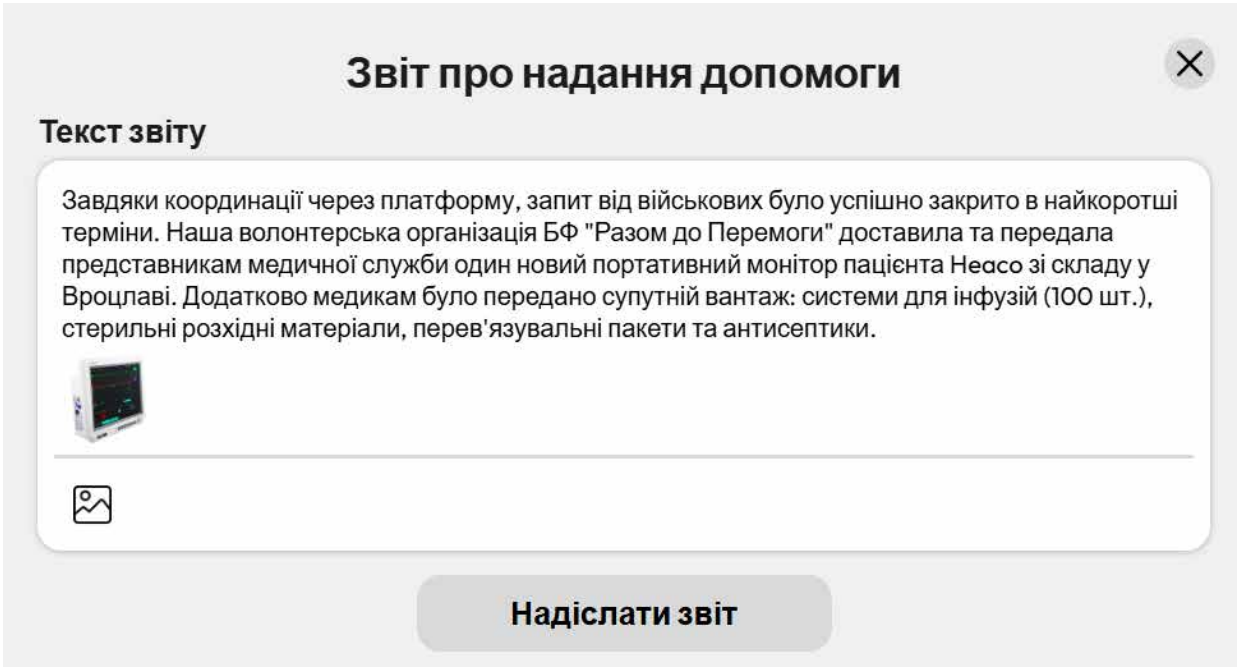



Рис. 23 Форма звіту про надання допомоги

- Перегляд створеного волонтером звіту – після успішного створення волонтером звіту, статус запиту на стороні волонтера змінюється на закритий, а згенерований звіт відображається на сторінці «Звіти» (рис. 24), де його можна детально перегляну (рис. 25.1 і рис. 25.2);

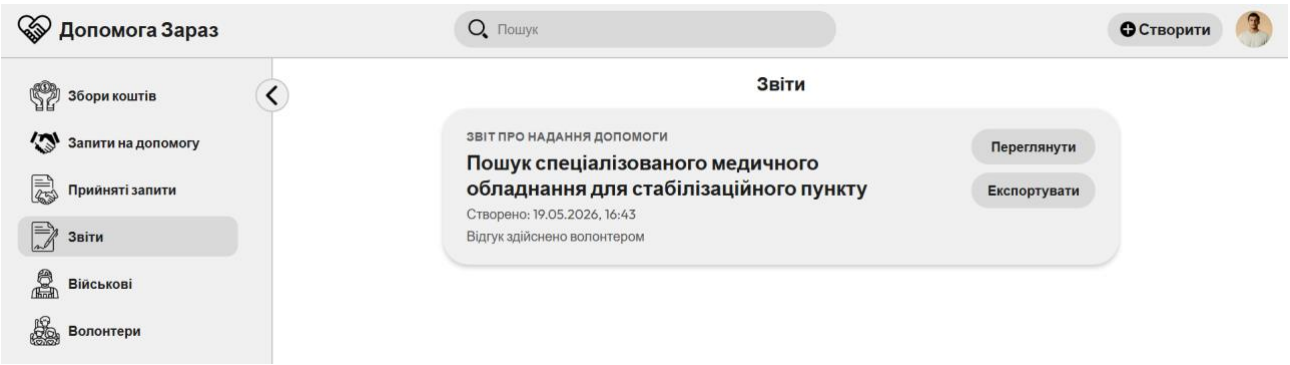



Рис. 24 Сторінка згенерованих звітів

ОФІЦІЙНИЙ ЗВІТ

# Звіт про надання допомоги

| НОМЕР ЗВІТУ | ДАТА СТВОРЕННЯ | РОЛЬ АВТОРА ЗВІТУ |
|---|---|---|
| 1 | 19.05.2026, 16:43 | Волонтер |

## Інформація про запит

### Пошук спеціалізованого медичного обладнання для стабілізаційного пункту

Автор: Олександр (Військовий)

Дата запиту: 19.05.2026, 16:08

Логістика | Медицина | Медичне обладнання | Терміново

Терміново потребуємо допомоги у пошуку та доставці двох портативних моніторів пацієнта та розхідних матеріалів для надання невідкладної допомоги. Стабілізаційний пункт працює на Бахмутському напрямку. Потрібна координація з волонтерами, які мають досвід логістики медичних вантажів.

**Фото із запиту**

## Текст звіту

Завдяки координації через платформу, запит від військових було успішно закрито в найкоротші терміни. Наша волонтерська організація БФ "Разом до Перемоги" доставила та передала представникам медичної служби один новий портативний монітор пацієнта Heaco зі складу у Вроцлаві. Додатково медикам було передано супутній вантаж: системи для інфузій (100 шт.), стерильні розхідні матеріали, перев'язувальні пакети та антисептики.

## Додані фото

Рис. 25 Детальний перегляд звіту

- Закриття запиту військовим після отримання допомоги та створення звіту – закриття та звітування військовим відбувається аналогічно волонтеру, але військовий створює звіт про отримання допомоги (рис. 26) і після цього запит закривається повністю для всіх користувачів. Створений звіт можна переглянути та перевірити на сторінці «Звіти» (рис. 27).

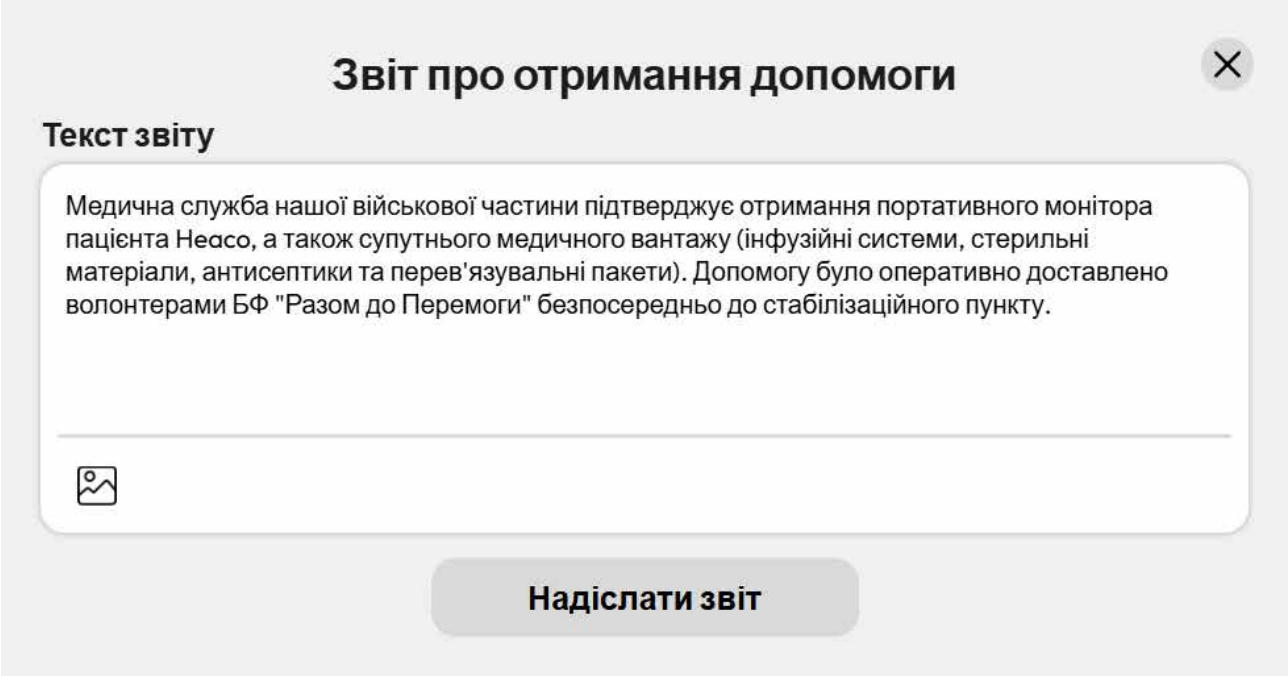



Рис. 26 Форма звіту про отримання допомоги

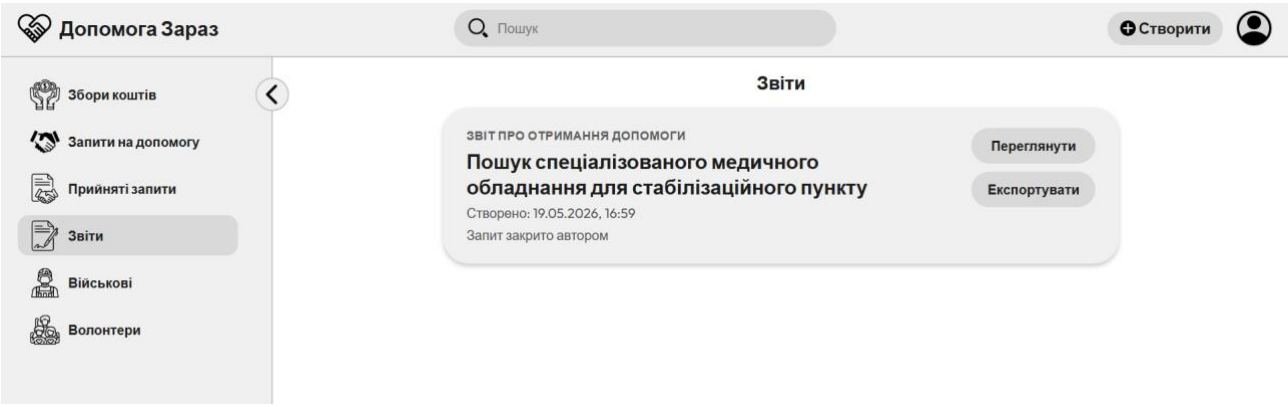



Рис. 27 Сторінки «Звіти»

Всі етапи тестування пройшли успішно без виявлення жодних критичних помилок у функціоналі, візуальному відображенні елементів користувацького інтерфейсу чи процесах передачі даних між фронтенд та бекенд частинами застосунку, що підтвердило працездатність та стабільність розробленої системи, а також її готовність до експлуатації.

## 4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення коректного та безпечного функціонування розробленої системи обліку взаємодії волонтерів і військових дуже важливим є правильна організація її архітектури, а також чітке визначення вимог до апаратного та програмного забезпечення. Для візуалізації фізичного розміщення програмних компонентів системи на апаратному забезпеченні, а також мережевих зв'язків та взаємодії між ними використовується діаграма розгортання мови UML. Вона відображає, які саме програмні модулі (артефакти) виконуються на відповідних фізичних пристроях (вузлах) та за допомогою яких протоколів (з'єднань) відбувається комунікацію між ними [26].

На рис. 28 представлено діаграму розгортання розробленої системи.

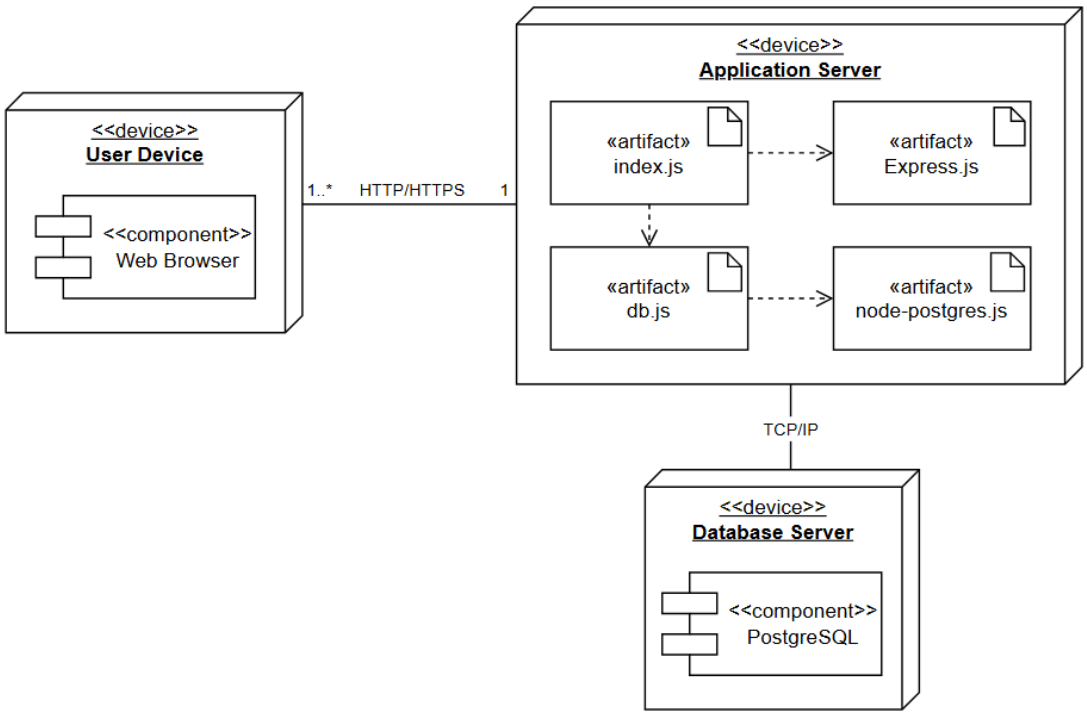



Рис. 28 Діаграма розгортання

**Опис вузлів діаграми та їх вмісту:**

- Вузол «User Device» – відображає будь-який пристрій кінцевого користувача. Він містить внутрішній компонент Web Browser, який виступає середовищем виконання для фронтенд-частини застосунку. Відповідає за відображення користувацького інтерфейсу та обробку всіх подій.

- Вузол «Application Server» – основний обчислювальний вузол, на якому зосереджена вся бізнес-логіка системи. Вузол містить наступні взаємопов’язані артефакти:
  - index.js – головний файл коду бекенду, який відповідає за роботу системи, підключення необхідних модулів. Має залежності від файлу конфігурації бази даних (db.js) та фреймворку (Express.js);
  - Express.js – фреймворк, який забезпечує REST API і спрощує маршрутизацію та обробку запитів;
  - db.js – відповідає за налаштування параметрів підключення та з’єднань до бази даних;
  - node-postgres.js – бібліотека, яка забезпечує взаємодію між середовищем Node.js та СУБД PostgreSQL.
- Вузол «Database Server» - сервер, призначений для надійного розгортання та швидкого доступу до бази даних. На цьому вузлі функціонує реляційна СУБД PostgreSQL, яка забезпечує роботу з даними в системі.

**Опис мережевих з’єднань:**

- Вузли «User Device» та «Application Server»:
  - Протокол – HTTP/HTTPS;
  - Характеристика: тип зв’язку 1..* до 1 вказує, що до одного сервера застосунків може підключатись багато клієнтський пристроїв. За допомогою цього з’єднання веббраузер надсилає асинхронні запити та отримує відповіді і дані від бекенду, для подальшого використання на фронтенді.
- Вузли «Application Server» та «Database Server»:
  - Протокол – TCP/IP;
  - Характеристика – з’єднання між сервером застосунків та сервером СУБД. Через який бекенд передає SQL-запити на

роботу з даними до PostgreSQL, який виконує отримані запити та повертає оброблені дані.

Представлена діаграма розгортання відображає чіткий розподіл функціональних обов'язків між обчислювальними вузлами системи, мережеві зв'язки та протоколи взаємодії між ними. Це забезпечує швидкість обробки запитів, стабільність функціонування системи під час навантажень та спрощує подальшу підтримку та масштабування.

**Вимоги до серверної частини:**

- **Мінімальні апаратні вимоги:**
  - Операційна система – завдяки кросплатформеності Node.js та PostgreSQL, система може коректно працювати на більшості сучасних операційних систем;
  - Процесор – 2-ядерний 64-бітний процесор, наприклад Intel Core i3 або аналогічний;
  - Оперативна пам'ять – 4 ГБ;
  - Дисковий простір – 64 ГБ.
- **Програмні вимоги:**
  - Node.js (v18.0.0+) – асинхронне середовище для виконання JavaScript коду;
  - npm (v9.0.0+) – менеджер пакетів;
  - PostgreSQL (v14.0+) – реляційна СУБД;
  - Express.js (v4.18.0+) – фреймворк для організації архітектури REST API та маршрутизації HTTP-запитів;
  - node-postgres.js (v8.0.0+) – бібліотека для забезпечення стабільного підключення та виконання SQL-запитів до СУБД;
- **Хостинг –** для розгортання та забезпечення доступності елементів системи використовуються наступні хмарні платформи:
  - Render – використовується для хостингу фронтенд та бекенд частин вебзастосунку. Сервіс забезпечує автоматичне розгортання коду, а також надає інструменти моніторингу [27];

- Neon – платформа, на якій розгортається реляційна база даних PostgreSQL. Сервіс забезпечує автоматичне масштабування обчислювальних ресурсів під навантаження, швидку обробку запитів, а також автоматично створює резервні копії [28].

**Вимоги до клієнтських пристроїв:**

- **Апаратні вимоги:**
  - Тип пристрою – персональні комп'ютери, ноутбуки, планшети та смартфони;
  - Оперативна пам'ять – від 512 МБ;
  - Дисплей – завдяки реалізації адаптивної верстки інтерфейс коректно відображається на екранах більшості розмірів. Мінімальна ширина екрана становить 320px.
- **Програмні вимоги** – для коректної візуалізації та роботи застосунку користувачу необхідний будь-який сучасний веббраузер з підтримкою HTML5, CSS3 та ES6+.

## 4.3 Склад інсталяційного пакету

Для розгортання та налаштування клієнтської та серверної частин вебзастосунку використовується інсталяційний пакет, який містить усі необхідні компоненти, конфігураційні файли для коректної роботи системи.

До інсталяційного пакету вебзастосунку входять наступні каталоги та файли:

- Файли керування залежностями та конфігурацією середовище – включають файли package.json та package-lock.json, які визначають перелік сторонніх бібліотек, драйверів і їх версій, а також файл .env.example, що містить шаблон конфігураційних змінних для підключення до хмарних платформ розгортання;
- Клієнтська частина – містить файли HTML-розмітки, каталоги CSS-стилів для забезпечення візуалізації та стилізації інтерфейсу, а також

клієнтські JS-скрипти, що відповідають за динамічність інтерфейсу та надсилання асинхронних Fetch-запитів;

- Серверна частина – складається з головного файлу index.js, який налаштовує роботу Express.js та обробку та маршрутизацію запитів, а також файлу db.js для забезпечення з’єднання із базою даних PostgreSQL та обробки SQL-запитів.

Сформований склад інсталяційного пакету дозволяє полегшити розгортання та конфігурації вебзастосунку з нуля. Наявність усіх необхідних файлів конфігурації та залежностей зменшує ризики виникнення конфліктів та гарантує правильну роботу системи на віддалених хмарних серверах.

# ВИСНОВКИ

Під час виконання бакалаврської роботи на тему «Фронтенд частина системи обліку взаємодії волонтерів і військових було успішно здійснено аналіз, проєктування, розробку та тестування вебзастосунку. Основною метою розробки було створення інтерфейсу, який забезпечить швидку комунікацію між обома сторонами взаємодії та спростить процес пошуку, створення і обробки запитів на допомогу та зборів коштів.

Аналіз та моделювання предметної області допомогли виявити основних акторів, їх функціональні можливості та способи взаємодії між ними. Огляд існуючих рішень дозволив визначити основні недоліки, такі як відсутність централізованого обліку, складність пошуку інформації, високі ризики шахрайства та недостатня прозорість взаємодії. На основі отриманих результатів аналізу було сформовано постановку задачі і визначено основні вимоги до системи, що стало основою для подальшого проєктування.

Визначено принципи побудови вебзастосунку, описано взаємодію між фронтенд та бекенд частинами на основі клієнт-серверної архітектури та наведено механізми взаємодії та обміну даними, а саме REST API, HTTP-запити, JSON та Fetch API. Також було спроєктовано архітектуру системи у вигляді діаграми пакетів, що дозволила візуалізувати ієрархію майбутнього продукту та визначити зони відповідальності модулів шляхом розділення їх на окремі пакети.

При розробці програмного забезпечення було описано основні абстракції предметної області для визначення властивостей та обов’язків об’єктів застосунку. За допомогою діаграми класів описано статичну структуру системи, а діаграми кооперацій дозволили описати динамічну взаємодію об’єктів у відповідних сценаріях. Візуалізація модульного розподілу системи за допомогою діаграми компонентів дозволила визначити фізичну організацію застосунку та розподілити функціонал. Технологічний стек для розробки фронтенду складається з середовища Visual Studio Code, HTML5, CSS3 та JavaScript для реалізації інтерфейсу, технологій Fetch API та JWT для асинхронного обміну даними і безпеки сесій користувачів, а також Git та GitHub для контролю версій.

На основі побудованих блок-схем алгоритмів було програмно реалізовано процеси клієнтської частини системи.

Перед впровадженням та експлуатацією системи була проведена детальна перевірка фронтенду за допомогою різних видів тестування, що дозволило завчасно виявити та виправити помилки. Побудована діаграма розгортання дала можливість візуалізувати фізичне розміщення програмних компонентів на апаратному забезпеченні для оптимізації взаємодії між ними через відповідні протоколи. Далі було визначено вимоги до програмного та апаратного забезпечення, а також описано методи хостингу окремих частин системи. Крім цього, сформовано інсталяційний пакет, що забезпечує швидке розгортання.

У кінцевому результаті була повністю досягнута мета бакалаврської роботи. Розроблена фронтенд частина системи обліку взаємодії волонтерів і військових повністю відповідає технічному завданню та визначеним вимогам. Створений застосунок надає зручний та адаптивний інтерфейс для вирішення актуальних проблем централізації інформації та шахрайства у сфері волонтерської допомоги військовим.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Волонтерство для ЗСУ: як працюють збори, запити від підрозділів і передача допомоги – [Електронний ресурс] – режим доступу: https://www.blue-bird.tech/news/volonterstvo-dlya-zsu-yak-praczyuyut-zbori-zapiti-vid-pidrozdiliv-i-peredacha-dopomogi/ (дата звернення: 04.05.2026)
2. What is Unified Modeling Language (UML)? – [Електронний ресурс] – режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (дата звернення: 06.05.2026)
3. Use Case Diagram – Unified Modeling Language (UML) – [Електронний ресурс] – режим доступу: https://www.geeksforgeeks.org/system-design/use-case-diagram/ (дата звернення: 06.05.2026)
4. What is Activity Diagram? – [Електронний ресурс] – режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/ (дата звернення: 07.05.2026)
5. People`s Project Всеукраїнський центр волонтерів – [Електронний ресурс] – режим доступу: https://www.peoplesproject.com/ (дата звернення: 07.05.2026)
6. Офіційний сайт Фонду «Повернись живим» – [Електронний ресурс] – режим доступу: https://savelife.in.ua/ (дата звернення: 07.05.2026)
7. What is a Web Application? Complete Guide to Web Apps – [Електронний ресурс] – режим доступу: https://www.codecademy.com/article/what-is-a-web-application-complete-guide-to-webapps (дата звернення: 08.05.2026)
8. Frontend Vs Backend Development – [Електронний ресурс] – режим доступу: https://www.geeksforgeeks.org/blogs/frontend-vs-backend/ (дата звернення: 08.05.2026)
9. Client-Server Architecture Explained with Examples, Diagrams, and Real-World Applications – [Електронний ресурс] – режим доступу:

https://medium.com/nerd-for-tech/client-server-architecture-explained-with-examples-diagrams-and-real-world-applications-407e9e04e2d1 (дата звернення: 08.05.2026)

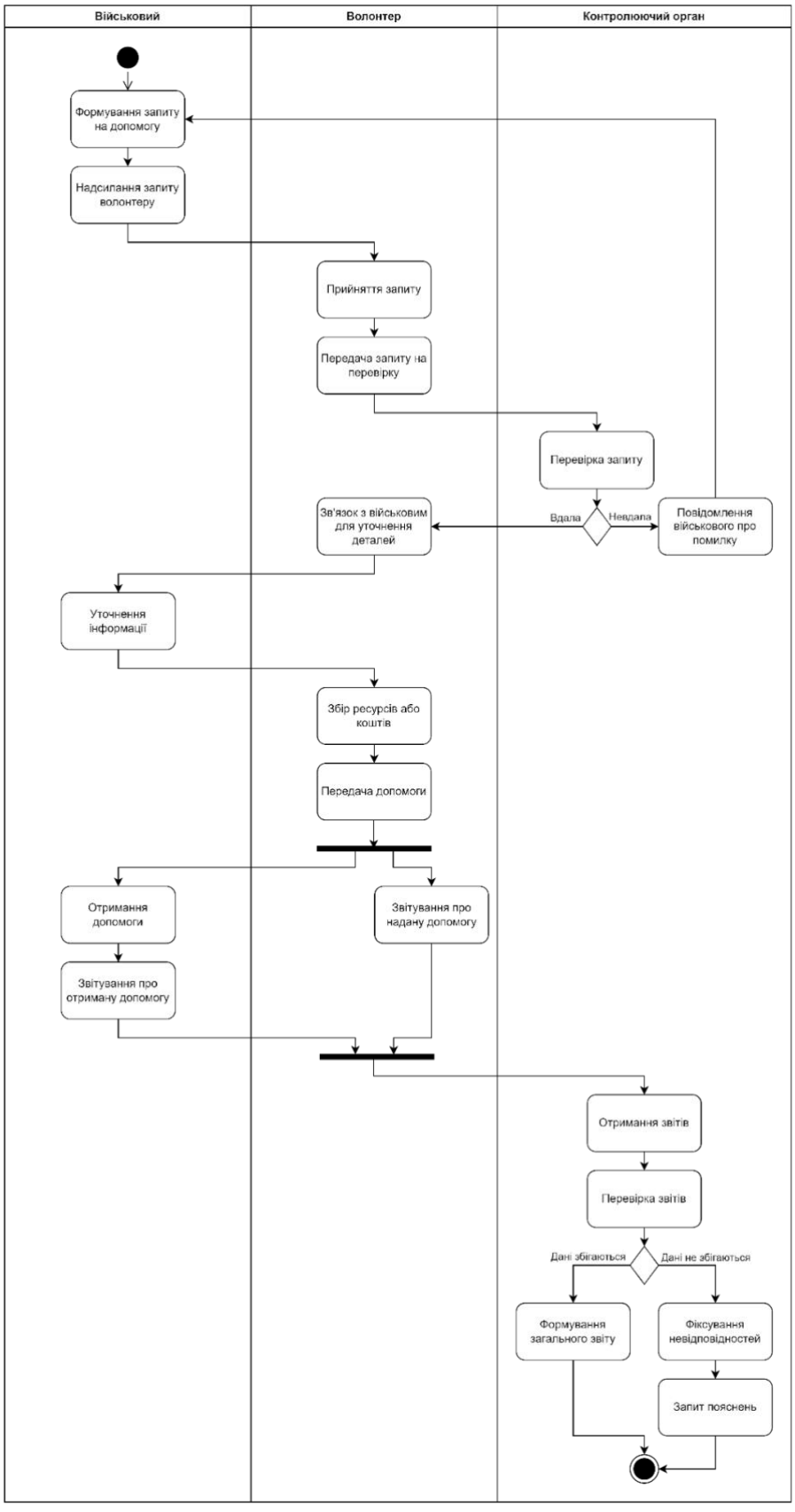
10. What is REST API (RESTful API)? Explained – [Електронний ресурс] – режим доступу: https://www.codecademy.com/article/what-is-rest-api (дата звернення: 08.05.2026)
11. Introducing JSON – [Електронний ресурс] – режим доступу: https://www.json.org/json-en.html (дата звернення: 08.05.2026)
12. Fetch API – [Електронний ресурс] – режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 08.05.2026)
13. What is Package Diagram? – [Електронний ресурс] – режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/ (дата звернення: 09.05.2026)
14. What is Class Diagram? – [Електронний ресурс] – режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/ (дата звернення: 10.05.2026)
15. Component Based Diagram – Unified Modeling Language (UML) – [Електронний ресурс] – режим доступу: https://www.geeksforgeeks.org/software-engineering/component-based-diagram/ (дата звернення: 12.05.2026)
16. What is Visual Studio Code, What is it Used For? – [Електронний ресурс] – режим доступу: https://coderspace.io/en/blog/what-is-visual-studio-code-what-is-it-used-for/ (дата звернення: 13.05.2026)
17. What Is HTML and How Does It Work? – [Електронний ресурс] – режим доступу: https://dev.to/waynetasaki/what-is-html-and-how-does-it-work-32p9 (дата звернення: 13.05.2026)
18. Semantic HTML – [Електронний ресурс] – режим доступу: https://web.dev/learn/html/semantic-html (дата звернення: 13.05.2026)

19. What is CSS? – [Електронний ресурс] – режим доступу: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/What_is_CSS (дата звернення: 13.05.2026)
20. Що таке JavaScript і для чого він потрібен – [Електронний ресурс] – режим доступу: https://goit.global/ua/articles/shcho-take-javascript-i-dlia-choho-vin-potriben/ (дата звернення: 13.05.2026)
21. Introduction to JSON Web Tokens – [Електронний ресурс] – режим доступу: https://www.jwt.io/introduction#what-is-json-web-token (дата звернення: 13.05.2026)
22. About GitHub and Git – [Електронний ресурс] – режим доступу: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git (дата звернення: 13.05.2026)
23. Chrome DevTools – [Електронний ресурс] – режим доступу: https://developer.chrome.com/docs/devtools/overview (дата звернення: 13.05.2026)
24. Flowchart Tutorial (with Symbols, Guide and Examples) – [Електронний ресурс] – режим доступу: https://www.visual-paradigm.com/tutorials/flowchart-tutorial/ (дата звернення: 14.05.2026)
25. Тестування веб-проєктів: основні етапи та поради – [Електронний ресурс] – режим доступу: https://qalight.ua/baza-znaniy/testuvannya-veb-proektiv-osnovni-etapi-ta-poradi/ (дата звернення: 16.05.2026)
26. UML Deployment Diagrams – [Електронний ресурс] – режим доступу: https://www.uml-diagrams.org/deployment-diagrams.html (дата звернення: 18.05.2026)
27. Deploying on Render – [Електронний ресурс] – режим доступу: https://render.com/docs/deploys (дата звернення: 19.05.2026)
28. Neon documentation – [Електронний ресурс] – режим доступу: https://neon.com/docs/introduction (дата звернення: 19.05.2026)

# ДОДАТКИ

## ДОДАТОК А

## Діаграма активності



# ДОДАТОК Б

## HTML код розмітки форм авторизації та реєстрації

```
<h1 class="modal-heading">Вхід</h1>
    <form class="form active-modal" id="login-form">
        <div class="text-input">
            <label for="email" class="input-title">Електронна пошта</label>
            <input type="email" id="email_login" name="email" required>
        </div>
        <div class="text-input password-wrapper">
            <label for="password" class="input-title">Пароль</label>
            <input type="password" id="password_login" name="password"
required>
            <button type="button" class="toggle-password" aria-
label="Показати/Сховати пароль">
                <img src="/public/images/hidden-icon.png" alt="password
hidden">
            </button>
        </div>
        <button type="submit" class="submit-button">Увійти</button>
        <div class="footer">
            <p>Немає акаунту?</p>
            <a href="#">Зареєструватись</a>
        </div>
    </form>
    <form class="form" id="registration-form">
        <div class="text-input">
            <label for="role" class="input-title">Роль</label>
            <select name="role" id="role">
                <option value="vo" selected>Волонтер</option>
                <option value="mi">Військовий</option>
            </select>
        </div>
        <div class="text-input">
            <label for="email" class="input-title">Електронна пошта</label>
            <input type="email" id="email_registration" name="email"
required>
        </div>
        <div class="text-input">
            <label for="username" class="input-title">Ім'я / Назва</label>
            <input type="text" id="username_registration" name="username"
required>
        </div>
        <div class="text-input">
            <label for="rnokpp" class="input-title">РНОКПП</label>
            <input type="text" id="rnokpp_registration" name="rnokpp"
minlength="10" maxlength="10" pattern="[0-9]*" required>
        </div>
```

```
        <div class="text-input password-wrapper">
            <label for="password" class="input-title">Пароль</label>
            <input type="password" id="password-registration"
name="password" minlength="8" required>
            <button type="button" class="toggle-password" aria-
label="Показати/Сховати">
                <img src="/public/images/hidden-icon.png" alt="password
hidden">
            </button>
        </div>
        <div class="text-input password-wrapper">
            <label for="password-confirmation" class="input-
title">Підтвердження пароля</label>
            <input type="password" id="password-confirmation"
name="password-confirmation" minlength="8" oninput="validatePassword()"
required>
            <button type="button" class="toggle-password" aria-
label="Показати/Сховати">
                <img src="/public/images/hidden-icon.png" alt="password
confirmation hidden">
            </button>
        </div>
        <button type="submit" class="submit-button">Зареєструватись</button>
        <div class="footer">
            <p>Вже є акаунт?</p>
            <a href="#" id="register">Увійти</a>
        </div>
    </form>
```

# ДОДАТОК В

## Код для керування процесом авторизації, реєстрації та забезпечення взаємодії між фронтендом та API бекенду

```
function wireLoginRegistrationModal() {
  const modal = document.getElementById('login-reg-modal');
  const heading = modal?.querySelector('.modal-heading');
  const loginForm = document.getElementById('login-form');
  const registrationForm = document.getElementById('registration-form');
  const regLink = loginForm?.querySelector('.footer a');
  const loginLink = registrationForm?.querySelector('.footer a');

  initTogglePassword();
  bindSharedModalClose();

  regLink?.addEventListener('click', (e) => {
    e.preventDefault();
    heading.textContent = 'Реєстрація';
    registrationForm.classList.add('active-modal');
    loginForm.classList.remove('active-modal');
  });

  loginLink?.addEventListener('click', (e) => {
    e.preventDefault();
    heading.textContent = 'Вхід';
    loginForm.classList.add('active-modal');
    registrationForm.classList.remove('active-modal');
  });

  loginForm?.addEventListener('submit', async (e) => {
    e.preventDefault();
    const submitButton = loginForm.querySelector('.submit-button');
    const email = loginForm.querySelector('#email_login')?.value.trim();
    const password = loginForm.querySelector('#password_login')?.value;
    try {
      window.LoadingUi?.setButtonLoading(submitButton, true, 'Входимо...');
      const response = await fetch('/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        credentials: 'same-origin',
        body: JSON.stringify({ email, password }),
      });

      const result = await response.json();
      if (!response.ok) {
        alert(result.message || 'Невірний email або пароль.');
        return;
```

```
      }
      window.AuthState?.setUser(result.user);
      closeModal();
    } catch (error) {
      console.error('Помилка входу:', error);
      alert('Сервер недоступний. Спробуйте пізніше.');
    } finally {
      window.LoadingUi?.setButtonLoading(submitButton, false);
    }
  });
  registrationForm?.addEventListener('submit', async (e) => {
    e.preventDefault();
    const submitButton = registrationForm.querySelector('.submit-button');
    const fullName =
registrationForm.querySelector('#username_registration')?.value.trim();
    const rnokpp =
registrationForm.querySelector('#rnokpp_registration')?.value.trim();
    const email =
registrationForm.querySelector('#email_registration')?.value.trim();
    const password = registrationForm.querySelector('#password-
registration')?.value;
    const roleId = registrationForm.querySelector('#role')?.value;
    try {
      window.LoadingUi?.setButtonLoading(submitButton, true,
'Реєструємо...');
      const response = await fetch('/auth/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        credentials: 'same-origin',
        body: JSON.stringify({
          full_name: fullName,
          rnokpp,
          email,
          password,
          role_id: roleId,
        }),
      });
      const result = await response.json();
      if (!response.ok) {
        alert(result.message || 'Помилка реєстрації.');
        return;
      }
      registrationForm.reset();
      window.AuthState?.setUser(result.user);
      closeModal();
    } catch (error) {
      console.error('Помилка реєстрації:', error);
      alert('Сталася помилка. Спробуйте пізніше.');
```

```
    } finally {
      window.LoadingUi?.setButtonLoading(submitButton, false);
    }
  });
}
```

# ДОДАТОК Г

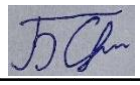
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**
**Декларація про академічну доброчесність та використання ШІ**

Я, Бачинський Олександр Анатолійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Фронтенд частина системи обліку взаємодії волонтерів і військових» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що:

- інструменти ШІ ChatGPT, Gemini, DeepL були використані для:
  - перекладу іншомовних джерел;
  - стилістичного редагування та перевірки граматики;
  - допомоги у написанні/відлагодженні програмного коду.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«22» травня 2026 р.  **Олександр БАЧИНСЬКИЙ**

(підпис)