**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: «Підсистема адміністрування та реалізації бізнес-логіки системи обліку взаємодії волонтерів та 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** _____ **Антон 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 Антону Олексійовичу

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Підсистема адміністрування та реалізації бізнес-логіки системи обліку взаємодії волонтерів та 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** | _____ (підпис) | **Матвій ГОРБАЧ** |
| **Консультант бакалаврської кваліфікаційної роботи** к.т.н., доцент | _____ (підпис) | **Белла ГОЛУБ** |
| **Завдання взяв до виконання** | _____ (підпис) | **Антон БАЧИНСЬКИЙ** |

# Зміст

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

**БД** — база даних

**ОЗП** — оперативна запам’ятовувальна пам’ять

**РНОКПП** — реєстраційний номер облікової картки платника податків

**СУБД** — система керування базами даних

**API** (*Application Programming Interface*) — прикладний програмний інтерфейс

**HTTP** (*HyperText Transfer Protocol*) — протокол передавання гіпертексту

**HTTPS** (*HyperText Transfer Protocol Secure*) — захищена версія протоколу HTTP

**JSON** (*JavaScript Object Notation*) — текстовий формат обміну даними

**npm** (*Node Package Manager*) — менеджер пакетів середовища Node.js

**REST** (*Representational State Transfer*) — архітектурний стиль побудови вебсервісів

**SQL** (*Structured Query Language*) — мова структурованих запитів

**SSD** (*Solid-State Drive*) — твердотільний накопичувач

**UML** (*Unified Modeling Language*) — уніфікована мова моделювання

# ВСТУП

**Актуальність.** Організація оперативного й прозорого обліку матеріальних ресурсів, що передаються Силам оборони України в межах волонтерської діяльності, є критично важливою для підтримання обороноздатності держави. Створення жорстких механізмів контролю дозволяє мінімізувати ризики компрометації даних та запобігти правопорушенням: від маніпулювання потребами до несанкціонованого розподілу допомоги. Наявні децентралізовані канали комунікації (соціальні мережі, месенджери) не мають інструментів контролю доступу, рольового розмежування та формалізованого супроводу заявок, що призводить до хаотичності, дублювання потреб та підвищення ризиків зловживань.

Вирішення окреслених проблем потребує розробки спеціалізованого програмного середовища з жорстко регламентованою бізнес-логікою. Надійний серверний контроль життєвого циклу запитів та безпечне збереження даних зумовлюють актуальність даного дослідження.

Робота виконується в межах командного проєкту з розробки вебсистеми «Допомога Зараз». Особистий внесок автора полягає у проєктуванні реляційної бази даних, розробці архітектури REST API, реалізації серверної бізнес-логіки, захищених механізмів автентифікації та хмарного розгортання бекенду додатка. Клієнтська частина (фронтенд) реалізується іншим виконавцем і розглядається окремо.

Об’єктом розробки є процес обліку, координації та документування взаємодії між військовими підрозділами, волонтерами та контролюючими органами.

Предметом дослідження є методи побудови безпечної серверної архітектури, проектування схем баз даних, управління сесіями та розгортання бізнес-логіки вебсистем у хмарній інфраструктурі.

**Мета.** Метою бакалаврської роботи є розробка підсистеми адміністрування та реалізації бізнес-логіки системи обліку взаємодії волонтерів

і військових, що забезпечує безпеку збереження даних, захист від несанкціонованого доступу та стабільну асинхронну обробку запитів у хмарному середовищі.

Практичне значення отриманих результатів полягає у створенні захищеної серверної підсистеми, яка автоматизує життєвий цикл заявок через тристоронній контроль, виключає маніпуляції зі звітністю та забезпечує стабільну роботу в умовах хмарної інфраструктури.

**Апробації програмного продукту.** Результати розробки були опубліковані у вигляді тез на тему «Бекенд 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»

Пояснювальна записка складається зі вступу, чотирьох розділів, висновків, списку використаних джерел із 23 найменувань та додатків. У першому розділі проаналізовано предметну область, розглянуто особливості взаємодії між військовими та волонтерами, проведено порівняльний аналіз існуючих аналогів і виконано UML-моделювання системи. У другому розділі подано проєктування архітектури підсистеми та бази даних, обґрунтовано вибір СУБД і описано формування фізичної моделі БД. Третій розділ присвячено вибору технологічного стеку, архітектурі серверної частини, організації REST API, реалізації механізмів автентифікації, рольового доступу та бізнес-логіки системи. У четвертому розділі наведено результати тестування, визначено вимоги до апаратного та програмного забезпечення, а також описано склад інсталяційного пакету й особливості розгортання системи. Загальний обсяг роботи становить 75 сторінок, вона містить 29 рисунків та 7 таблиць.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Характеристика предметної області та аналіз бізнес-процесів

На сучасному етапі взаємодія між волонтерськими організаціями та військовими підрозділами в Україні значною мірою здійснюється через соціальні мережі, месенджери та інші загальнодоступні цифрові канали. Хоча такі засоби забезпечують оперативність поширення інформації, вони не створюють належних умов для структурованого обліку потреб, контролю стану їх виконання та формування єдиного інформаційного простору взаємодії. Унаслідок цього інформація про запити на допомогу, волонтерські ініціативи та результати їх виконання існує фрагментарно, що ускладнює відстеження відповідальних осіб і знижує прозорість процесу.

Важливою проблемою є також відсутність єдиного серверного середовища, у межах якого могли б бути закріплені правила доступу, рольові обмеження та логіка обробки даних. Через це в неспеціалізованих каналах комунікації фактично відсутній формалізований механізм цифрової фіксації результатів взаємодії, що ускладнює як внутрішній аналіз, так і подальший зовнішній контроль.

Для розв’язання зазначених проблем у межах дипломного проєкту запропоновано та реалізовано централізовану вебсистему, орієнтовану на підтримку взаємодії між військовими та волонтерами. Система базується на рольовій моделі доступу, у межах якої виділено дві основні категорії користувачів: військовослужбовця та волонтера. Військовослужбовець формує запити на допомогу, що відображають актуальні потреби підрозділу, тоді як волонтер переглядає такі запити, відгукується на них і, за необхідності, створює власні збори коштів.

Ключовою особливістю системи є перенесення основних правил предметної області на серверний рівень. Це означає, що допустимі дії

користувача визначаються не лише інтерфейсом клієнтської частини, а й перевіряються серверною бізнес-логікою. Зокрема, військовослужбовець може створювати запити на допомогу, але не може відгукуватися на них як виконавець, тоді як волонтер, навпаки, може залишати відгуки, але не може створювати запити від імені військової сторони.

У реалізованій моделі життєвий цикл запиту має послідовний характер. Спочатку військовий формує запит із описом потреби, тегами та, за необхідності, зображеннями. Після цього волонтер може залишити відгук, що фіксує намір узяти запит у роботу. Наступним етапом є формування звітності: автор запиту може підтвердити факт отримання допомоги, а волонтер — факт її надання. Створення звіту супроводжується зміною стану пов’язаного об’єкта, що дозволяє фіксувати завершення відповідного етапу взаємодії.

Особливе значення в системі має підсистема звітності. Дані для звітів зберігаються у базі даних та можуть бути переглянуті в межах системи, що забезпечує цифрову фіксацію результатів взаємодії між сторонами. У контексті даного дослідження контролюючий орган доцільно розглядати як зовнішній державний або уповноважений суб’єкт, який за потреби може ознайомитися з уже сформованою звітною інформацією.

Отже, запропонована вебсистема забезпечує перехід від децентралізованої моделі взаємодії до централізованого цифрового середовища з визначеними ролями, правилами доступу та фіксованими станами обробки запитів. Це дозволяє впорядкувати інформаційні потоки, покращити контроль за проходженням запиту та сформувати базовий механізм документального підтвердження результатів надання допомоги.

## 1.2 Порівняльний аналіз існуючих аналогів та подібних вебсистем

У межах передпроєктного дослідження було проведено аналіз цифрових рішень, які можуть використовуватися для координації допомоги, збору коштів та інформаційної взаємодії між учасниками волонтерської діяльності. Метою

аналізу було виявлення функціональних обмежень існуючих підходів і визначення особливостей, які доцільно врахувати під час розробки вебсистеми «Допомога Зараз».

Для обґрунтування доцільності створення власної системи аналоги доцільно порівнювати за такими критеріями:

- доступність для широкого кола користувачів;
- підтримка рольового розмежування дій;
- можливість контролю станів запитів;
- наявність засобів цифрової звітності;
- адаптованість до специфіки українського середовища.

До першої групи аналогів належать внутрішні інформаційні системи великих благодійних фондів і волонтерських організацій. Їх перевагою є централізований облік ресурсів і впорядкованість процесів, однак такі рішення зазвичай орієнтовані на роботу в межах однієї організації та не підтримують відкриту взаємодію між незалежними військовими й волонтерами [17, 18].

Другу групу становлять міжнародні платформи краудфандингу. Вони мають надійну інфраструктуру та зручні фінансові механізми, проте переважно зосереджені на зборі коштів і не враховують специфіку матеріальних запитів, відгуків і звітів у єдиному процесі [19].

До третьої групи належать вітчизняні інформаційні платформи та агрегатори допомоги. Їх перевагами є доступність і простота використання, але в більшості випадків вони виконують лише функцію інформування та не забезпечують серверного контролю станів, формалізованої рольової моделі й цифрової звітності [20, 21].

Окремо слід виділити соціальні мережі та месенджери, які часто використовуються для координації допомоги на практиці. Вони забезпечують швидкий обмін інформацією, однак не підтримують централізоване збереження даних, структурований облік запитів і формалізований контроль результатів взаємодії [22, 23].

Для узагальнення результатів проведеного аналізу доцільно подати порівняння основних класів рішень у табличній формі.

*Таблиця 1.2.1*

**Порівняльний аналіз існуючих аналогів**

| **Клас рішень** | **Переваги** | **Недоліки** | **Що не покриває відносно системи “Допомога Зараз”** |
|---|---|---|---|
| Внутрішні системи великих благодійних фондів | Висока організаційна впорядкованість, централізований облік ресурсів, внутрішній контроль процесів | Закритість, орієнтація на одну організацію, відсутність відкритої реєстрації | Не підтримують публічну взаємодію між незалежними військовими та волонтерами |
| Міжнародні платформи краудфандингу | Надійна інфраструктура, зручні фінансові механізми, масштабованість | Орієнтація переважно на фінансові збори, відсутність підтримки матеріальних запитів і звітності за фактом передачі допомоги | Не підтримують зв’язок «запит – відгук – звіт» у межах єдиного процесу |

*Продовження Таблиці 1.2.1*

| **Клас рішень** | **Переваги** | **Недоліки** | **Що не покриває відносно системи “Допомога Зараз”** |
|---|---|---|---|
| Вітчизняні інформаційні платформи та агрегатори допомоги | Простота використання, швидкість публікації інформації, доступність | Обмежена функціональність, відсутність серверного контролю станів, слабка формалізація ролей | Не забезпечують рольову модель, контроль життєвого циклу запиту та цифрову звітність |
| Соціальні мережі та месенджери | Максимальна доступність, швидке поширення інформації, низький поріг входу | Хаотичність даних, дублювання запитів, відсутність структурованого обліку | Не підтримують централізоване збереження даних, контроль статусів і формалізовану звітність |
| Вебсистема «Допомога Зараз» | Рольове розмежування, серверний контроль логіки, підтримка запитів, відгуків і звітів | Потребує подальшого розвитку в частині масштабування та зовнішнього контролю | Орієнтована саме на координацію взаємодії між військовими та волонтерами |

Наведене порівняння показує, що існуючі аналоги, попри окремі переваги, не забезпечують одночасної наявності відкритої взаємодії між користувачами, рольового розмежування дій, серверного контролю станів запиту та механізму цифрової звітності. Саме поєднання цих властивостей формує функціональну нішу, яку покликана заповнити вебсистема «Допомога Зараз».

Отже, результати проведеного аналізу свідчать про наявність функціональної ніші для спеціалізованої вебсистеми, яка поєднувала б відкриту модель реєстрації користувачів із серверним контролем бізнес-логіки, рольовим розмежуванням дій, підтримкою життєвого циклу запиту та механізмом формування цифрової звітності. Саме така концепція покладена в основу розроблюваної системи «Допомога Зараз», у межах якої акцент зроблено не лише

на публікації потреб, а й на структурованому супроводі взаємодії між військовими та волонтерами від моменту створення запиту до фіксації результатів його виконання.

## 1.3 Об’єктно-орієнтований аналіз та UML-моделювання системи

Проєктування серверної підсистеми вебсистеми «Допомога Зараз» здійснюється із застосуванням об’єктно-орієнтованого аналізу та UML-моделювання [2]. Використання UML дає змогу формалізувати вимоги до системи, визначити її функціональні межі, описати основні сутності предметної області та відобразити ключові сценарії взаємодії користувачів із серверною частиною.

На початковому етапі моделювання формується діаграма прецедентів, яка відображає зовнішніх акторів системи та доступні їм функції. У поточній реалізації основними акторами є військовослужбовець і волонтер. Військовослужбовець може реєструватися, входити в систему, створювати запити на допомогу, редагувати власні дописи, переглядати відгуки та формувати звіт про отримання допомоги. Волонтер може реєструватися, входити, переглядати відкриті запити, створювати відгуки, переглядати прийняті запити, створювати збори коштів і формувати звіт про надання допомоги. Контролюючий орган у межах даної моделі розглядається як перспективний зовнішній актор, який у майбутньому може отримати доступ до перегляду сформованої звітної інформації.

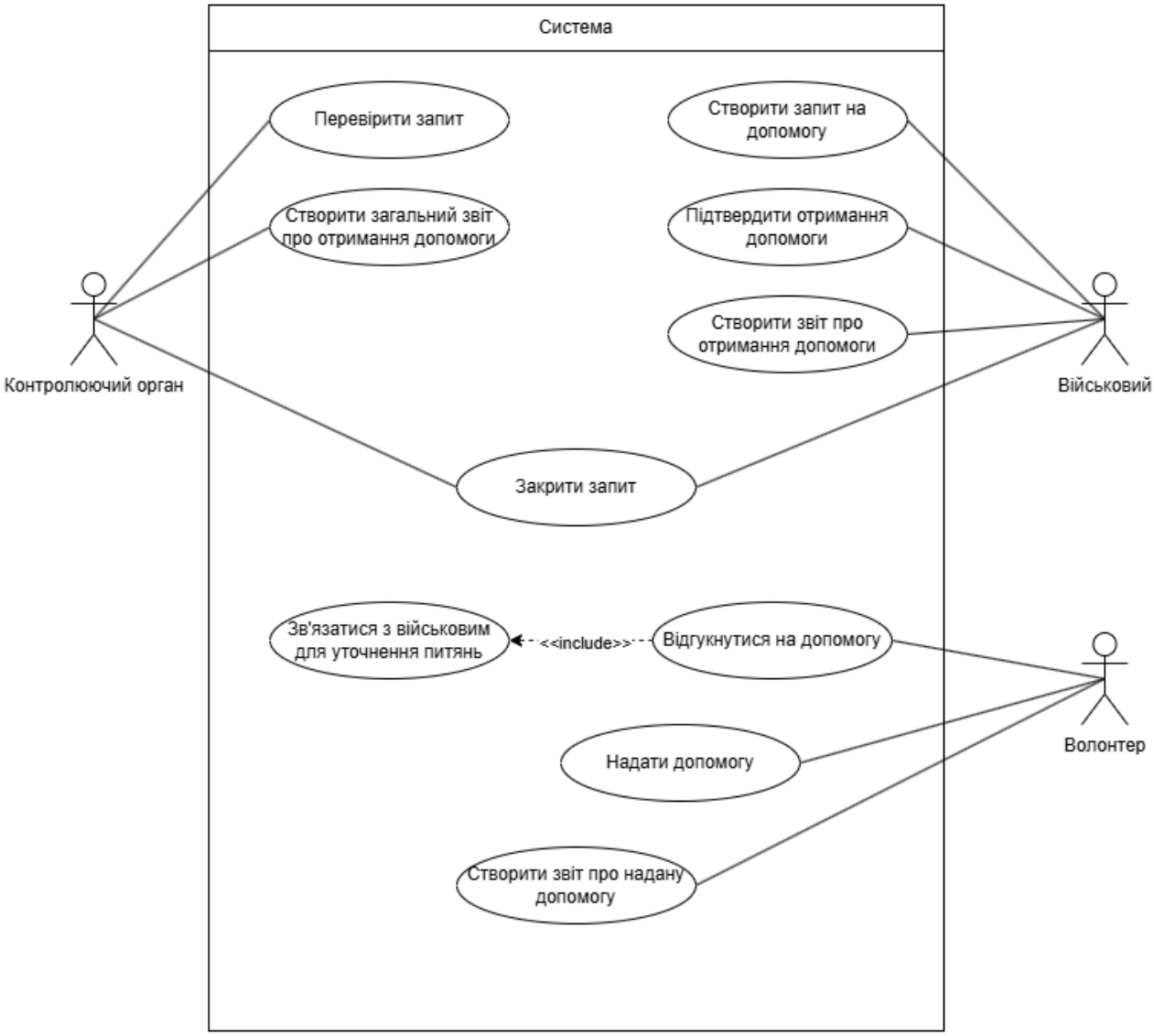



*Рис. 1.3.1 Діаграма прецедентів*

Для опису статичної структури системи застосовується діаграма класів. У межах цієї моделі доцільно виділити такі основні сутності: користувач, роль, допис, відгук, звіт, тег і зображення. Користувач пов’язується з роллю, яка визначає доступні йому дії в системі. Допис використовується для представлення запитів на допомогу та зборів коштів, а відгуки і звіти фіксують результати взаємодії між сторонами. Така діаграма дає змогу відобразити основні зв’язки між сутностями, що надалі реалізуються у структурі бази даних і серверній логіці.

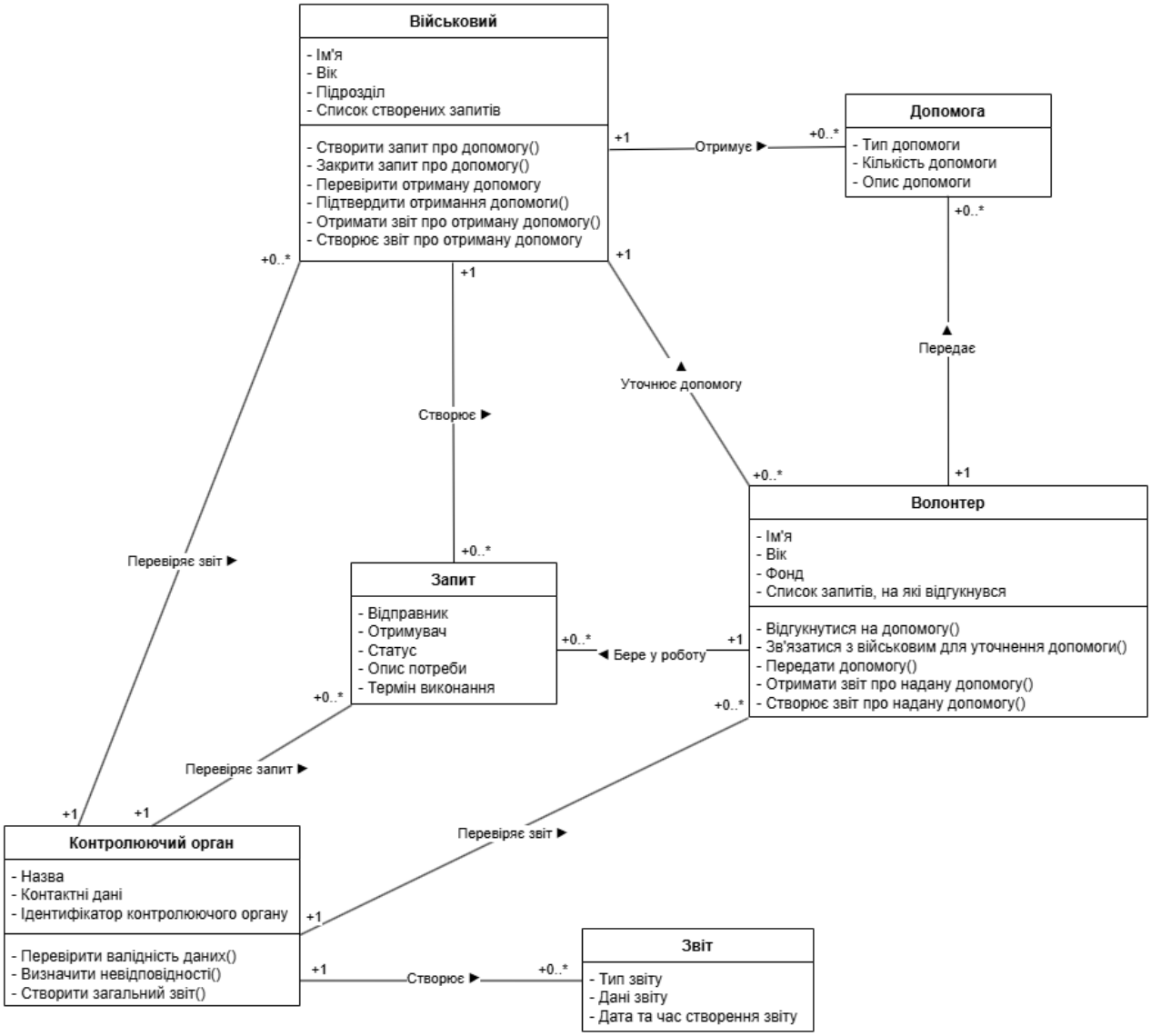



*Рис. 1.3.2 Діаграма класів*

Динамічні аспекти роботи серверної частини описуються за допомогою діаграм послідовності. Вони відображають порядок взаємодії між клієнтською частиною, маршрутами, контролерами, механізмами сесійної автентифікації та базою даних. Наприклад, у сценарії створення відгуку волонтером система послідовно виконує перевірку автентифікації, ролі користувача, стану запиту та відсутності дубльованого відгуку, після чого зберігає дані у базі та повертає клієнту результат виконання операції.

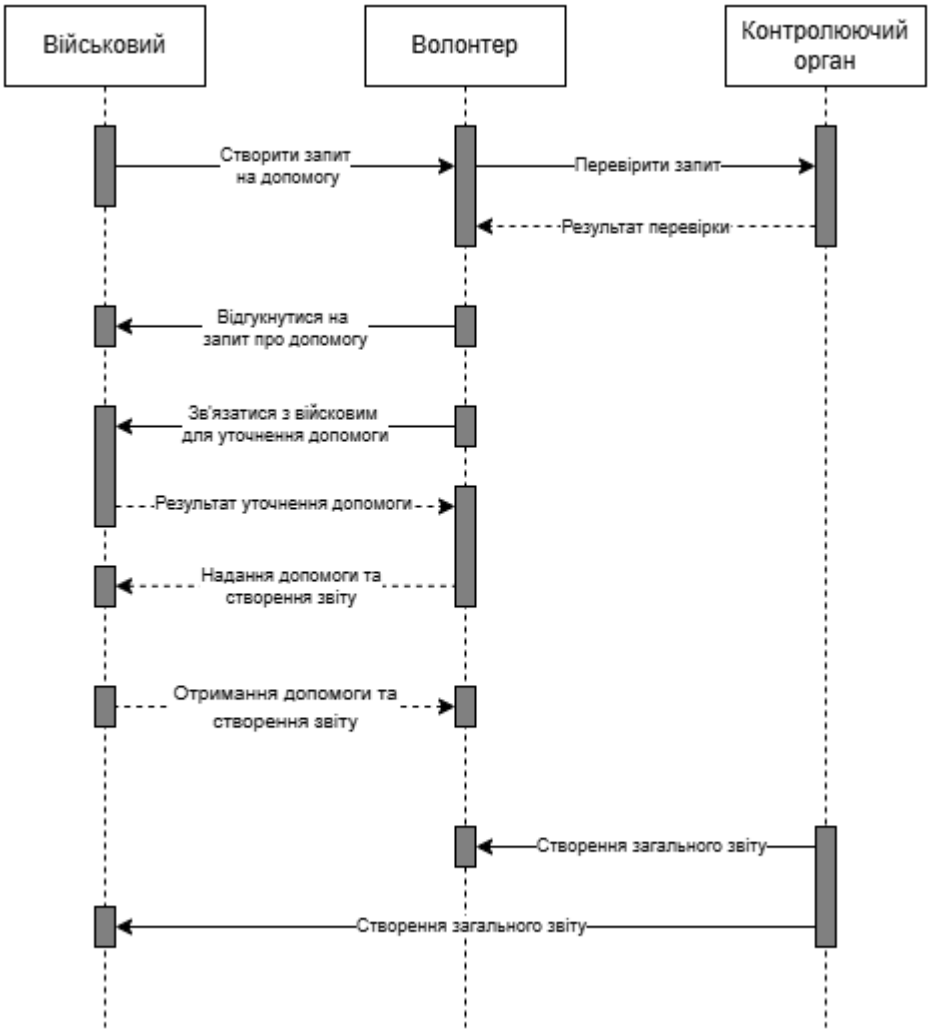



*Рис. 1.3.3 Діаграма послідовності*

Отже, UML-моделювання дозволяє описати серверну підсистему з функціональної, структурної та динамічної точок зору. Діаграма прецедентів визначає ролі користувачів і доступні їм функції, діаграма класів формалізує основні сутності предметної області, діаграма послідовності відображає виконання ключових сценаріїв.

## 1.4 Постановка задачі

На основі проведеного дослідження предметної області та аналізу наявних технологічних рішень було сформульовано завдання на розробку серверної підсистеми вебсистеми «Допомога Зараз». Основною метою розробки є створення серверної частини системи, яка забезпечує підтримку взаємодії між

військовими та волонтерами в процесі формування, опрацювання та документування запитів на допомогу.

У межах поставленої задачі серверна підсистема повинна забезпечувати реєстрацію та автентифікацію користувачів, розмежування доступу відповідно до їх ролей, обробку запитів на допомогу, відгуків і звітів, а також централізоване збереження даних, необхідних для функціонування системи. Окрему увагу приділено реалізації логіки, за якої військовий користувач може створювати запити на допомогу, а волонтер — переглядати їх, відгукуватися на них і фіксувати результати виконання.

Поставлена задача також передбачає реалізацію механізму збереження та обробки текстового і медіаконтенту, пов’язаного з дописами, відгуками та звітами. Серверна частина має забезпечувати узгоджене проходження основних етапів життєвого циклу запиту, від моменту його створення до формування звітної інформації про результат взаємодії між сторонами.

Таким чином, у межах бакалаврської роботи ставиться задача розробки серверної підсистеми, яка забезпечує функціональну основу для координації взаємодії між військовими та волонтерами, централізованого обліку даних і формування звітності щодо надання допомоги.

# 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА БАЗИ ДАНИХ ПІДСИСТЕМИ

## 2.1 Проєктування концептуальної та логічної схеми бази даних

База даних є центральним елементом серверної підсистеми вебсистеми «Допомога Зараз», оскільки саме вона забезпечує структуроване збереження інформації про користувачів, їх ролі, дописи, відгуки, звіти, теги та пов’язані з ними графічні матеріали. Від якості її проєктування безпосередньо залежить можливість реалізації серверної бізнес-логіки, підтримки рольової моделі доступу, відстеження станів запитів на допомогу та формування узгодженої звітної інформації. З огляду на це проєктування бази даних у межах роботи розглядається як одна з ключових складових усієї серверної підсистеми.

На концептуальному рівні база даних відображає основні сутності предметної області, що виникають у процесі взаємодії між військовими та волонтерами. До таких сутностей належать користувач, роль, допис, тип допису, тег, відгук, звіт та зображення. Користувач виступає базовим суб’єктом системи, оскільки саме з ним пов’язуються майже всі інші об’єкти. Роль визначає допустимі дії користувача. Допис відображає або запит на допомогу, або збір коштів. Відгук фіксує реакцію волонтера на запит військового. Звіт документує результат взаємодії сторін. Теги використовуються для тематичної класифікації користувачів і дописів, а зображення забезпечують можливість додавання візуального супроводу до основних інформаційних об’єктів.

На логічному рівні зазначені сутності реалізовано у вигляді дванадцяти взаємопов’язаних таблиць. Їх структура побудована з урахуванням принципів нормалізації, що дає змогу зменшити дублювання даних, чітко розмежувати довідникову, основну та допоміжну інформацію, а також підтримувати цілісність зв’язків між записами. Для зручності аналізу таблиці доцільно розглядати за функціональними групами: таблиці користувачів і ролей, таблиці

класифікації та тегів, таблиці основного контенту, таблиці взаємодії та таблиці звітності. Логічну модель бази даних наведено на рис. 2.1.1.

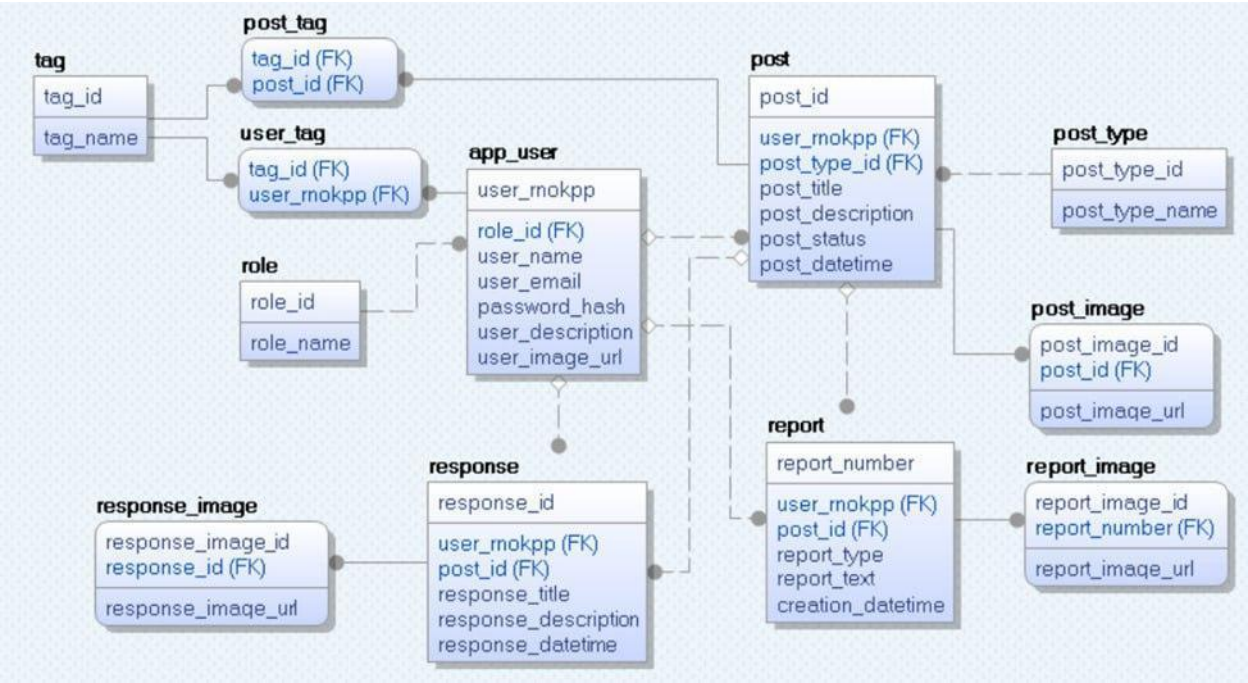



*Рис. 2.1.1 Логічна модель бази даних серверної підсистеми вебсистеми «Допомога Зараз»*

Як показано на рис. 2.1.1, логічна схема бази даних охоплює основні сутності предметної області та забезпечує їх взаємозв’язок на рівні реляційної моделі. Це створює структуровану основу для реалізації бізнес-логіки серверної підсистеми.

Першу групу утворюють таблиці role та app_user. Таблиця role виконує функцію довідника ролей і містить обмежений набір значень, що визначають тип користувача в системі. У поточній реалізації це роль військовослужбовця та роль волонтера. Винесення ролей в окрему таблицю дозволяє централізовано керувати допустимими типами користувачів і використовувати зовнішній ключ у таблиці користувачів замість дублювання текстових значень.

Таблиця app_user є основною таблицею користувачів. Вона містить ідентифікаційні та профільні атрибути, зокрема реєстраційний номер облікової картки платника податків, ім’я користувача, адресу електронної пошти, хеш пароля, опис профілю, посилання на зображення аватара та зовнішній ключ на роль. Обрана структура забезпечує однозначну ідентифікацію користувача в

межах системи та створює основу для подальшого зв’язування його з дописами, відгуками та звітами.

Другу групу становлять таблиці класифікації та тегів: tag, user_tag, post_type і post_tag. Таблиця tag містить унікальні назви тегів і використовується як централізований довідник тематичних категорій. Таблиця user_tag реалізує зв’язок типу «багато-до-багатьох» між користувачами та тегами. Аналогічно таблиця post_tag реалізує зв’язок «багато-до-багатьох» між дописами та тегами, що забезпечує можливість гнучкого тематичного маркування публікацій.

Таблиця post_type виконує роль довідника типів дописів. Її використання є важливим архітектурним рішенням, оскільки дозволяє не створювати окремі таблиці для запитів на допомогу та зборів коштів, а представляти їх у межах єдиної сутності post із фіксацією типу через зовнішній ключ. Такий підхід зменшує дублювання атрибутів, оскільки обидва види публікацій мають спільні поля: автора, заголовок, опис, статус, дату створення та пов’язані теги або зображення.

До третьої групи належать таблиці основного контенту: post і post_image. Таблиця post є центральною таблицею системи, оскільки саме в ній зберігається інформація про публікації користувачів. Вона містить посилання на автора, тип допису, заголовок, опис, логічний статус і часову мітку створення. У цій таблиці зберігаються як запити на допомогу, так і збори коштів. Наявність поля статусу дає змогу серверній частині відстежувати життєвий цикл допису.

Таблиця post_image винесена окремо від таблиці post, оскільки один допис може містити кілька зображень. Таке рішення дозволяє реалізувати зв’язок «один-до-багатьох» між дописом і його графічними матеріалами, не перевантажуючи основну таблицю повторюваними полями.

Четверту групу формують таблиці взаємодії: response і response_image. Таблиця response використовується для фіксації відгуків волонтерів на запити військових. Вона містить посилання на користувача, який залишив відгук, посилання на відповідний допис, текстові поля, статус та часову мітку створення.

На логічному рівні ця таблиця відображає факт прийняття запиту в роботу певним волонтером.

Таблиця response_image є допоміжною й використовується для збереження зображень, доданих до відгуку. Її наявність обґрунтовується тим, що один відгук може містити кілька графічних матеріалів, тому зберігання таких даних в окремій таблиці підтримує структурованість схеми.

П’яту групу утворюють таблиці звітності: report і report_image. Таблиця report призначена для збереження інформації про результат взаємодії між військовим і волонтером. У ній зберігаються посилання на допис, а в певних сценаріях також і на відгук, тип звіту та текстовий опис. Така структура дозволяє підтримувати як звіти автора запиту про отримання допомоги, так і звіти волонтера про її надання.

Таблиця report_image призначена для збереження зображень, пов’язаних зі звітами. Вона дозволяє додавати до звітного запису кілька візуальних підтверджень, наприклад фотографії переданої допомоги або інші супровідні матеріали.

Важливим аспектом логічної схеми є характер зв’язків між таблицями. Між role і app_user реалізовано зв’язок типу «один-до-багатьох», оскільки одна роль може бути призначена багатьом користувачам. Між app_user і post також існує зв’язок «один-до-багатьох», оскільки один користувач може створити багато дописів. Аналогічно один користувач може залишити багато відгуків, а один допис може мати багато відгуків, що формує зв’язки між app_user, response та post. Зв’язки між користувачами і тегами, а також між дописами і тегами реалізовано через проміжні таблиці user_tag і post_tag, що відповідає відношенням «багато-до-багатьох». Для зображень у таблицях post_image, response_image та report_image застосовано зв’язки «один-до-багатьох» із відповідними основними таблицями.

Окремого обґрунтування потребує використання єдиної таблиці post для зберігання як запитів на допомогу, так і зборів коштів. Таке рішення дозволяє підтримувати спільну структуру публікації та не дублювати однакові атрибути в

кількох таблицях. Розмежування між видами дописів здійснюється за допомогою таблиці post_type, тоді як обмеження щодо допустимих дій користувачів і сценаріїв роботи з конкретним типом допису контролюються серверною бізнес-логікою.

З погляду нормалізації спроєктована схема бази даних дозволяє розділити довідникові сутності, основні операційні об’єкти та допоміжні зв’язувальні таблиці. Винесення ролей, типів дописів і тегів у довідникові таблиці зменшує дублювання текстових значень. Реалізація відношень «багато-до-багатьох» через user_tag і post_tag усуває потребу зберігати множинні значення в одному полі. Винесення зображень в окремі таблиці спрощує структуру основних записів і дозволяє масштабувати кількість пов’язаних файлів без зміни логічної моделі базових сутностей.

Отже, спроєктована концептуальна та логічна схема бази даних охоплює всі основні сутності предметної області та забезпечує структуровану основу для функціонування серверної підсистеми вебсистеми «Допомога Зараз». Вона підтримує рольову модель користувачів, зберігання дописів, відгуків і звітів, реалізацію тематичної класифікації, роботу з кількома зображеннями та фіксацію взаємозв’язків між усіма ключовими об’єктами системи.

## 2.2 Вибір СУБД

Після формування концептуальної та логічної схеми бази даних наступним етапом стало обґрунтування вибору системи керування базами даних, у середовищі якої буде реалізовано спроєктовану структуру. Оскільки серверна підсистема вебсистеми «Допомога Зараз» оперує взаємопов’язаними сутностями, рольовою моделлю доступу, статусами об’єктів і звітною інформацією, до СУБД висувалися вимоги щодо підтримки реляційної моделі, цілісності зв’язків, транзакційності, надійності збереження даних і зручності інтеграції з серверною частиною.

У межах предметної області система працює з даними, які мають чітко визначені структурні зв’язки. Користувачі пов’язуються з ролями, дописи — з

авторами, тегами та зображеннями, відгуки — із запитами та користувачами, а звіти — з дописами та, за потреби, з відгуками. Така структура природно відповідає реляційному підходу до організації даних. Саме тому як основний клас рішень доцільно було розглядати реляційні системи керування базами даних.

Альтернативою реляційній СУБД могли бути файлові сховища або документо-орієнтовані бази даних. Проте їх використання в межах цього проєкту було б менш доцільним. Файловий підхід не забезпечує належного рівня контролю зв’язків між сутностями та ускладнює реалізацію одночасного доступу до даних. Документо-орієнтовані СУБД є ефективними для слабоструктурованих або вкладених даних, однак у даній системі ключове значення має саме підтримка чітких реляційних зв’язків, перевірка зовнішніх ключів та узгоджене виконання операцій над кількома пов’язаними об’єктами.

З урахуванням зазначених вимог як СУБД було обрано PostgreSQL. Такий вибір зумовлений тим, що PostgreSQL є зрілою реляційною системою керування базами даних, яка підтримує роботу зі складними схемами, зовнішніми ключами, транзакціями та механізмами забезпечення цілісності даних. Для розроблюваної серверної підсистеми це має принципове значення, оскільки база даних повинна не лише зберігати інформацію, а й підтримувати коректність зв’язків між користувачами, дописами, відгуками, звітами та супровідними сутностями.

Важливою перевагою PostgreSQL є підтримка транзакційного підходу до виконання операцій. У межах серверної підсистеми окремі бізнес-сценарії передбачають послідовне створення або оновлення кількох взаємопов’язаних записів. Наприклад, під час створення допису може одночасно виконуватися додавання запису до таблиці дописів, прив’язка тегів і збереження кількох зображень. Аналогічно під час створення відгуку або звіту можуть змінюватися стани пов’язаних об’єктів. У таких ситуаціях підтримка транзакцій дозволяє гарантувати, що зміни будуть або виконані повністю, або скасовані у разі помилки, що є важливим для збереження узгодженості даних.

Ще однією вагомою причиною вибору PostgreSQL є підтримка механізмів обмежень цілісності. Система дозволяє задавати первинні та зовнішні ключі, унікальні обмеження та перевірки значень, що допомагає частково переносити контроль коректності даних на рівень самої СУБД. Для розроблюваного проєкту це особливо важливо, оскільки таблиці бази даних мають велику кількість взаємозв’язків, а порушення цих зв’язків могло б призвести до втрати узгодженості між дописами, відгуками та звітами.

З практичного погляду PostgreSQL також є зручним вибором для інтеграції з обраним серверним стеком. Серверна частина реалізована на платформі Node.js, а взаємодія з базою даних виконується через бібліотеку node-postgres (pg), яка забезпечує пряме виконання SQL-запитів і підтримку пулу з’єднань. Такий підхід добре узгоджується з архітектурою поточного проєкту, де значна частина бізнес-логіки реалізується на рівні серверних контролерів, а доступ до даних виконується через параметризовані SQL-запити.

На етапі розробки та тестування основним робочим середовищем є локально розгорнутий екземпляр PostgreSQL, що забезпечує стабільність і незалежність від зовнішніх інфраструктурних обмежень. Водночас обрана СУБД допускає подальше хмарне розміщення. Як один із варіантів такого розміщення може використовуватися сервіс Neon, який надає можливість розгортання PostgreSQL у зовнішньому середовищі та підтримує підключення до серверної частини через стандартний рядок з’єднання. Таким чином, використання PostgreSQL є доцільним як для локального етапу розробки, так і для подальшого перенесення бази даних у хмарну інфраструктуру.

Таким чином, вибір PostgreSQL як системи керування базами даних є обґрунтованим як з позицій логіки предметної області, так і з позицій архітектури серверної підсистеми. Використання саме реляційної СУБД дозволяє підтримувати цілісність зв’язків між основними сутностями, забезпечувати транзакційну узгодженість операцій та створювати надійну основу для реалізації бізнес-логіки вебсистеми «Допомога Зараз».

## 2.3 Формування бази даних

Після проєктування концептуальної та логічної схеми наступним етапом стало формування фізичної моделі бази даних. На цьому етапі виділені сутності предметної області були реалізовані у вигляді конкретних таблиць, полів, первинних і зовнішніх ключів засобами мови SQL у середовищі PostgreSQL. Основною метою цього етапу було перенесення логічної структури даних у фізичну форму, придатну до безпосереднього використання серверною підсистемою вебсистеми «Допомога Зараз».

Формування бази даних здійснювалося шляхом створення таблиць, які відповідають основним сутностям предметної області, а також допоміжних зв'язувальних таблиць для реалізації складніших типів зв'язків. На рівні фізичної моделі були визначені типи полів, первинні ключі, зовнішні ключі та базові обмеження цілісності. Такий підхід дозволив забезпечити структурованість даних і створити основу для коректної роботи серверної бізнес-логіки.

Одними з перших було сформовано довідникові таблиці role і post_type, які використовуються для збереження сталих категорій даних. Таблиця role містить ролі користувачів, а таблиця post_type — типи дописів. Винесення цих сутностей в окремі таблиці дозволило уникнути дублювання текстових значень в основних таблицях і забезпечило використання зовнішніх ключів для контролю коректності зв'язків. Приклад створення довідникової таблиці ролей наведено на рис. 2.3.1.

```
CREATE TABLE "role" (
    "role_id" integer PRIMARY KEY,
"role_name" varchar
);
```

*Рис. 2.3.1 Фрагмент SQL-коду створення таблиці role*

Наведений фрагмент SQL-коду демонструє створення довідникової таблиці ролей, яка використовується для реалізації рольової моделі доступу в серверній підсистемі.

Після цього було створено основну таблицю користувачів app_user. Вона містить ідентифікаційні, облікові та профільні атрибути, зокрема реєстраційний номер облікової картки платника податків, ім’я користувача, адресу електронної пошти, хеш пароля, опис профілю, посилання на зображення аватара та зовнішній ключ на роль. Обрана структура забезпечує однозначну ідентифікацію користувача в межах системи та створює основу для подальшого зв’язування його з дописами, відгуками та звітами. Фізичну реалізацію таблиці користувачів наведено на рис. 2.3.2.

```
CREATE TABLE "app_user" (
"user_rnokpp" varchar PRIMARY KEY,
"role_id" integer,
"user_name" varchar,
"user_email" varchar,
"password_hash" varchar,
"user_description" text,
"user_image_url" varchar
);

ALTER TABLE "app_user"
    ADD FOREIGN KEY ("role_id") REFERENCES "role" ("role_id");
```

*Рис. 2.3.2 Фрагмент SQL-коду створення таблиці користувачі*

У цьому фрагменті SQL-коду показано фізичну реалізацію основної таблиці користувачів та зв’язку з таблицею ролей через зовнішній ключ.

Для реалізації тематичної класифікації було створено таблицю tag, а також проміжні таблиці user_tag і post_tag. Таблиця tag містить унікальні значення тегів, тоді як user_tag і post_tag забезпечують зв’язки типу «багато-до-багатьох» між тегами та відповідно користувачами або дописами. Фізична реалізація цих зв’язків через окремі таблиці дозволяє уникати зберігання множинних значень у межах одного поля та підтримує гнучке масштабування класифікаційної підсистеми.

Центральне місце у структурі бази даних займає таблиця post, яка використовується для збереження інформації про всі дописи користувачів. На

фізичному рівні вона містить первинний ключ, зовнішні ключі на користувача та тип допису, а також основні атрибути публікації: заголовок, опис, статус і часову мітку створення. Окремою особливістю проєктного рішення є використання єдиної таблиці post для представлення як запитів на допомогу, так і зборів коштів. Це дозволило уникнути дублювання однакових структурних атрибутів у кількох таблицях і зберегти цілісність логічної моделі.

Для підтримки тематичного маркування дописів було сформовано проміжну таблицю post_tag, яка реалізує зв’язок типу «багато-до-багатьох» між публікаціями та тегами. Приклад фізичної реалізації таблиці дописів і проміжної таблиці тегів наведено на рис. 2.3.3.

```
CREATE TABLE "post" (
"post_id" integer PRIMARY KEY,
"user_rnokpp" varchar,
"post_type_id" integer,
"post_title" varchar,
"post_description" text,
"post_status" varchar,
"post_datetime" timestamp
);

CREATE TABLE "post_tag" (
"tag_id" integer,
"post_id" integer
);

ALTER TABLE "post"
ADD FOREIGN KEY ("user_rnokpp") REFERENCES "app_user" ("user_rnokpp");

ALTER TABLE "post"
ADD FOREIGN KEY ("post_type_id") REFERENCES "post_type" ("post_type_id");

ALTER TABLE "post_tag"
ADD FOREIGN KEY ("tag_id") REFERENCES "tag" ("tag_id");

ALTER TABLE "post_tag"
ADD FOREIGN KEY ("post_id") REFERENCES "post" ("post_id");
```

*Рис. 2.3.3 Фрагмент SQL-коду створення таблиці дописів і реалізація зв’язку з тегами*

Повний скрипт створення таблиць і зв’язків між ними подано в додатку А.

# 3 РОЗРОБКА АРХІТЕКТУРИ ТА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СЕРВЕРНОЇ ПІДСИСТЕМИ

## 3.1 Обґрунтування вибору технологічного стеку підсистеми бекенду

Для реалізації серверної підсистеми вебсистеми «Допомога Зараз» було обрано технологічний стек, який забезпечує обробку HTTP-запитів, взаємодію з реляційною базою даних, реалізацію бізнес-логіки та можливість подальшого хмарного розгортання. Вибір здійснювався з урахуванням вимог до продуктивності, гнучкості розробки та придатності до побудови REST-орієнтованої архітектури [1].

Як середовище виконання серверної частини використано Node.js [5]. Його застосування зумовлене підтримкою асинхронної моделі введення-виведення, що є доцільним для вебзастосунків, орієнтованих на обробку великої кількості мережевих запитів. Додатковою перевагою є широка екосистема пакетів і зручність інтеграції допоміжних бібліотек.

Для побудови серверного інтерфейсу було використано Express.js [6-8]. Цей фреймворк надає зручні засоби маршрутизації, підтримує проміжне програмне забезпечення та дозволяє реалізувати REST API, перевірку вхідних даних, контроль доступу й централізовану обробку помилок. Обмін даними між клієнтом і сервером організовано у форматі JSON [4].

Як систему керування базами даних обрано PostgreSQL [9], оскільки предметна область передбачає роботу зі структурованими взаємопов’язаними даними: користувачами, ролями, дописами, відгуками, звітами, зображеннями та тегами. PostgreSQL забезпечує підтримку реляційної моделі, цілісності зв’язків і транзакційних операцій, що є важливим для узгодженого збереження станів об’єктів системи. Для доступу до бази даних використано бібліотеку node-postgres (pg) [10, 11], яка підтримує SQL-запити та пул з’єднань.

Для реалізації автентифікації та керування доступом використано сесійний підхід [13]. Захист облікових даних забезпечується за допомогою бібліотеки bcrypt [12], яка використовується для хешування паролів перед їх збереженням у базі даних. Це дозволяє підвищити безпеку збереження облікової інформації та підтримати рольове розмежування доступу в системі.

Як перспективне середовище розгортання розглядалися Render для серверної частини [14] та Neon як один із варіантів хмарного розміщення PostgreSQL [15, 16]. Водночас на поточному етапі основним робочим середовищем залишається локальна конфігурація, що забезпечує стабільність під час тестування та доопрацювання функціональності.

Отже, використання Node.js, Express.js, PostgreSQL та node-postgres є обґрунтованим для реалізації серверної підсистеми вебсистеми, оскільки цей стек забезпечує побудову REST API, реалізацію бізнес-логіки, безпечне збереження даних і підтримку автентифікації користувачів.

## 3.2 Архітектура серверної частини програмної системи

Архітектура серверної частини вебсистеми «Допомога Зараз» побудована з урахуванням принципу розділення відповідальності між основними програмними модулями. Такий підхід дає змогу відокремити обробку HTTP-запитів, прикладну бізнес-логіку, роботу із сесіями користувачів, доступ до бази даних і допоміжні службові операції. У результаті серверна підсистема набуває більш чіткої внутрішньої організації, що спрощує її підтримку, тестування та подальше розширення.

У структурі бекенду можна виділити кілька основних рівнів. Перший рівень становлять маршрути, які визначають набір доступних API-ендпоінтів і виконують первинне спрямування HTTP-запитів [3] до відповідних обробників. Саме на цьому рівні задається структура прикладного програмного інтерфейсу, через який клієнтська частина взаємодіє із сервером. Маршрути не містять складної прикладної логіки, а виконують функцію зв’язувальної ланки між зовнішнім запитом і внутрішніми програмними механізмами системи.

Другий рівень утворюють контролери, у яких зосереджено основну прикладну логіку серверної підсистеми. Контролери приймають дані від маршрутів, здійснюють перевірку вхідних параметрів, викликають операції читання і запису в базі даних, формують відповіді клієнту та реалізують перевірки, що випливають із бізнес-правил системи. Саме на рівні контролерів виконується значна частина логіки, пов’язаної з реєстрацією користувачів, створенням дописів, відгуків і звітів, контролем статусів об’єктів і рольовими обмеженнями.

Окремим компонентом архітектури є модуль керування сесіями, який забезпечує підтримку авторизованого стану користувача. Його призначення полягає у створенні, перевірці та обробці сесійних даних, що дозволяє серверній частині ідентифікувати користувача під час виконання захищених запитів. Саме через цей модуль реалізується зв’язок між механізмами автентифікації та рольовою моделлю доступу, оскільки під час обробки запиту сервер повинен не лише визначити, хто є автором дії, а й чи має цей користувач право на виконання відповідної операції.

Ще одним важливим рівнем є модуль доступу до бази даних. У межах поточної архітектури він забезпечує встановлення з’єднання з PostgreSQL і використовується контролерами для виконання SQL-запитів. Через цей модуль серверна частина взаємодіє з таблицями користувачів, ролей, дописів, відгуків, звітів, тегів і зображень. Таким чином, модуль доступу до бази даних виступає інфраструктурною основою для реалізації всієї прикладної логіки системи.

Окрему роль у структурі серверної частини відіграють допоміжні модулі. До них належать засоби перевірки зображень, модуль надсилання повідомлень електронною поштою та інші утилітарні компоненти, які не формують основної бізнес-логіки, але забезпечують коректну роботу прикладних сценаріїв. Винесення таких функцій в окремі модулі дозволяє зменшити перевантаження контролерів і підвищити читабельність програмного коду.

З погляду практичної реалізації архітектура серверної частини включає такі основні структурні елементи: пакет маршрутів, пакет контролерів, модуль

роботи з базою даних, модуль сесійної автентифікації, а також допоміжні сервіси й утиліти. Логічний розподіл цих компонентів доцільно подати окремою схемою, наведеною на рис. 3.2.1.

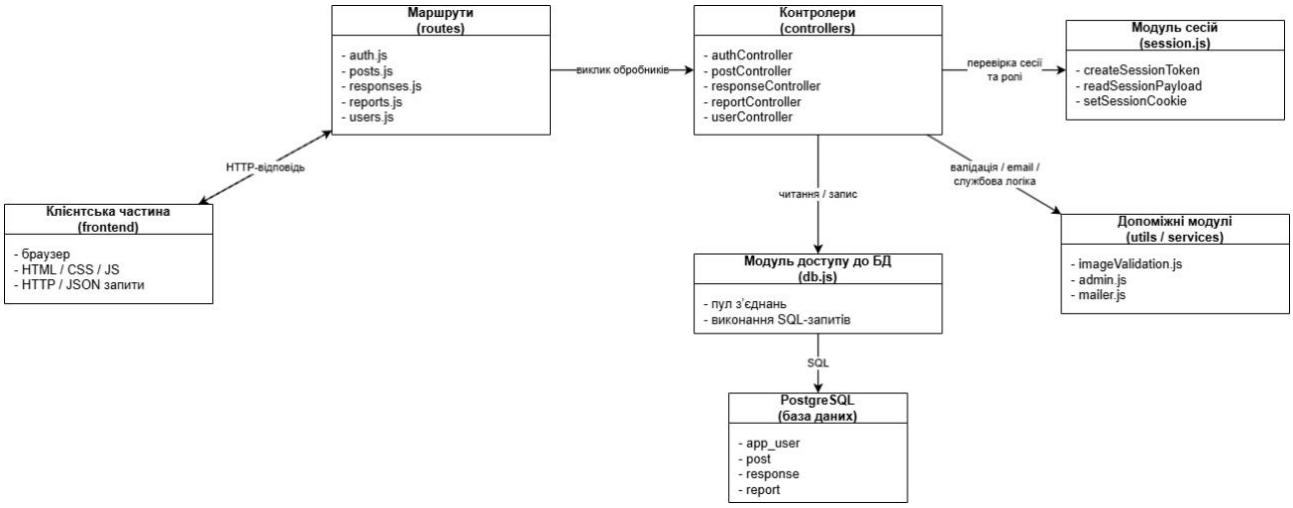



*Рис. 3.2.1 Архітектура серверної частини вебсистеми «Допомога Зараз»*

Як показано на рис. 3.2.1, взаємодія між компонентами організована послідовно. Клієнтський запит надходить до маршруту, далі передається контролеру, який за потреби звертається до механізмів сесійної перевірки, допоміжних модулів і бази даних. Після виконання необхідних операцій результат повертається клієнтській частині у вигляді HTTP-відповіді. Така архітектурна схема забезпечує чітке відокремлення прикладного рівня від інфраструктурного та зменшує залежність між окремими частинами системи.

Перевагою обраної архітектури є її простота й водночас достатня гнучкість для реалізації основних функцій вебсистеми. Маршрути можуть змінюватися без суттєвого впливу на логіку контролерів, а допоміжні модулі можуть розширюватися без зміни загальної структури серверної частини. Крім того, такий підхід дозволяє поетапно вдосконалювати систему, додаючи нові механізми без повного перегляду вже реалізованої архітектури.

Отже, архітектура серверної частини вебсистеми «Допомога Зараз» побудована на модульному принципі з чітким розподілом відповідальності між маршрутами, контролерами, механізмами сесійної автентифікації, модулем доступу до бази даних і допоміжними сервісами. Така організація забезпечує

структурованість програмного коду, підтримує реалізацію бізнес-логіки системи та створює основу для подальшого розвитку серверної підсистеми.

## 3.3 Організація REST API серверної підсистеми

Важливою складовою серверної частини вебсистеми «Допомога Зараз» є організація прикладного програмного інтерфейсу, через який клієнтська частина взаємодіє з бекендом. У межах поточної реалізації для цього використано підхід REST API [1], який забезпечує стандартизований обмін даними між клієнтом і сервером за допомогою HTTP-запитів [3]. Такий підхід є доцільним для вебзастосунків, оскільки дозволяє чітко розмежувати функції клієнтської та серверної частин, спростити обробку запитів і забезпечити передбачувану структуру взаємодії між компонентами системи.

У серверній підсистемі REST API організовано за ресурсним принципом. Це означає, що окремі групи маршрутів відповідають за роботу з конкретними сутностями предметної області, такими як користувачі, дописи, відгуки та звіти. Кожен маршрут призначений для виконання певної операції над відповідним ресурсом, а саме створення, перегляду, оновлення або видалення даних. Така структура інтерфейсу забезпечує логічну впорядкованість програмного доступу до функцій системи та полегшує підтримку серверного коду.

У поточній реалізації серверна частина містить окремі групи маршрутів для автентифікації, роботи з дописами, відгуками, звітами та профілями користувачів. Узагальнену структуру основних ендпоінтів наведено в табл. 3.3.1.

*Таблиця 3.3.1*

**Основні ендпоінти REST API серверної підсистеми**

| **Група маршрутів** | **HTTP-метод** | **Шлях** | **Призначення** |
|---|---|---|---|
| Автентифікація | POST | /auth/register | Реєстрація нового користувача |

*Продовження Таблиці 3.3.1*

| **Група маршрутів** | **HTTP-метод** | **Шлях** | **Призначення** |
|---|---|---|---|
| Автентифікація | POST | /auth/login | Вхід користувача в систему |
| Автентифікація | POST | /auth/logout | Завершення сесії користувача |
| Автентифікація | GET | /auth/session | |
| Автентифікація | GET | /auth/profile | Отримання даних про поточну сесію |
| Автентифікація | PATCH | /auth/profile | Оновлення профілю користувача |
| Дописи | GET | /posts | Отримання списку дописів |
| Дописи | POST | /posts | Створення нового допису |
| Дописи | PATCH | /posts/:postId | Редагування допису |
| Дописи | DELETE | /posts/:postId | Видалення допису |
| Відгуки | GET | /responses/mine | Отримання прийнятих запитів волонтера |
| Відгуки | POST | /responses | Створення нового відгуку |

*Продовження Таблиці 3.3.1*

| **Група маршрутів** | **HTTP-метод** | **Шлях** | **Призначення** |
|---|---|---|---|
| Відгуки | DELETE | /responses/:responseId | Видалення відгуку |
| Звіти | POST | /reports | Створення звіту |
| Звіти | GET | /reports | Отримання звітів |
| Користувачі | GET | /users/:userRnokpp | Отримання даних конкретного користувача |

Окрему групу маршрутів становлять маршрути автентифікації, які забезпечують реєстрацію користувачів, вхід до системи, завершення сесії та перевірку поточного авторизованого стану. Приклад організації маршрутів автентифікації наведено на рис. 3.3.1.

```
const express = require('express');
const router = express.Router();

const {
  getSessionUser,
  getUserProfile,
  loginUser,
  logoutUser,
  registerUser,
  updateUserProfile,
} = require('../controllers/authController');

router.post('/register', registerUser);
router.post('/login', loginUser);
router.post('/logout', logoutUser);
router.get('/session', getSessionUser);
router.get('/profile', getUserProfile);
router.patch('/profile', updateUserProfile);

module.exports = router;
```

*Рис. 3.3.1 Приклад коду реалізації маршрутів автентифікації*

Саме через ці ендпоінти клієнтська частина передає реєстраційні або облікові дані, а сервер виконує перевірку коректності введеної інформації, хешування пароля, створення користувача або перевірку облікових записів.

Наведений приклад коду на рис. 3.3.1 демонструє, що маршрути згруповано за функціональним призначенням, а кожен ендпоінт пов’язаний із конкретним обробником контролера. Такий підхід спрощує підтримку API та забезпечує зрозумілу структуру серверної частини.

Наступну важливу групу утворюють маршрути роботи з дописами. Вони охоплюють створення, отримання, редагування та видалення публікацій, а також доступ до окремих вибірок, пов’язаних із типом або станом допису. Саме ці маршрути забезпечують підтримку основного контенту системи, зокрема запитів на допомогу та зборів коштів. У межах цих ендпоінтів реалізується також обробка тегів і супровідних зображень, що дозволяє серверу формувати повноцінний об’єкт публікації для подальшого використання клієнтською частиною.

Окремо організовано маршрути для відгуків, через які волонтери можуть створювати відгуки на відкриті запити, отримувати пов’язані з ними дані та передавати зображення, якщо це передбачено сценарієм роботи. У межах цієї частини API сервер виконує не лише операції запису або читання даних, а й перевіряє роль користувача, стан запиту та допустимість створення нового відгуку. Приклад організації маршрутів для роботи з відгуками наведено на рис. 3.3.2.

```
const express = require('express');
const router = express.Router();
const {
  createResponse,
  deleteResponse,
  fetchAcceptedRequests,
} = require('../controllers/responseController');
router.get('/mine', fetchAcceptedRequests);
router.post('/', createResponse);
router.delete('/:responseId', deleteResponse);
```

```
module.exports = router;
```

*Рис. 3.3.2 Приклад коду реалізації маршрутів для роботи з відгуками*

Ще одну групу становлять маршрути звітності, які забезпечують формування та отримання звітів про результати надання або отримання допомоги. Через ці ендпоінти користувачі передають текстову інформацію, а за потреби й супровідні зображення, що використовуються для фіксації завершення окремого етапу взаємодії. На рівні серверної логіки саме ці маршрути пов’язані з документуванням результату виконання запиту та зміною станів відповідних об’єктів.

Крім цього, у структурі API передбачено маршрути роботи з профілем користувача, які дозволяють отримувати й оновлювати профільні дані, зокрема текстовий опис, теги та зображення користувача. Такі маршрути доповнюють основний функціональний контур системи та забезпечують персоналізацію облікових записів.

Обмін даними між клієнтською і серверною частинами здійснюється у форматі JSON [4], що є стандартним для REST-орієнтованих вебзастосунків. Клієнт надсилає серверу структуровані дані в тілі HTTP-запиту, а сервер повертає відповіді у вигляді JSON-об’єктів або масивів. Приклад запиту на реєстрацію користувача наведено на рис. 3.3.3.

```
{
  "full_name": "Іван Петренко",
  "email": "ivan.petrenko@example.com",
  "password": "SecurePass123",
  "rnokpp": "1234567890",
  "role_id": "mi"
}
```

*Рис. 3.3.3 Приклад JSON-запиту на реєстрацію користувача*

Більш прикладним для предметної області є сценарій створення відгуку на запит. Саме цей сценарій відображає ключову функціональну особливість системи, оскільки забезпечує перехід від опублікованої потреби до реальної взаємодії між військовим і волонтером. На рівні контролера сервер послідовно

перевіряє коректність вхідних даних, авторизований стан користувача, його роль, існування цільового запиту, його тип і статус, а також відсутність дубльованого відгуку. Додатково контролюється, щоб користувач не міг відгукнутися на власний запит або виконати дію, яка суперечить встановленим бізнес-правилам. Такий підхід забезпечує цілісність прикладної логіки та коректність подальшого проходження запиту в межах системи. Фрагмент такої реалізації наведено на рис. 3.3.4.

```
async function createResponse(req, res) {
  const postId = Number(req.body.post_id);
  const title = String(req.body.title || '').trim();
  const description = String(req.body.description || '').trim();

  if (!Number.isInteger(postId)) {
    return res.status(400).json({ message: 'Некоректний запит.' });
  }

  const currentUser = await getCurrentUser(req);
  if (!currentUser) {
    return res.status(401).json({ message: 'Потрібно увійти в систему.' });
  }

  if (!isVolunteer(currentUser)) {
    return res.status(403).json({
      message: 'Відгукатися на запити можуть лише волонтери.'
    });
  }

}
```

*Рис. 3.3.4 Фрагмент реалізації створення відгуку*

Наведений приклад показує, що REST API у даній системі виконує не лише функцію транспортного рівня для передачі даних, а й виступає точкою інтеграції з прикладною бізнес-логікою. Саме через обробники маршрутів реалізуються валідація, контроль доступу, перевірка станів об’єктів і формування структурованих відповідей клієнтській частині.

Важливою властивістю організації REST API є використання стандартних HTTP-методів і статус-кодів [3]. Для створення нових записів використовуються запити типу POST, для отримання даних — GET, для оновлення — PATCH, а для видалення — DELETE. У відповідях сервер повертає статус-коди, які інформують клієнтську частину про результат виконання операції. Завдяки цьому фронтенд може коректно обробляти як успішні відповіді, так і помилки автентифікації, валідації або доступу.

Отже, REST API серверної підсистеми вебсистеми «Допомога Зараз» організовано за ресурсним принципом із виділенням окремих груп маршрутів для автентифікації, роботи з дописами, відгуками, звітами та профілями користувачів. Використання стандартизованих HTTP-методів, структурованих JSON-повідомлень і чітко розмежованих ендпоінтів забезпечує логічну впорядкованість прикладного інтерфейсу та створює надійну основу для взаємодії між клієнтською і серверною частинами системи.

## 3.4 Реалізація механізмів автентифікації та рольового доступу

Однією з ключових складових серверної підсистеми вебсистеми «Допомога Зараз» є реалізація механізмів автентифікації користувачів і рольового розмежування доступу до функцій системи. Саме ці механізми забезпечують ідентифікацію користувача, підтримку його авторизованого стану та перевірку допустимості виконання конкретних дій відповідно до обраної ролі. Для розроблюваної системи це має принципове значення, оскільки бізнес-логіка безпосередньо залежить від того, чи є користувач військовим або волонтером.

Механізм автентифікації починається з реєстрації користувача. На цьому етапі сервер приймає реєстраційні дані, перевіряє коректність обов’язкових полів, унікальність електронної пошти та РНОКПП, а також правильність значення ролі. Після проходження перевірок пароль не зберігається у відкритому вигляді, а обробляється за допомогою алгоритму хешування. У межах поточної реалізації для цього використовується бібліотека bcrypt [12], яка дозволяє

зберігати в базі даних лише результат криптографічного перетворення, а не вихідне значення пароля.

Після успішної реєстрації або входу до системи сервер формує сесію користувача. У поточній архітектурі це реалізовано на основі підписаного cookie-механізму [13]. Такий підхід дозволяє серверу зберігати мінімально необхідну інформацію про користувача в сесійному токені та надалі використовувати її для перевірки авторизованого стану. Під час надходження захищеного запиту серверна частина зчитує сесійні дані, визначає користувача та виконує подальші перевірки вже з урахуванням його ролі.

Рольова модель доступу ґрунтується на використанні довідникової таблиці ролей та її зв’язку з таблицею користувачів. У поточній реалізації система підтримує дві основні ролі: військовослужбовця та волонтера. Кожна з них відповідає певному набору допустимих дій у межах серверної підсистеми. Основні можливості ролей наведено в табл. 3.4.1.

*Таблиця 3.4.1*

**Рольова модель користувачів серверної підсистеми**

| **Роль** | **Основні дозволені дії** |
|---|---|
| Військовослужбовець | Реєстрація, вхід, створення запитів на допомогу, редагування власних дописів, перегляд відгуків, формування звітів про отримання допомоги |
| Волонтер | Реєстрація, вхід, перегляд відкритих запитів, створення відгуків, перегляд прийнятих запитів, створення зборів коштів, формування звітів про надання допомоги |

На рівні програмної реалізації роль користувача задається через конфігурацію допустимих значень, яка пов’язує короткий код ролі з її ідентифікатором та назвою. Фрагмент такої реалізації наведено на рис. 3.4.1.

```
const ROLE_CONFIG = {
  vo: { id: 1, code: 'vo', name: 'Волонтер' },
  mi: { id: 2, code: 'mi', name: 'Військовий' },
};
```

```
const ROLE_BY_ID = Object.fromEntries(
  Object.values(ROLE_CONFIG).map((role) => [role.id, role])
);
```

*Рис. 3.4.1 Приклад коду конфігурації ролей користувачів*

Наведений лістинг показує, що рольова модель має централізоване подання на рівні серверного коду. Це дозволяє використовувати однакові значення ролей як під час реєстрації, так і під час подальших перевірок доступу.

Окремо важливим етапом є валідація вхідних даних реєстрації. Сервер перевіряє, чи заповнено обов'язкові поля, чи має пароль достатню довжину, чи відповідає РНОКПП встановленому формату та чи належить роль до підтримуваних значень. Фрагмент відповідної реалізації наведено на рис. 3.4.2.

```
function validateRegistrationPayload({ full_name, email, password, rnokpp, role_id
}) {
  if (!full_name?.trim()) {
    return 'Вкажіть ім'я або назву профілю.';
  }

  if (!email?.trim()) {
    return 'Вкажіть електронну пошту.';
  }

  if (!password || password.length < 8) {
    return 'Пароль має містити щонайменше 8 символів.';
  }

  if (!/^\d{10}$/.test(String(rnokpp || '').trim())) {
    return 'РНОКПП має містити 10 цифр.';
  }

  if (!resolveRole(role_id)) {
    return 'Оберіть коректну роль.';
  }
}
```

*Рис. 3.4.2 Фрагмент перевірки даних під час реєстрації*

Цей фрагмент демонструє, що перевірка коректності даних виконується ще до створення користувача в базі даних. Такий підхід дозволяє зменшити кількість помилкових запитів і забезпечує базовий рівень консистентності вхідної інформації.

Після перевірки реєстраційних даних виконується хешування пароля та збереження користувача. При вході до системи сервер порівнює введений пароль із хешем, збереженим у базі даних, і в разі успішної перевірки формує сесію користувача. Саме завдяки цьому механізму сервер може повторно розпізнавати користувача без повторного введення облікових даних під час кожного запиту.

Не менш важливим є етап перевірки ролі під час виконання прикладних операцій. У поточній реалізації це виконується безпосередньо в контролерах. Наприклад, під час створення відгуку сервер спочатку визначає поточного користувача за сесією, а потім перевіряє, чи має він роль волонтера. Якщо користувач має іншу роль, виконання операції забороняється. Фрагмент такої перевірки наведено на рис. 3.4.3.

```
function isVolunteer(user) {
  return Number(user?.role_id) === 1;
}

const currentUser = await getCurrentUser(req);
if (!currentUser) {
  return res.status(401).json({ message: 'Потрібно увійти в систему.' });
}

if (!isVolunteer(currentUser)) {
  return res.status(403).json({
    message: 'Відгукатися на запити можуть лише волонтери.'
  });
}
```

*Рис. 3.4.3 Приклад коду перевірки ролі користувача під час створення відгуку*

Наведений приклад показує, що механізм рольового доступу реалізується на серверному рівні та не залежить лише від обмежень інтерфейсу клієнтської частини. Завдяки цьому система захищена від спроб виконання недозволених дій

навіть у випадку штучного формування HTTP-запитів поза межами стандартного інтерфейсу.

Загалом реалізований механізм автентифікації та рольового доступу поєднує кілька взаємопов’язаних складових: перевірку реєстраційних даних, хешування паролів, створення сесії користувача, зчитування сесійної інформації під час виконання захищених запитів і перевірку ролі на рівні контролерів. Саме така послідовність забезпечує не лише вхід користувача до системи, а й контроль допустимих дій у межах прикладної логіки серверної підсистеми.

Отже, механізми автентифікації та рольового доступу в серверній підсистемі вебсистеми «Допомога Зараз» реалізовано як невід’ємну частину загальної архітектури безпеки. Вони забезпечують ідентифікацію користувачів, підтримку авторизованого стану, захист облікових даних і серверний контроль за виконанням операцій відповідно до ролі користувача.

## 3.5 Реалізація бізнес-логіки обробки запитів, відгуків та звітів

Бізнес-логіка серверної підсистеми вебсистеми «Допомога Зараз» реалізує прикладні правила взаємодії між військовими та волонтерами, контролює допустимі переходи між станами об’єктів і забезпечує узгоджене завершення основних сценаріїв роботи системи. На відміну від структури бази даних або організації REST API, цей рівень відповідає не лише за збереження інформації, а й за перевірку того, чи відповідають дії користувачів встановленим правилам предметної області. Найбільш важливими з погляду прикладної логіки є сценарії створення запитів на допомогу, формування відгуків волонтерів і створення звітів, які завершують окремі етапи життєвого циклу запиту.

На початковому етапі взаємодії серверна частина обробляє створення дописів. У межах поточної реалізації допис виступає універсальною сутністю, яка може представляти як запит на допомогу, так і збір коштів. Під час створення такого об’єкта сервер перевіряє тип допису, коректність заголовка і текстового опису, допустиму кількість зображень, а також формує зв’язки з тегами. Крім

цього, на рівні контролера задається початковий статус допису, який надалі використовується для контролю доступних сценаріїв взаємодії.

Після появи відкритого запиту на допомогу серверна логіка дозволяє волонтерам створювати відгуки. Цей сценарій є принципово важливим, оскільки саме він переводить запит із пасивного стану в стан фактичної взаємодії між сторонами. При створенні відгуку сервер виконує кілька послідовних перевірок: наявність авторизованого користувача, відповідність його ролі, існування цільового запиту, тип допису та його поточний статус. Крім того, окремо перевіряється, чи не намагається користувач відгукнутися на власний запит і чи не існує вже раніше створеного відгуку від цього самого волонтера на той самий допис. Таким чином, об’єкт Response формується лише за умов дотримання всіх встановлених правил.

Фрагмент серверної логіки, яка забезпечує перевірку допустимості створення відгуку, наведено на рис. 3.5.1.

```
if (!isVolunteer(currentUser)) {
  return res.status(403).json({
    message: 'Відгукатися на запити можуть лише волонтери.'
  });
}
if (post.user_rnokpp === currentUser.user_rnokpp) {
  return res.status(403).json({
    message: 'Ви не можете відгукнутися на власний запит.'
  });
}
if (post.post_type_id !== 2) {
  return res.status(400).json({
    message: 'Відгук доступний лише для запитів на допомогу.'
  });
}
if (post.post_status === 'closed') {
  return res.status(400).json({
    message: 'Цей запит уже закрито.'
  });
}
```

*Рис. 3.5.1 Фрагмент перевірки умов створення відгуку*

Наведений фрагмент демонструє, що бізнес-логіка відгуків реалізується не на рівні інтерфейсу, а саме в серверному контролері. Це дозволяє зберігати цілісність прикладних правил навіть у разі прямого надсилання HTTP-запитів поза межами стандартної клієнтської частини.

Центральною темою даного підрозділу є логіка формування звітів, оскільки саме звітність завершує основний бізнес-процес системи та документує результат взаємодії між сторонами. У поточній реалізації сервер підтримує два типи звітів:

- звіт автора запиту про отримання допомоги (author);
- звіт волонтера про надання допомоги (helper).

Таке розмежування є принципово важливим, оскільки один і той самий факт взаємодії може бути зафіксований із двох різних позицій: з боку отримувача допомоги та з боку її надавача. Серверна логіка не зводить ці два сценарії до єдиного шаблону, а обробляє їх окремо, оскільки вони мають різні умови допустимості, різні зв’язки з іншими сутностями та різний вплив на зміну статусів об’єктів.

Основні властивості двох типів звітів наведено в табл. 3.5.1

*Таблиця 3.5.1*

**Типи звітів у серверній підсистемі**

| **Тип звіту** | **Хто може створити** | **Обов’язкові умови** | **Наслідок створення** |
|---|---|---|---|
| author | Автор запиту | Користувач має бути власником відповідного допису | Статус запиту (post) змінюється на closed |
| helper | Волонтер | Користувач має бути автором відповідного відгуку на цей запит | Статус відгуку (response) змінюється на closed |

Перед створенням звіту сервер виконує первинну валідацію вхідних даних. Перевіряється коректність ідентифікатора запиту, допустимість ролі автора

звіту, наявність тексту звіту, а для звіту типу helper — також наявність ідентифікатора відповідного відгуку. Це дозволяє відхилити некоректні запити ще до подальшої обробки. Фрагмент такої перевірки наведено на рис. 3.5.2.

```
const postId = Number(req.body.post_id);
const responseId = req.body.response_id ? Number(req.body.response_id) : null;
const reporterRole = String(req.body.reporter_role || '').trim().toLowerCase();
const reportText = String(req.body.text || '').trim();

if (!Number.isInteger(postId) || !['author', 'helper'].includes(reporterRole) ||
!reportText) {
  return res.status(400).json({ message: 'Некоректні дані звіту.' });
}
if (reporterRole === 'helper' && !Number.isInteger(responseId)) {
  return res.status(400).json({ message: 'Для цього звіту потрібен відгук
волонтера.' });
}
```

*Рис. 3.5.2 Приклад первинної валідації даних під час створення звіту*

Після успішної первинної перевірки сервер переходить до перевірки повноважень на створення звіту. Для звіту типу author контролер перевіряє, чи є поточний користувач власником відповідного допису. Для звіту типу helper окремо завантажується об’єкт відгуку, після чого перевіряється, чи саме поточний користувач є автором цього відгуку. У такий спосіб реалізується сувора прив’язка звіту до конкретного учасника взаємодії. Фрагмент цієї логіки наведено на рис. 3.5.3.

```
if (reporterRole === 'author') {
  if (post.user_rnokpp !== currentUser.user_rnokpp) {
    await client.query('ROLLBACK');
    return res.status(403).json({
      message: 'Ви не можете закрити чужий запит.'
    });
  }
} else {
  const responseResult = await client.query(
    `
      SELECT response_id, user_rnokpp
```

```
      FROM response
      WHERE response_id = $1 AND post_id = $2
    `,
    [responseId, postId]
  );

  const response = responseResult.rows[0];
  if (!response) {
    await client.query('ROLLBACK');
    return res.status(404).json({ message: 'Відгук не знайдено.' });
  }

  if (response.user_rnokpp !== currentUser.user_rnokpp) {
    await client.query('ROLLBACK');
    return res.status(403).json({
      message: 'Ви не можете закрити чужий відгук.'
    });
  }
}
```

*Рис. 3.5.3 Приклад перевірки прав на створення звіту*

Наведений фрагмент добре ілюструє, що звітність у системі не є формальним додатком до запиту, а виконує роль контрольної точки, для якої сервер перевіряє не лише структуру даних, а й повноваження конкретного користувача на завершення відповідного етапу процесу.

Після перевірки повноважень сервер створює новий запис звіту в таблиці report, а за наявності зображень — додає також записи до таблиці report_image. Саме на цьому етапі звітність набуває завершеної фізичної форми в базі даних. Важливо, що створення звіту та збереження його зображень виконується в межах однієї транзакції, що дає змогу уникнути часткового запису даних у разі помилки.

Принципово важливою властивістю звітної логіки є те, що створення звіту змінює стан пов’язаного об’єкта. Якщо звіт створює автор запиту, сервер переводить відповідний запис у таблиці post у стан closed. Якщо ж звіт створює волонтер, сервер переводить у стан closed відповідний запис у таблиці response.

Саме ця поведінка робить звіт не лише інформаційним документом, а й механізмом завершення окремого етапу бізнес-процесу. Фрагмент відповідної логіки наведено на рис. 3.5.4.

```
if (reporterRole === 'author') {
  await client.query(
    `
      UPDATE post
      SET post_status = 'closed'
      WHERE post_id = $1
    `,
    [postId]
  );
} else {
  await client.query(
    `
      UPDATE response
      SET response_status = 'closed'
      WHERE response_id = $1
    `,
    [responseId]
  );
}
```

*Рис. 3.5.4 Приклад зміни стану об’єктів після створення звіту*

Цей механізм має важливе архітектурне значення. Він забезпечує узгодженість між фактом формування звітної інформації та станом відповідної сутності в системі. Інакше кажучи, після того як допомога зафіксована в звіті, пов’язаний об’єкт уже не залишається в попередньому робочому стані. Завдяки цьому бізнес-логіка набуває завершеного циклічного характеру: запит створюється, на нього надходить відгук, після чого одна зі сторін формує звіт, а система фіксує завершення відповідного етапу.

Окремої уваги заслуговує отримання звітів користувача. У поточній реалізації сервер підтримує маршрут GET /reports/mine, який повертає звіти, створені поточним авторизованим користувачем. Під час формування відповіді контролер не обмежується лише даними з таблиці report, а виконує об’єднання з

таблицями post, app_user, role, report_image, post_image, post_tag і tag. У результаті формується розширене представлення звіту, яке містить не лише текст і час створення, а й зображення звіту, відомості про початковий запит, теги та інші пов’язані атрибути. Це дає змогу фронтенду відображати звіт у контексті повного сценарію взаємодії, а не як ізольований текстовий запис.

З погляду предметної області така організація звітності виконує одразу кілька функцій:

- документує факт отримання або надання допомоги;
- фіксує завершення окремого етапу життєвого циклу запиту;
- забезпечує цифрове збереження текстових і візуальних підтверджень;
- дозволяє користувачу переглядати власну історію завершених дій;
- створює основу для подальшого аналізу або зовнішнього ознайомлення зі сформованою звітною інформацією.

Таким чином, тема звітів у серверній підсистемі є не другорядним доповненням до механізму запитів і відгуків, а одним із центральних елементів усієї прикладної логіки. Саме звіти замикають основний цикл взаємодії між військовими та волонтерами, перетворюють факт надання допомоги на структурований цифровий запис і впливають на подальший стан об’єктів у системі. У поєднанні з логікою створення дописів і відгуків це формує завершений серверний механізм координації, документування та контролю взаємодії між сторонами.

## 3.6 Реалізація бізнес-логіки обробки запитів, відгуків і звітів

Додатковим елементом прикладної логіки серверної підсистеми є підтримка адміністративних дій. У поточній реалізації адміністратор має можливість переглядати список користувачів і дописів, а також виконувати видалення окремих об’єктів у межах встановлених обмежень. Зокрема, адміністратор може видаляти дописи, якщо з ними ще не пов’язані відгуки або звіти, а також видаляти користувачів лише за умови відсутності в них пов’язаних дописів, відгуків і звітів. Такий підхід дозволяє поєднати функції модерації із

збереженням цілісності даних і підтриманням коректної структури взаємозв’язків між основними сутностями системи.

# 4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

## 4.1 Тестування системи

Тестування серверної підсистеми вебсистеми «Допомога Зараз» проводилося з метою перевірки коректності реалізації основних функціональних можливостей, механізмів автентифікації, рольового доступу та прикладної бізнес-логіки обробки запитів, відгуків і звітів. Основна увага приділялася перевірці того, чи відповідає фактична поведінка серверної частини вимогам предметної області та чи забезпечує система узгоджену обробку даних у типових і помилкових сценаріях роботи.

У межах тестування перевірялися такі групи функцій:

- реєстрація та автентифікація користувачів;
- підтримка сесійного механізму доступу;
- створення, редагування та видалення дописів;
- створення відгуків волонтерами;
- формування звітів про отримання або надання допомоги;
- перегляд власних звітів;
- обмеження доступу до дій, не дозволених відповідній ролі.

Тестування мало переважно функціональний характер і виконувалося шляхом послідовної перевірки ключових сценаріїв взаємодії із серверним API. Для цього використовувалися локально розгорнута серверна частина, база даних PostgreSQL, клієнтський інтерфейс і засоби надсилання HTTP-запитів. У ході тестування аналізувалися як успішні сценарії роботи, так і випадки некоректних або недозволених дій користувача.

Одним із перших перевірявся сценарій реєстрації користувача. Контролювалося, чи коректно сервер приймає реєстраційні дані, перевіряє обов’язкові поля, формат РНОКПП, унікальність електронної пошти та створює новий обліковий запис у базі даних. Окремо перевірялися помилкові сценарії,

зокрема порожні поля, короткий пароль і некоректне значення ролі, які мали призводити до повернення відповідного повідомлення про помилку.

Наступним етапом було тестування входу до системи та підтримки сесії користувача. Перевірялося, чи формується сесія після успішної автентифікації, чи блокується доступ у разі введення неправильного пароля або неіснуючої адреси електронної пошти, а також як сервер реагує на запити до захищених маршрутів без активної сесії.

Окрему увагу було приділено перевірці операцій із дописами. У межах тестування контролювалися створення запиту на допомогу, додавання тегів і зображень, редагування власного допису, а також заборона на зміну або видалення об’єктів, які не належать поточному користувачеві. Додатково перевірялися початкові та кінцеві стани запитів, а також коректність створення дописів типу збору коштів.

Важливим блоком тестування було створення відгуків волонтерами. Успішним вважався сценарій, у якому авторизований користувач із роллю волонтера створював відгук на відкритий запит військового. Окремо перевірялися обмеження, закладені в бізнес-логіку системи. Зокрема, тестувалися такі заборонені ситуації:

- спроба військового створити відгук;
- спроба волонтера відгукнутися на власний допис;
- спроба створити відгук на закритий запит;
- спроба повторного створення відгуку одним і тим самим користувачем на той самий запит.

Результати цих перевірок є особливо важливими, оскільки саме вони підтверджують працездатність механізму рольового доступу та коректність прикладної логіки взаємодії між військовими та волонтерами.

У межах тестування також було перевірено роботу адміністративної панелі. Зокрема, контролювалася коректність перегляду списку користувачів і дописів, а також правильність обмежень на видалення об’єктів, для яких у системі вже існують пов’язані дані. Додаткові скриншоти, що ілюструють

результати тестування основних функціональних сценаріїв системи, наведено в додатку Б.

Найбільш детально тестувалася підсистема звітності, оскільки вона завершує основний цикл взаємодії користувачів у системі. Перевірялися два базові сценарії:

- створення звіту автором запиту про отримання допомоги;
- створення звіту волонтером про надання допомоги.

У першому випадку перевірялося, чи може лише автор відповідного запиту створити звіт типу author, а також чи переводиться після цього пов’язаний допис у стан closed. У другому випадку перевірялося, чи може лише автор відповідного відгуку створити звіт типу helper, а також чи змінюється після цього статус пов’язаного відгуку на closed. Окремо тестувалися помилкові сценарії:

- спроба створити звіт без тексту;
- спроба передати некоректний тип звіту;
- спроба створити звіт для чужого запиту;
- спроба створити звіт для чужого відгуку;
- спроба створити звіт типу helper без вказаного response_id.

Такі перевірки дозволили підтвердити, що звітність у системі реалізована не як формальне збереження тексту, а як повноцінний механізм завершення окремого етапу бізнес-процесу із перевіркою повноважень користувача та зміною станів пов’язаних об’єктів.

Для узагальнення основних результатів тестування доцільно подати короткий перелік перевірених сценаріїв у табличній формі.

*Таблиця 4.1.1*

**Основні сценарії тестування серверної підсистеми**

| **Сценарій** | **Очікуваний результат** |
|---|---|
| Реєстрація нового користувача | Користувач успішно створюється, сесія ініціалізується |

*Продовження Таблиці 4.1.1*

| **Сценарій** | **Очікуваний результат** |
|---|---|
| Вхід до системи | Сервер автентифікує користувача та формує сесію |
| Доступ до захищеного маршруту без сесії | Сервер повертає помилку авторизації |
| Створення запиту на допомогу | Новий допис зберігається в базі даних |
| Створення відгуку волонтером | Створюється новий об’єкт response |
| Спроба військового створити відгук | Сервер повертає помилку доступу |
| Створення звіту автором запиту | Створюється звіт, статус post змінюється на closed |
| Створення звіту волонтером | Створюється звіт, статус response змінюється на closed |
| Спроба створити звіт для чужого об’єкта | Сервер повертає помилку доступу |
| Отримання власних звітів | Сервер повертає список звітів поточного користувача |

За результатами проведених перевірок було встановлено, що серверна підсистема коректно реалізує основні функціональні можливості, забезпечує контроль доступу відповідно до ролі користувача та узгоджено обробляє життєвий цикл запитів, відгуків і звітів. Особливо важливим результатом є підтвердження працездатності підсистеми звітності, оскільки саме вона забезпечує фіксацію результатів надання допомоги та логічне завершення ключових сценаріїв взаємодії між користувачами.

Отже, проведене тестування підтвердило працездатність основних механізмів серверної частини вебсистеми «Допомога Зараз» і дало змогу виявити, що реалізовані функції відповідають закладеним вимогам до системи на рівні її прикладної логіки.

## 4.2 Вимоги до апаратного та програмного забезпечення

Для коректного впровадження та експлуатації вебсистеми «Допомога Зараз» вимоги до апаратного та програмного забезпечення доцільно розглядати з двох точок зору: з боку кінцевого користувача та з боку розробника або адміністратора системи. Кінцевий користувач працює із системою через вебінтерфейс, тому не потребує встановлення спеціалізованого програмного

забезпечення. Для роботи достатньо пристрою з доступом до мережі та сучасного веббраузера з підтримкою JavaScript, HTTP/HTTPS і cookies [3, 13], оскільки автентифікація в системі використовує cookie для збереження авторизованого стану.

Отже, мінімальні вимоги до клієнтського робочого місця користувача є такими:

- пристрій із доступом до мережі Інтернет або локальної мережі;
- сучасний веббраузер;
- увімкнена підтримка cookies;
- стабільне мережеве з’єднання.

З боку розробника або адміністратора вимоги є ширшими, оскільки охоплюють запуск серверної частини, підключення до бази даних, тестування API та подальше розгортання системи. Серверна підсистема реалізована на платформі Node.js [5] і взаємодіє з реляційною базою даних PostgreSQL [9], тому середовище має підтримувати запуск Node.js-застосунків, збереження змінних середовища та мережеву взаємодію.

На етапі локальної розробки й тестування серверна частина може працювати на персональному комп’ютері або ноутбуці з операційною системою Windows, Linux або macOS. Для базового запуску достатньо двоядерного процесора, 4 ГБ оперативної пам’яті та кількох гігабайтів вільного місця на диску. Для комфортної роботи доцільною є конфігурація з чотириядерним процесором, 8 ГБ оперативної пам’яті та SSD-накопичувачем.

До програмного забезпечення, необхідного для розробки, тестування та адміністрування, належать:

- Node.js;
- npm;
- PostgreSQL;
- редактор програмного коду;
- засоби командного рядка.

PostgreSQL може використовуватися як локально, так і у віддаленому середовищі. Для локальної розробки достатньо встановленого екземпляра PostgreSQL, а для зовнішнього розгортання база даних може бути винесена в хмарний сервіс Neon [15, 16]. Аналогічно серверна частина може бути розгорнута як локально, так і на платформі Render [14], що забезпечує запуск Node.js-застосунку, керування змінними середовища та мережеву доступність сервісу.

Для узагальнення основних вимог доцільно подати їх у табличній формі.

*Таблиця 4.2.1*

**Вимоги до апаратного та програмного забезпечення системи**

| **Категорія** | **Мінімальні вимоги** | **Рекомендовані вимоги** |
|---|---|---|
| Клієнтське робоче місце користувача | Пристрій із сучасним браузером, доступ до мережі, підтримка cookies | Ноутбук або ПК із актуальним браузером і стабільним Інтернет-з’єднанням |
| Серверна частина | 2-ядерний процесор, 4 ГБ ОЗП, Node.js, npm | 4-ядерний процесор, 8 ГБ ОЗП, SSD, Node.js, npm |
| Вузол бази даних | PostgreSQL, локальне або віддалене розміщення | PostgreSQL у виділеному середовищі або через хмарний сервіс Neon |

*Продовження Таблиці 4.2.1*

| **Категорія** | **Мінімальні вимоги** | **Рекомендовані вимоги** |
|---|---|---|
| Середовище розробки | Редактор коду, командний рядок, локальний PostgreSQL | Редактор коду, Node.js, PostgreSQL, Git, стабільний доступ до Інтернету |

Таким чином, з точки зору користувача вебсистема «Допомога Зараз» не потребує спеціального встановлення програмного забезпечення та може використовуватися через звичайний браузер. З точки зору розробки, тестування та впровадження система вимагає середовища, що підтримує Node.js, PostgreSQL та мережеву взаємодію, а також допускає як локальне, так і хмарне розгортання серверної частини й бази даних.

## 4.3 Склад інсталяційного пакету та порядок розгортання системи

Для впровадження вебсистеми «Допомога Зараз» необхідно підготувати інсталяційний пакет, який містить усі основні програмні компоненти, потрібні для запуску серверної частини, підключення до бази даних і роботи клієнтського інтерфейсу. Оскільки система реалізована як вебзастосунок із клієнтсько-серверною архітектурою, інсталяційний пакет охоплює програмний код серверної частини, статичні файли клієнтського інтерфейсу, конфігураційні файли середовища та SQL-дані для формування структури бази даних.

До складу інсталяційного пакету входять такі основні компоненти:

- каталог backend, який містить серверну частину системи, зокрема маршрути, контролери, модулі керування сесіями, доступу до бази даних, допоміжні сервіси та утиліти;
- каталог public, у якому розміщено статичні файли клієнтського інтерфейсу;
- файл package.json, що містить опис залежностей проєкту та команди запуску;

- файл .env.example, який використовується як шаблон для створення конфігурації середовища;
- SQL-файл neon-import.sql, що містить структуру та початкові дані для формування бази даних;
- файл render.yaml, який задає параметри розгортання серверної частини на платформі Render;
- каталог scripts, який містить допоміжні сценарії локальної підготовки та обслуговування проєкту.

Таким чином, інсталяційний пакет охоплює всі елементи, необхідні як для локального запуску системи, так і для її подальшого розгортання у зовнішньому середовищі.

З практичного погляду процес розгортання системи можна розглядати у двох варіантах:

- локальне розгортання для розробки та тестування;
- хмарне розгортання для зовнішнього доступу до серверної частини.

Під час локального розгортання спочатку необхідно встановити залежності проєкту, після чого створити файл конфігурації .env на основі шаблону .env.example. У цьому файлі задаються параметри підключення до бази даних, мережевий порт застосунку та, за потреби, параметри поштового сервісу. Після цього необхідно підготувати базу даних PostgreSQL [9] і виконати імпорт SQL-структури. У поточній реалізації для спрощення локального запуску передбачено сценарій автоматизованої підготовки, який встановлює залежності, створює конфігураційний файл і виконує імпорт структури бази даних.

Після завершення локальної підготовки запуск серверної частини виконується стандартною командою запуску Node.js-застосунку. Після старту система стає доступною через локальну адресу, а працездатність сервера може бути перевірена через службовий маршрут /health.

Узагальнений порядок локального розгортання можна подати так:

1. отримати копію проєкту та перейти до каталогу застосунку;
2. встановити залежності;

3. створити та заповнити файл .env;
4. підготувати базу даних PostgreSQL;
5. виконати імпорт SQL-структури;
6. запустити серверну частину;
7. перевірити доступність системи через браузер або маршрут /health.

Окрему увагу слід приділити хмарному розгортанню, оскільки саме воно забезпечує зовнішній доступ до серверної частини. У поточному проєкті як платформа для розміщення серверного застосунку використовується Render [14], а як сервіс для розміщення PostgreSQL — Neon [15, 16]. Такий підхід дозволяє розділити обчислювальний вузол серверної частини та вузол бази даних, що є типовим для сучасних вебзастосунків.

Під час розгортання на платформі Render [14] до репозиторію підключається готовий конфігураційний файл render.yaml, у якому задано тип сервісу, середовище виконання, команду збирання, команду запуску та маршрут перевірки працездатності. Відповідно до наявної конфігурації для проєкту використовуються:

- команда збирання npm install;
- команда запуску npm start;
- маршрут перевірки працездатності /health.

Для наочного подання розміщення основних компонентів системи у середовищі розгортання доцільно використати діаграму розгортання, наведену на рис. 4.3.1.

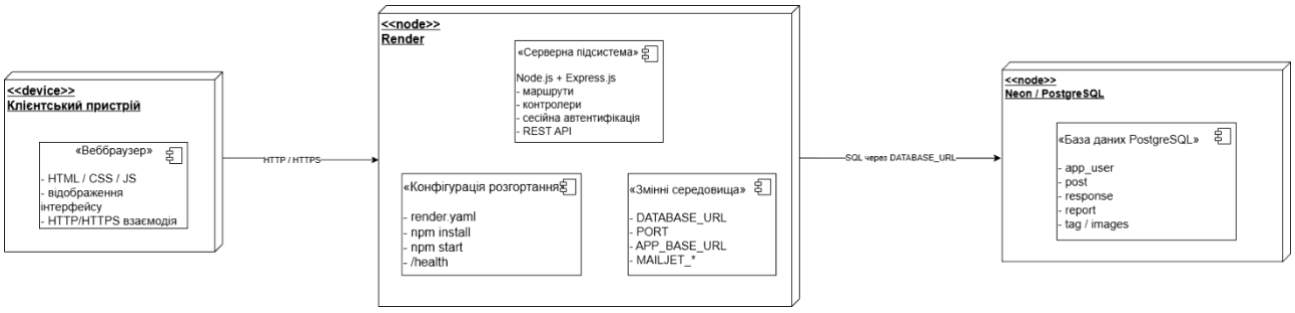



*Рис. 4.3.1 Діаграма розгортання вебсистеми «Допомога Зараз»*

Як показано на рис. 4.3.1, клієнтський пристрій взаємодіє із серверною частиною через мережу, серверний застосунок може бути розгорнутий на

платформі Render, а база даних PostgreSQL — розміщена локально або у хмарному середовищі Neon.

Для коректної роботи системи у хмарному середовищі необхідно також задати змінні середовища. Основними з них є:

- DATABASE_URL — рядок підключення до бази даних PostgreSQL, розміщеної в Neon;
- MAILJET_API_KEY і MAILJET_SECRET_KEY — параметри доступу до поштового сервісу;
- SMTP_FROM — адреса відправника листів;
- APP_BASE_URL — базова адреса розгорнутого застосунку;
- PORT — мережевий порт, який використовується платформою Render.

Порядок хмарного розгортання можна подати так:

1. створити або підготувати екземпляр PostgreSQL у сервісі Neon;
2. отримати рядок підключення DATABASE_URL;
3. створити вебсервіс на платформі Render;
4. підключити репозиторій проєкту;
5. задати необхідні змінні середовища;
6. виконати розгортання сервісу;
7. перевірити працездатність застосунку через публічну адресу та маршрут /health.

З погляду впровадження така схема є зручною, оскільки серверна частина не потребує ручного налаштування великої кількості інфраструктурних компонентів. Основні параметри зосереджені у змінних середовища та конфігураційних файлах, а сам процес публікації виконується засобами Render і Neon. Це спрощує перенесення системи між локальним і хмарним середовищами, а також зменшує кількість дій, які необхідно виконувати адміністратору під час первинного запуску або повторного розгортання. Крім того, такий підхід забезпечує централізоване керування конфігурацією та підвищує зручність подальшого супроводу системи. Для узагальнення складу інсталяційного пакету доцільно подати його у табличній формі.

*Таблиця 4.3.1*

**Склад інсталяційного пакету системи**

| **Компонент** | **Призначення** |
|---|---|
| backend/ | Серверна логіка, маршрути, контролери, модулі доступу до БД |
| public/ | Статичні файли клієнтського інтерфейсу |
| package.json | Опис залежностей і команд запуску проєкту |
| .env.example | Шаблон конфігурації середовища |
| neon-import.sql | SQL-структура та початкові дані бази даних |
| render.yaml | Конфігурація розгортання серверної частини на Render |
| scripts/ | Допоміжні сценарії локальної підготовки та обслуговування |

Отже, інсталяційний пакет вебсистеми «Допомога Зараз» містить усі необхідні програмні та конфігураційні компоненти для локального запуску, тестування й подальшого хмарного розгортання. Використання PostgreSQL, Render і Neon дозволяє реалізувати як локальний, так і віддалений сценарій впровадження системи, а централізація конфігурації у змінних середовища спрощує процес перенесення проєкту між різними середовищами виконання.

# ВИСНОВКИ

У бакалаврській роботі досліджено проблему організації інформаційної взаємодії між військовими підрозділами та волонтерами в межах надання допомоги. Установлено, що використання загальнодоступних каналів комунікації не забезпечує належного рівня структурованого обліку потреб, контролю станів запитів, рольового розмежування дій та формування цифрової звітності. Це підтвердило актуальність розробки спеціалізованої вебсистеми із серверною реалізацією бізнес-логіки.

У процесі виконання роботи проведено аналіз предметної області та існуючих аналогів, за результатами якого обґрунтовано доцільність створення власної серверної підсистеми. Визначено, що наявні рішення не забезпечують одночасної підтримки рольової моделі, централізованого збереження даних, контролю життєвого циклу запиту та цифрового документування результатів взаємодії.

У межах роботи спроєктовано концептуальну, логічну та фізичну модель бази даних, яка охоплює основні сутності предметної області: користувачів, ролі, дописи, відгуки, звіти, теги та зображення. Обґрунтовано вибір PostgreSQL як системи керування базами даних і реалізовано фізичну модель засобами SQL. Це забезпечило структуровану основу для збереження даних і підтримки зв’язків між основними об’єктами системи.

Розроблено архітектуру серверної частини, побудовану на розділенні відповідальності між маршрутами, контролерами, механізмами керування сесіями, модулем доступу до бази даних і допоміжними сервісами. У межах серверної підсистеми реалізовано REST API для роботи з автентифікацією, дописами, відгуками, звітами та профілями користувачів.

Практичним результатом роботи стала реалізація механізмів автентифікації та рольового доступу, а також бізнес-логіки обробки запитів, відгуків і звітів. Особливе значення має підсистема звітності, яка забезпечує

фіксацію результатів надання допомоги та зміну станів пов’язаних об’єктів, завершуючи основний цикл взаємодії між військовими та волонтерами.

Проведене тестування підтвердило працездатність основних функціональних можливостей серверної частини, коректність реалізації рольового доступу, сесійної автентифікації та логіки створення дописів, відгуків і звітів. Також визначено вимоги до апаратного й програмного забезпечення та описано порядок локального і хмарного розгортання системи.

Отже, поставлену мету досягнуто: розроблено серверну підсистему вебсистеми обліку взаємодії між військовими та волонтерами, яка забезпечує централізоване збереження даних, підтримку REST API, рольовий доступ і цифрову звітність щодо результатів надання допомоги. Отримані результати можуть бути використані як основа для подальшого розвитку та практичного впровадження системи.

Перспективами подальшого розвитку є розширення механізмів зовнішнього контролю звітності, удосконалення хмарного розгортання та подальше функціональне розширення вебсистеми.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fielding R. T. Architectural Styles and the Design of Network-Based Software Architectures [Electronic resource]. 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf (дата звернення: 16.05.2026).

2. Object Management Group. Unified Modeling Language (UML), Version 2.5.1 [Electronic resource]. URL: https://www.omg.org/spec/UML/2.5.1 (дата звернення: 16.05.2026).

3. Nottingham M., Fielding R., Reschke J. HTTP Semantics. RFC 9110 [Electronic resource]. June 2022. URL: https://www.rfc-editor.org/rfc/rfc9110 (дата звернення: 16.05.2026).

4. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259 [Electronic resource]. December 2017. URL: https://www.rfc-editor.org/rfc/rfc8259 (дата звернення: 16.05.2026).

5. Node.js v26.2.0 documentation [Electronic resource]. URL: https://nodejs.org/api/documentation.html (date of access: 16.05.2026).

6. Express.js. Installing [Electronic resource]. URL: https://expressjs.com/en/starter/installing.html (дата звернення: 17.05.2026).

7. Express.js. Routing [Electronic resource]. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 17.05.2026).

8. Express.js. Using middleware [Electronic resource]. URL: https://expressjs.com/en/guide/using-middleware (дата звернення: 17.05.2026).

9. PostgreSQL Global Development Group. PostgreSQL 18 documentation [Electronic resource]. URL: https://www.postgresql.org/docs/current/index.html (дата звернення: 18.05.2026).

10. node-postgres. Pool API [Electronic resource]. URL: https://node-postgres.com/apis/pool (дата звернення: 18.05.2026).

11. node-postgres. Pooling [Electronic resource]. URL: https://node-postgres.com/features/pooling (дата звернення: 18.05.2026).

12. bcrypt - npm [Electronic resource]. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 18.05.2026).

13. MDN Web Docs. Using HTTP cookies [Electronic resource]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies (дата звернення: 19.05.2026).

14. Render Docs. Deploying on Render [Electronic resource]. URL: https://render.com/docs/deploys (дата звернення: 19.05.2026).

15. Neon Docs. Connecting Neon to your stack [Electronic resource]. URL: https://neon.com/docs/get-started-with-neon/connect-neon (дата звернення: 20.05.2026).

16. Neon Docs. Architecture overview [Electronic resource]. URL: https://neon.com/docs/introduction/architecture-overview (дата звернення: 20.05.2026).

17. Charitable Foundation “Come Back Alive”. About the foundation [Electronic resource]. URL: https://savelife.in.ua/en/intro-en/ (дата звернення: 13.05.2026).

18. Благодійний фонд Сергія Притули. Про фонд [Електронний ресурс]. URL: https://prytulafoundation.org/about (дата звернення: 13.05.2026).

19. GoFundMe. How fundraising works on GoFundMe [Electronic resource]. URL: https://support.gofundme.com/hc/en-us/articles/4405037344411-How-fundraising-works-on-GoFundMe (дата звернення: 13.05.2026)

20. UNITED24. About [Electronic resource]. URL: https://u24.gov.ua/uk/about (дата звернення: 13.05.2026).

21. dobro.ua. Про нас [Електронний ресурс]. URL: https://dobro.ua/about_us/ (дата звернення: 13.05.2026).

22. Telegram. FAQ [Electronic resource]. URL: https://telegram.org/faq (дата звернення 13.05.2026).

23. Facebook Help Center. The Meta Company Products [Electronic resource]. URL: https://www.facebook.com/help/195227921252400 (дата звернення: 13.05.2026).

# ДОДАТКИ

## Додаток А

```
CREATE TABLE "role" (
  "role_id" integer PRIMARY KEY,
  "role_name" varchar
);

CREATE TABLE "app_user" (
  "user_rnokpp" varchar PRIMARY KEY,
  "role_id" integer,
  "user_name" varchar,
  "user_email" varchar,
  "password_hash" varchar,
  "user_description" text,
  "user_image_url" varchar
);

CREATE TABLE "tag" (
  "tag_id" integer PRIMARY KEY,
  "tag_name" varchar
);

CREATE TABLE "user_tag" (
  "tag_id" integer,
  "user_rnokpp" varchar
);

CREATE TABLE "post_type" (
  "post_type_id" integer PRIMARY KEY,
```

```
  "post_type_name" varchar
);

CREATE TABLE "post" (
  "post_id" integer PRIMARY KEY,
  "user_rnokpp" varchar,
  "post_type_id" integer,
  "post_title" varchar,
  "post_description" text,
  "post_status" varchar,
  "post_datetime" timestamp
);

CREATE TABLE "post_tag" (
  "tag_id" integer,
  "post_id" integer
);

CREATE TABLE "post_image" (
  "post_image_id" integer PRIMARY KEY,
  "post_id" integer,
  "post_image_url" varchar
);

CREATE TABLE "response" (
  "response_id" integer PRIMARY KEY,
  "user_rnokpp" varchar,
  "post_id" integer,
  "response_title" varchar,
  "response_description" text,
```

```
  "response_datetime" timestamp
);

CREATE TABLE "response_image" (
  "response_image_id" integer PRIMARY KEY,
  "response_id" integer,
  "response_image_url" varchar
);

CREATE TABLE "report" (
  "report_number" integer PRIMARY KEY,
  "user_rnokpp" varchar,
  "post_id" integer,
  "report_type" varchar,
  "report_text" text,
  "creation_datetime" timestamp
);

CREATE TABLE "report_image" (
  "report_image_id" integer PRIMARY KEY,
  "report_number" integer,
  "report_image_url" varchar
);

COMMENT ON COLUMN "app_user"."user_rnokpp" IS 'Унікальний ідентифікатор, наприклад, РНОКПП';

ALTER TABLE "app_user" ADD FOREIGN KEY ("role_id") REFERENCES "role" ("role_id") DEFERRABLE INITIALLY IMMEDIATE;
```

ALTER TABLE "user_tag" ADD FOREIGN KEY ("tag_id") REFERENCES "tag" ("tag_id") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "user_tag" ADD FOREIGN KEY ("user_rnokpp") REFERENCES "app_user" ("user_rnokpp") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "post" ADD FOREIGN KEY ("user_rnokpp") REFERENCES "app_user" ("user_rnokpp") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "post" ADD FOREIGN KEY ("post_type_id") REFERENCES "post_type" ("post_type_id") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "post_tag" ADD FOREIGN KEY ("tag_id") REFERENCES "tag" ("tag_id") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "post_tag" ADD FOREIGN KEY ("post_id") REFERENCES "post" ("post_id") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "post_image" ADD FOREIGN KEY ("post_id") REFERENCES "post" ("post_id") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "response" ADD FOREIGN KEY ("user_rnokpp") REFERENCES "app_user" ("user_rnokpp") DEFERRABLE INITIALLY IMMEDIATE;

ALTER TABLE "response" ADD FOREIGN KEY ("post_id") REFERENCES "post" ("post_id") DEFERRABLE INITIALLY IMMEDIATE;

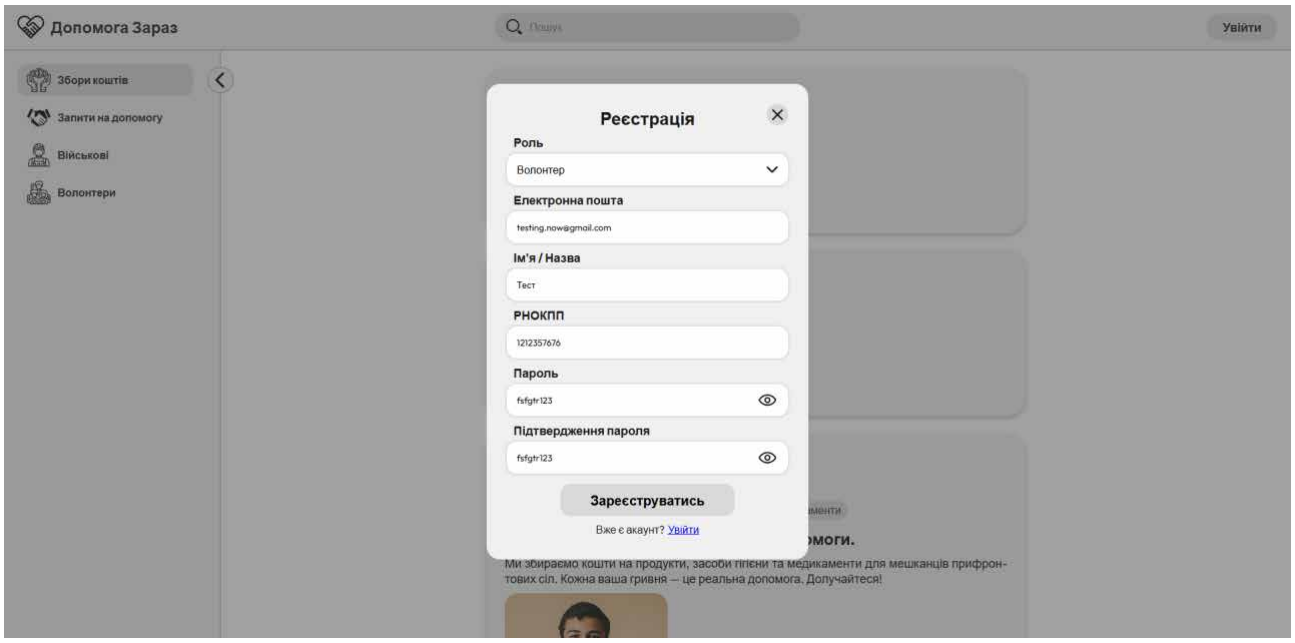
ALTER TABLE "response_image" ADD FOREIGN KEY ("response_id") REFERENCES "response" ("response_id") DEFERRABLE INITIALLY IMMEDIATE;

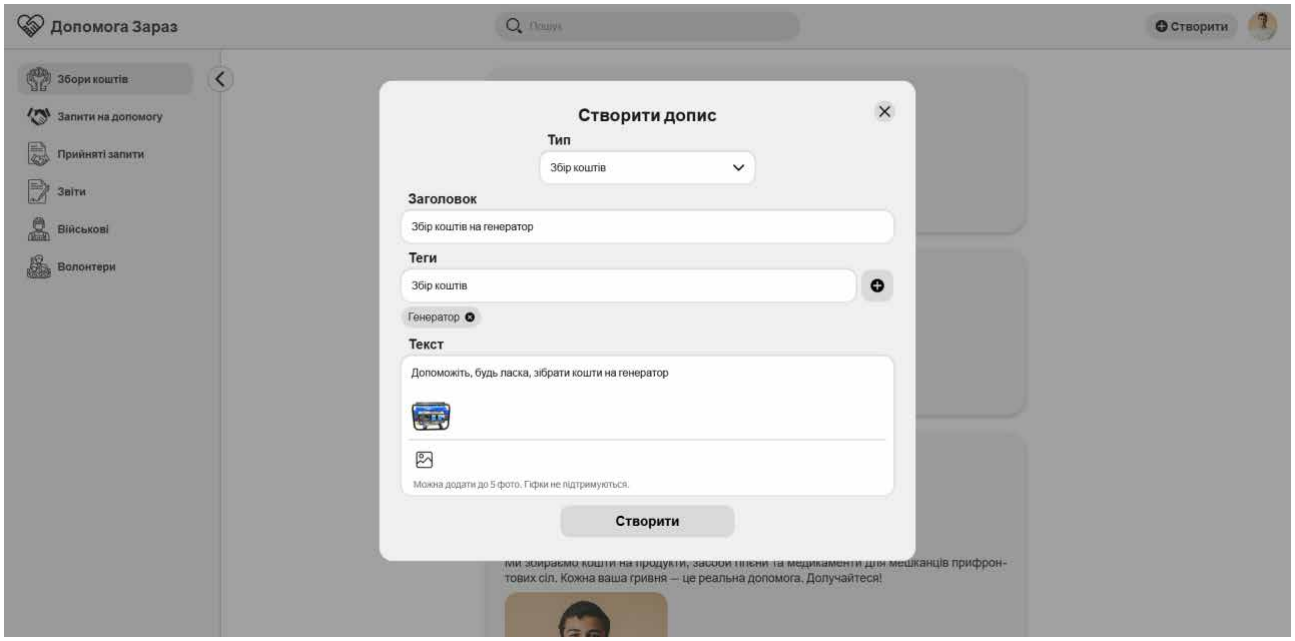
ALTER TABLE "report" ADD FOREIGN KEY ("user_rnokpp") REFERENCES "app_user" ("user_rnokpp") DEFERRABLE INITIALLY IMMEDIATE;

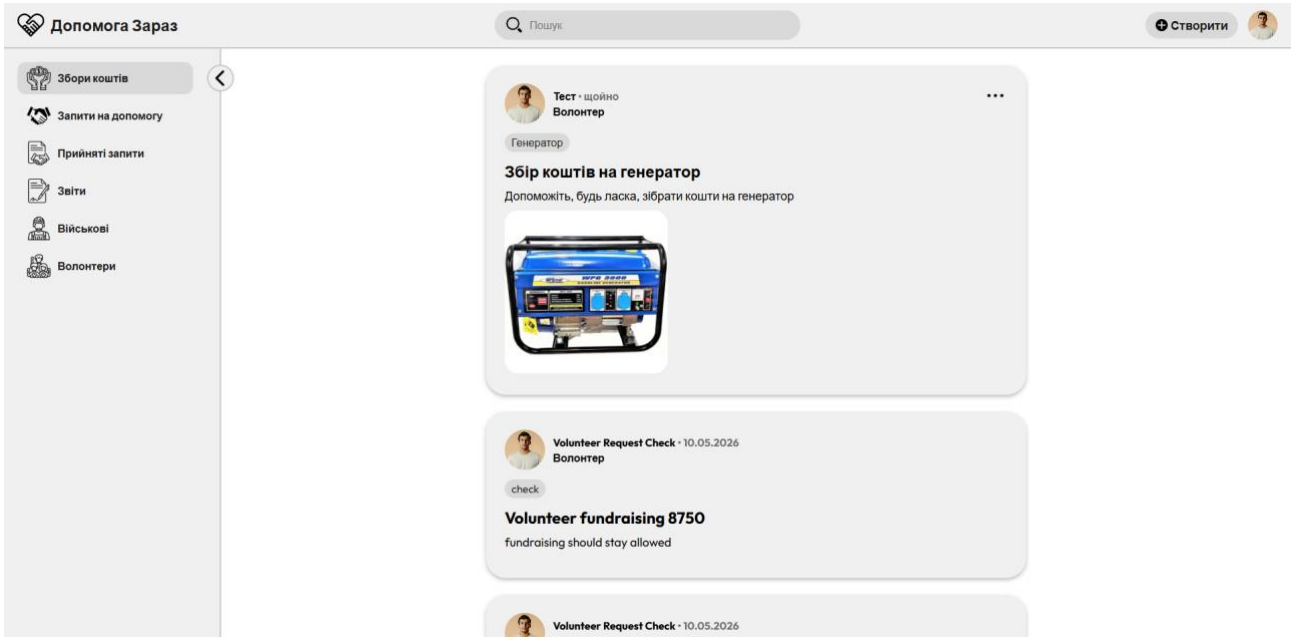
ALTER TABLE "report" ADD FOREIGN KEY ("post_id") REFERENCES "post" ("post_id") DEFERRABLE INITIALLY IMMEDIATE;

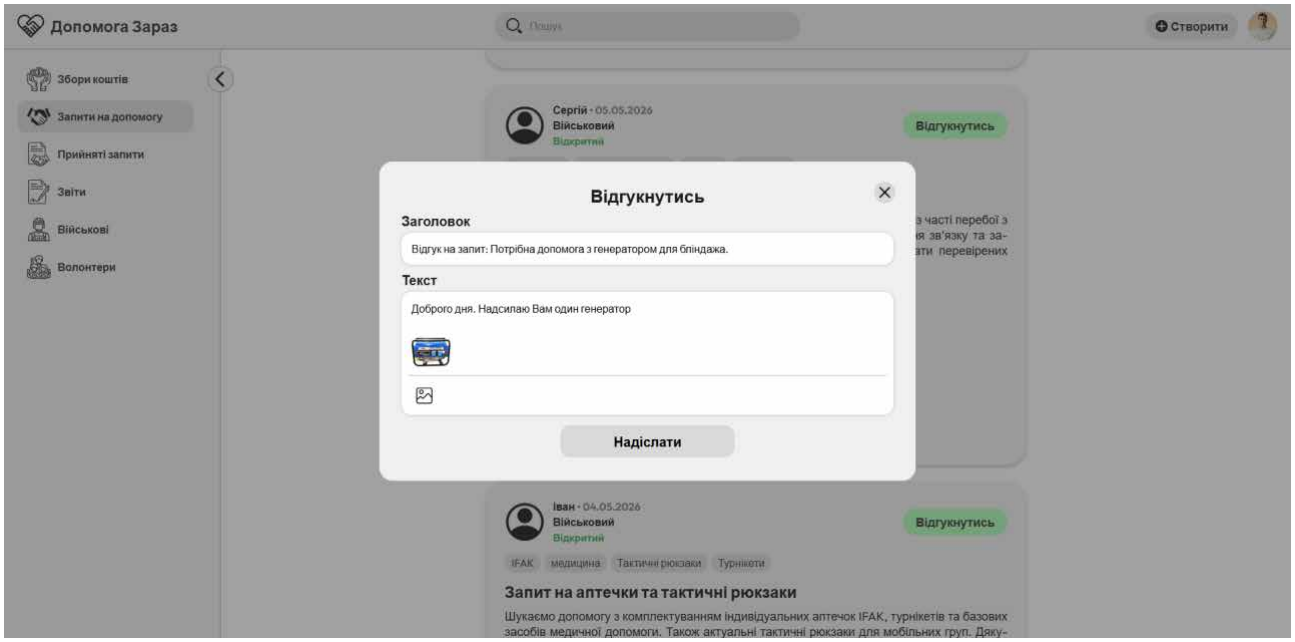
ALTER TABLE "report_image" ADD FOREIGN KEY ("report_number") REFERENCES "report" ("report_number") DEFERRABLE INITIALLY IMMEDIATE;

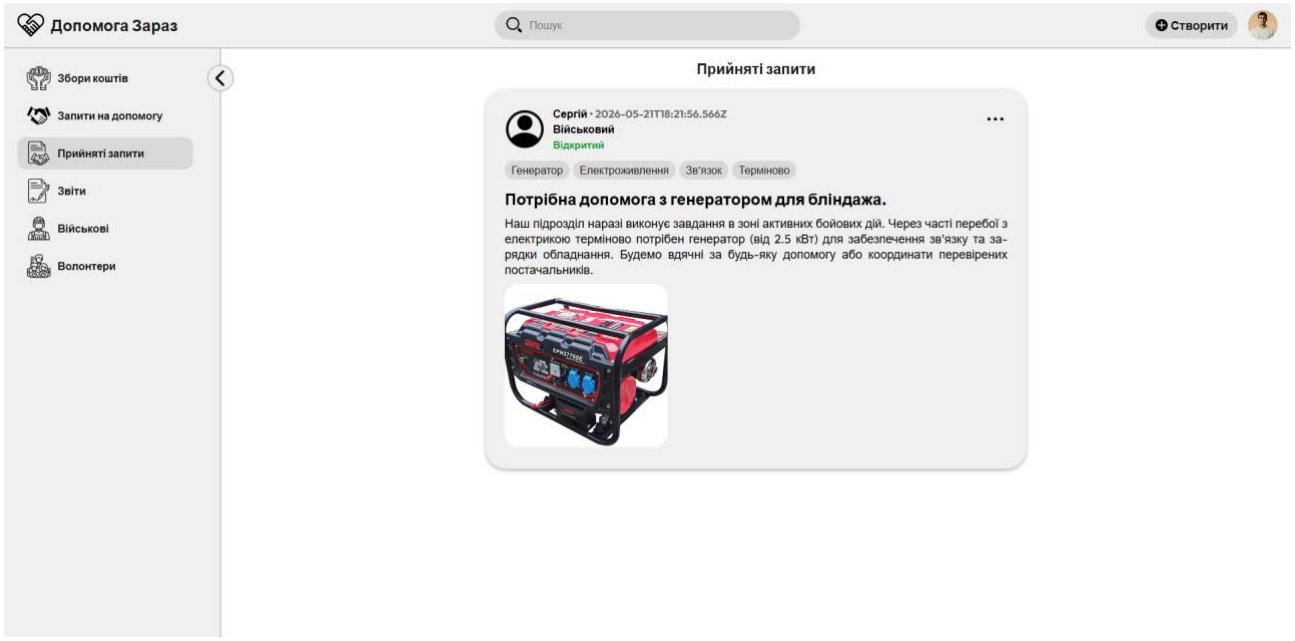
# Додаток Б

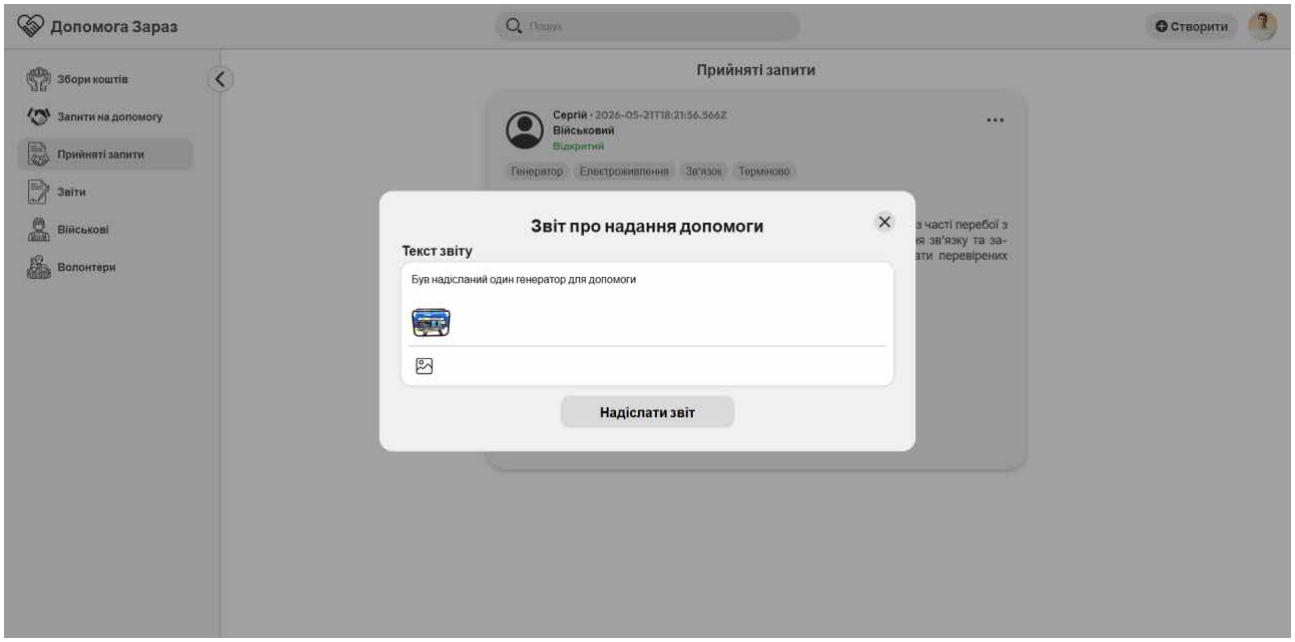

Допомога Зараз
Пошук
Увійти
Збори коштів
Запити на допомогу
Військові
Волонтери
Реєстрація
Роль
Волонтер
Електронна пошта
testing.now@gmail.com
Ім'я / Назва
Тест
РНОКПП
1212357676
Пароль
fsfgtr123
Підтвердження пароля
fsfgtr123
Зареєструватись
Вже є акаунт? Увійти
Ми збираємо кошти на продукти, засоби гігієни та медикаменти для мешканців прифронтових сіл. Кожна ваша гривня — це реальна допомога. Долучайтеся!

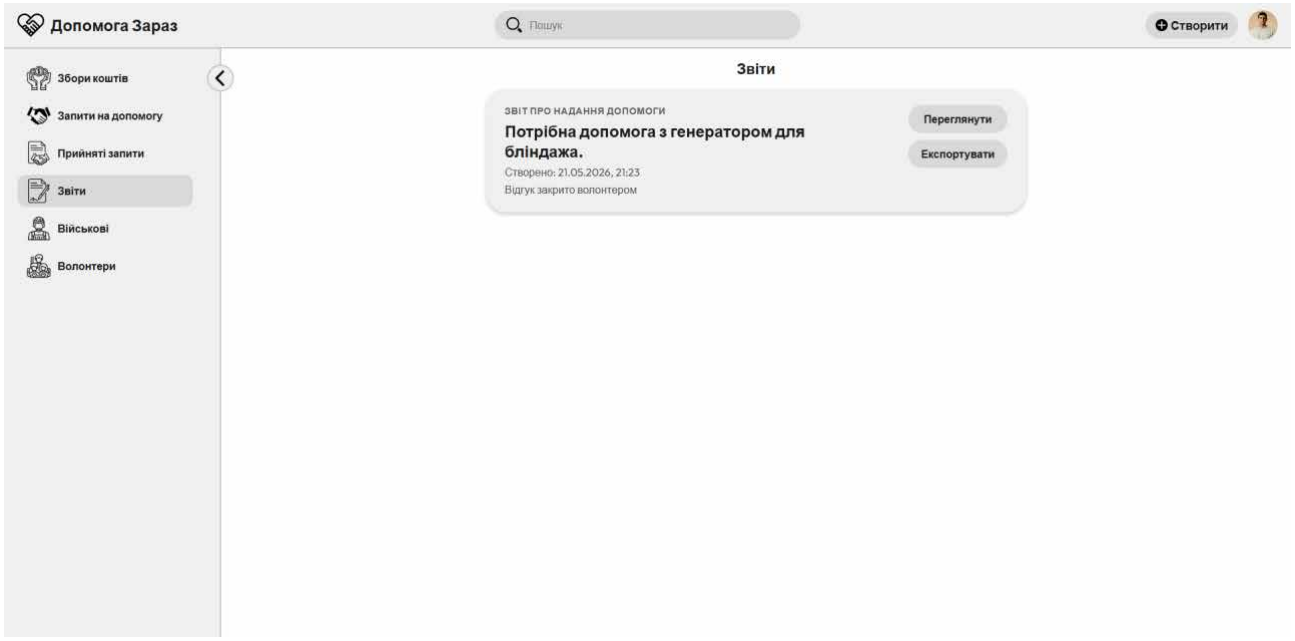

Допомога Зараз
Пошук
Створити
Збори коштів
Запити на допомогу
Прийняті запити
Звіти
Військові
Волонтери
Створити допис
Тип
Збір коштів
Заголовок
Збір коштів на генератор
Теги
Збір коштів
Генератор
Текст
Допоможіть, будь ласка, зібрати кошти на генератор
Можна додати до 5 фото. Гіфки не підтримуються.
Створити
Ми збираємо кошти на продукти, засоби гігієни та медикаменти для мешканців прифронтових сіл. Кожна ваша гривня — це реальна допомога. Долучайтеся!

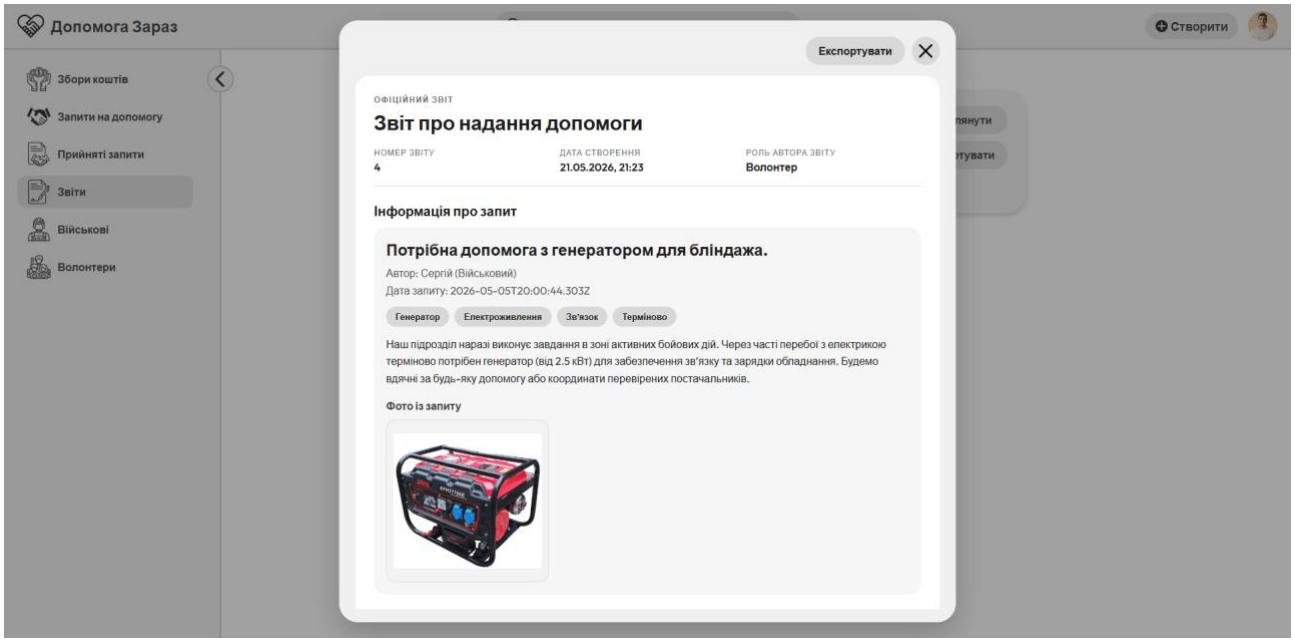

Допомога Зараз
Пошук
Створити
Збори коштів
Запити на допомогу
Прийняті запити
Звіти
Військові
Волонтери
Тест • щойно
Волонтер
Генератор
Збір коштів на генератор
Допоможіть, будь ласка, зібрати кошти на генератор
Volunteer Request Check • 10.05.2026
Волонтер
check
Volunteer fundraising 8750
fundraising should stay allowed
Volunteer Request Check • 10.05.2026

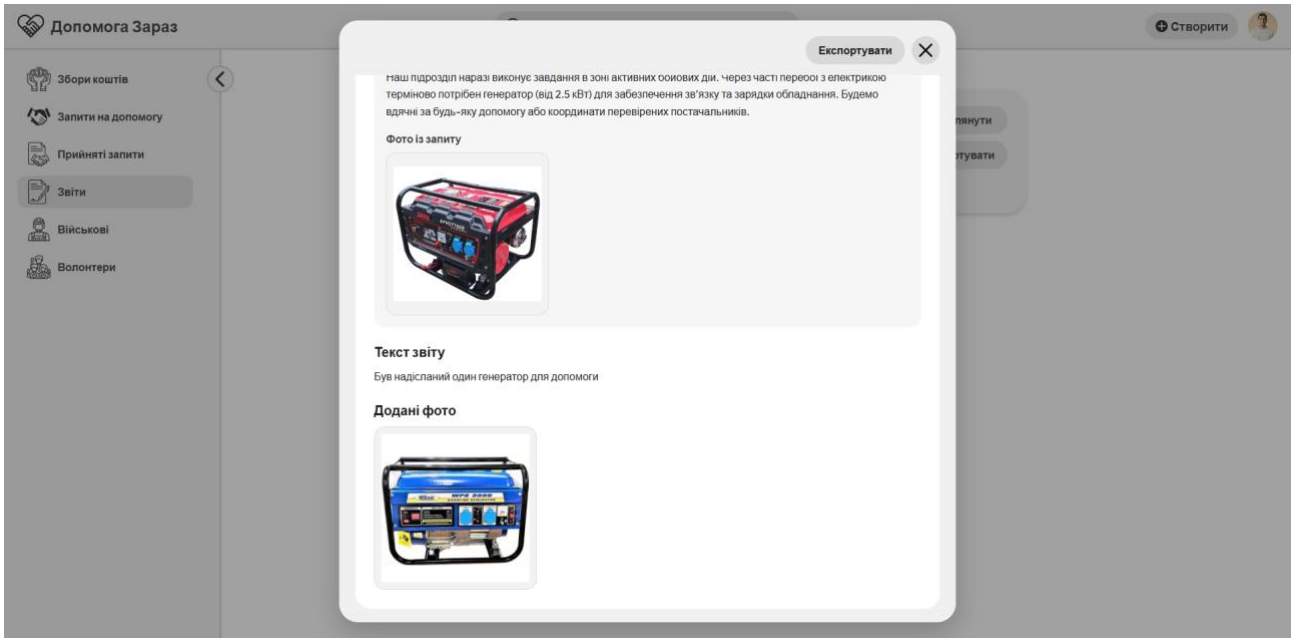

Допомога Зараз
Пошук
Створити
Збори коштів
Запити на допомогу
Прийняті запити
Звіти
Військові
Волонтери
Сергій • 05.05.2026
Військовий
Відкритий
Відгукнутись
Відгукнутись
Заголовок
Відгук на запит: Потрібна допомога з генератором для бліндажа.
Текст
Доброго дня. Надсилаю Вам один генератор
Надіслати
Іван • 04.05.2026
Військовий
Відкритий
Відгукнутись
IFAK
медицина
Тактичні рюкзаки
Турнікети
Запит на аптечки та тактичні рюкзаки
Шукаємо допомогу з комплектуванням індивідуальних аптечок IFAK, турнікетів та базових
засобів медичної допомоги. Також актуальні тактичні рюкзаки для мобільних груп. Дяку-


Допомога Зараз
Пошук
Створити
Збори коштів
Запити на допомогу
Прийняті запити
Звіти
Військові
Волонтери
Прийняті запити
Сергій • 2026-05-21T18:21:56.566Z
Військовий
Відкритий
Генератор
Електроживлення
Зв'язок
Терміново
Потрібна допомога з генератором для бліндажа.
Наш підрозділ наразі виконує завдання в зоні активних бойових дій. Через часті перебої з електрикою терміново потрібен генератор (від 2.5 кВт) для забезпечення зв'язку та зарядки обладнання. Будемо вдячні за будь-яку допомогу або координати перевірених постачальників.


Допомога Зараз
Створити
Збори коштів
Запити на допомогу
Прийняті запити
Звіти
Військові
Волонтери
Прийняті запити
Сергій • 2026-05-21T18:21:56.566Z
Військовий
Відкритий
Генератор
Електроживлення
Зв'язок
Терміново
Звіт про надання допомоги
Текст звіту
Був надісланий один генератор для допомоги
Надіслати звіт


Допомога Зараз
Пошук
Створити
Збори коштів
Запити на допомогу
Прийняті запити
Звіти
Військові
Волонтери
Звіти
ЗВІТ ПРО НАДАННЯ ДОПОМОГИ
Потрібна допомога з генератором для бліндажа.
Створено: 21.05.2026, 21:23
Відгук закрито волонтером
Переглянути
Експортувати


Допомога Зараз
Створити
Експортувати
ОФІЦІЙНИЙ ЗВІТ
Звіт про надання допомоги
НОМЕР ЗВІТУ
4
ДАТА СТВОРЕННЯ
21.05.2026, 21:23
РОЛЬ АВТОРА ЗВІТУ
Волонтер
Інформація про запит
Потрібна допомога з генератором для бліндажа.
Автор: Сергій (Військовий)
Дата запиту: 2026-05-05T20:00:44.303Z
Генератор
Електроживлення
Зв'язок
Терміново
Наш підрозділ наразі виконує завдання в зоні активних бойових дій. Через часті перебої з електрикою терміново потрібен генератор (від 2.5 кВт) для забезпечення зв'язку та зарядки обладнання. Будемо вдячні за будь-яку допомогу або координати перевірених постачальників.
Фото із запиту


Допомога Зараз
Створити
Збори коштів
Запити на допомогу
Прийняті запити
Звіти
Військові
Волонтери
Експортувати
Наш підрозділ наразі виконує завдання в зоні активних бойових дій, через часті перебої з електрикою терміново потрібен генератор (від 2.5 кВт) для забезпечення зв'язку та зарядки обладнання. Будемо вдячні за будь-яку допомогу або координати перевірених постачальників.
Фото із запиту
Текст звіту
Був надісланий один генератор для допомоги
Додані фото

**Додаток В**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**Декларація про академічну доброчесність та використання ШІ**

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

1. Моя бакалаврська кваліфікаційна робота на тему «Підсистема адміністрування та реалізації бізнес-логіки системи обліку взаємодії волонтерів та 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 генеративного ШІ не використовувалися.
- [✓] інструмент ШІ (Gemini) був використаний для:
  - [✓] перекладу іншомовних джерел;
  - [✓] стилістичного редагування та перевірки граматики;
  - [✓] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

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

«21» травня 2026 р. _____ **Антон БАЧИНСЬКИЙ**

(підпис)