Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка веб-платформи для координації волонтерської діяльності»

Виконала ст. гр. КН-22-1 _____ Пелюхова А. Д.

Керівник _____ Сьомочкина С. В.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2026

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *28* » *січня* 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-22-1 *Пелюховій Альбіні Дмитрівні*

1. Тема кваліфікаційної роботи: *«Розробка веб-платформи для координації волонтерської діяльності»*

затверджено наказом по університету № *173с* від *30.03.2026 р.*

2. Термін здачі кваліфікаційної роботи: *10.06.2026 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 83с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Сьомочкина С. В.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.26*
2	*Розділ 1*	*05.04.26*
3	*Розділ 2*	*01.05.26*
4	*Висновки*	*25.05.26*
5	*Оформлення кваліфікаційної роботи*	*28.05.26*
6	*Підготовка презентації та графічного матеріалу*	*20.05.26*
7	*Підготовка доповіді до захисту*	*10.06.26*

6. Дата видачі завдання: *28.01.2026р.*

Керівник _____ / Сьомочкина С. В./

7. Запевнення: *Я, Пелюхова Альбіна Дмитрівна, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студентка _____ / Пелюхова А. Д./

АНОТАЦІЯ

Пелюхова А. Д. Розробка веб-платформи для координації волонтерської діяльності: кваліфікаційна робота бакалавра : 122 – Комп'ютерні науки. Кривий Ріг. Криворізький національний університет, 2026. 81 с.

Актуальність роботи зумовлена необхідністю створення єдиного консолідованого ІТ-простору для ефективної координації волонтерських ініціатив в Україні та автоматизації гуманітарних потоків даних за допомогою сучасних веб-технологій.

Метою роботи є проектування, програмна реалізація та тестування SPA-платформи «UNITY+» для автоматизації процесів рекрутингу, таск-менеджменту та верифікації учасників волонтерської діяльності з використанням бібліотеки React та BaaS-платформи Appwrite.

У першому розділі виконано системний аналіз предметної області, визначено архітектурні обмеження наявних програмних аналогів та інженерно обґрунтовано вибір Single Page Application архітектури на базі екосистеми JavaScript (React + Appwrite).

У другому розділі представлено процеси структурно-функціонального проектування системи із застосуванням методологій SADT, DFD та UML-моделювання. Описано схему даних хмарної СУБД, технічні особливості реалізації фронтенд-модулів (Канбан-дошка, інтерактивна картографія, авторизація JWT) та наведено результати тестування і оцінки адаптивного UI/UX дизайну додатка.
Ключові слова:

АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ, ВЕБ-ПЛАТФОРМА, ВОЛОНТЕРСЬКА ДІЯЛЬНІСТЬ, КООРДИНАЦІЯ, СИСТЕМНИЙ АНАЛІЗ, APPWRITE, REACT, SINGLE PAGE APPLICATION, UML-МОДЕЛЮВАННЯ

ЗМІСТ

ВСТУП

Останнім часом для розв'язання задач координації взаємодії стейкхолдерів активно використовуються різноманітні цифрові комунікаційні інструменти. Проте використання розрізнених месенджерів (Telegram, Viber) та публічних соціальних мереж (Facebook, Instagram) має низку критичних деструктивних недоліків. До них належать: швидка втрата актуальності інформаційних повідомлень у загальних динамічних чатах, відсутність структурованого та категоріального пошуку за геопозицією чи специфікою запитів, дублювання ініціатив через брак єдиного координаційного центру, а також високі ризики шахрайства й дезінформації через відсутність інтегрованих інструментів верифікації волонтерів та організацій.

Актуальність теми. Ефективна координація волонтерської діяльності, швидкий розподіл гуманітарних, медичних та матеріально-технічних ресурсів є одними з найважливіших завдань для підтримання життєдіяльності суспільства в умовах сучасних кризових викликів. Безпрецедентне зростання обсягів допомоги та стрімке розширення масштабів волонтерського руху в Україні зумовлюють об'єктивну потребу в розробці та впровадженні консолідованих, прозорих і безпечних інформаційних систем для управління потоками даних у реальному часі.

Таким чином, виникає гостра науково-практична потреба у переході від аматорських засобів комунікації до створення профільного, безпечного та масштабованого цифрового простору. Розробка спеціалізованої веб-платформи «UNITY+» спрямована на автоматизацію процесів інформаційного обміну між отримувачами допомоги, індивідуальними волонтерами та благодійними фондами з використанням сучасних парадигм веб-розробки й хмарних BaaS-технологій.

Методи дослідження. Для вирішення поставлених у кваліфікаційній роботі інженерних задач було використано комплекс методів системного аналізу предметної області, принципи об’єктно-орієнтованого аналізу та проектування програмних систем, методології структурного аналізу (SADT, DFD), інструменти візуального моделювання системної архітектури за допомогою уніфікованої мови

моделювання UML, методи проектування реляційних структур баз даних, а також методи компонентного тестування та експертної оцінки UX/UI інтерфейсів.

Наукова новизна та практична цінність результатів. Практична цінність отриманих результатів полягає у створенні готового до розгортання та експлуатації програмного комплексу волонтерської координації «UNITY+». Спроектована клієнт-серверна архітектура дозволяє консолідувати запити на допомогу в межах єдиного SPA-додатка, інтегрувати засоби інтерактивної картографії для візуалізації гуманітарних потреб, оптимізувати швидкість пошуку релевантних ініціатив за рахунок SQL-подібної фільтрації та забезпечити високий рівень безпеки й конфіденційності даних користувачів завдяки ізоляції серверних сесій у хмарній інфраструктурі.

Об’єктом дослідження є процес інформаційної взаємодії та координації дій між волонтерами, некомерційними організаціями, благодійними фондами та отримувачами гуманітарної допомоги.

Предметом дослідження є моделі, методи, архітектурні рішення, структури даних та програмні інструменти розробки клієнт-серверної веб-платформи координації волонтерської діяльності.

Метою дослідження є підвищення ефективності, прозорості та безпеки координації волонтерського руху шляхом проектування, розробки та функціонального тестування веб-платформи «UNITY+» на основі компонентного підходу бібліотеки React та хмарної інфраструктури BaaS-платформи Appwrite.

Для досягнення цієї мети було вирішено такі завдання:

–огляд і аналіз сучасного стану автоматизації волонтерської діяльності та виявлення деструктивних факторів у поточних процесах комунікації стейкхолдерів;

– порівняльний аналіз функціональних можливостей існуючих програмних аналогів та автоматизованих систем на ринку;

– обґрунтування вибору клієнт-серверної архітектури та технологічного стеку для реалізації проекту (React, Appwrite, JavaScript);

– формулювання повного комплексу функціональних і нефункціональних вимог до системи та побудова процесних моделей за допомогою методології UML;

– проектування логічної та фізичної структури бази даних (колекцій хмарного сховища) з урахуванням реляційних зв'язків та моделей безпеки користувачів;

– програмна реалізація клієнтської частини додатка та інтеграція хмарних сервісів автентифікації, збереження даних та роботи з мапами.

РОЗДІЛ 1
ОБГРУНТУВАННЯ ІТ-ПРОЕКТУ

1.1 Аналіз сучасного стану та проблем координації волонтерської діяльності

Актуальність розробки та впровадження уніфікованого ІТ-рішення у сфері волонтерської діяльності в Україні є надзвичайно високою, що зумовлено необхідністю консолідації зусиль громадянського суспільства у кризові періоди та під час повоєнного відновлення. Сучасний етап розвитку цієї предметної області характеризується стрімким збільшенням обсягів інформаційних потоків, розширенням кола учасників гуманітарних ініціатив та виникненням нових вимог до швидкості та безпеки обміну даними.

Для побудови ефективної архітектури програмного забезпечення платформи «UNITY+» необхідно виконати детальний аналіз сутності, об'єктів та ключових процесів, що відбуваються у досліджуваній предметній області.

Предметна область проєкту охоплює інформаційні, організаційні та комунікаційні процеси взаємодії між волонтерами, некомерційними організаціями (НУО), благодійними фондами, донорами та державними структурами в цифровому середовищі. Межі предметної області обмежені процедурами реєстрації, верифікації, публікації гуманітарних чи матеріальних потреб, координації сумісних дій та моніторингу результатів виконання завдань.

Основними об'єктами автоматизації в межах розроблюваної системи є:

– Волонтерські ресурси: сукупність даних про користувачів, їхні специфічні навички (програмування, дизайн, SMM, логістика, медицина), географічне розташування та тимчасовий бюджет (вільний час);

– Волонтерські ініціативи та проєкти: складні структуровані об'єкти, які мають власний життєвий цикл (ідея, модерація, активний збір/реалізація, завершення, публічний звіт);

– Завдання (атоми робіт): мінімальні операційні одиниці діяльності всередині великих проєктів, які відображаються на Канбан-дошках, мають конкретних відповідальних виконавців, статуси виконання та часові обмеження;

– Соціальний капітал: гейміфікована підсистема обліку репутації, верифікованих досягнень, накопичених волонтерських годин та цифрових відзнак (бейджів).

Динаміка предметної області визначається базовими інформаційними процесами, до яких належать:

– Рекрутинг та онбординг: залучення нових учасників до системи, створення профілів та обов'язкова перевірка їх легітимності й доброчесності (верифікація громадських організацій за реєстраційними даними);

– Координація та оперативне управління: розподіл поточних завдань за допомогою інструментів візуалізації, встановлення пріоритетів та дедлайнів;

– Моніторинг та контроль: відстеження поточного статусу виконання завдань у реальному часі та автоматична фіксація відпрацьованого часу користувачів;

– Внутрішньосистемна комунікація: обмін інформаційними повідомленнями між учасниками ініціатив безпосередньо всередині веб-платформи;

– Аналітика та генерація звітності: автоматизоване формування звітних даних для донорів, державних органів та суспільства про обсяги та ефективність використання залучених ресурсів.

Системний аналіз сучасного стану предметної області дозволив виявити три ключові проблеми, що вимагають негайного інженерного та програмного вирішення:

– *Проблема фрагментованої комунікації та децентралізації даних.* Наразі більшість процесів координації здійснюється через розрізнені неформальні канали — групові чати у месенджерах (Telegram, Viber) та публікації у соціальних мережах. Це унеможливлює ведення єдиного централізованого обліку запитів,

призводить до швидкої втрати актуальності повідомлень у загальному потоці, викликає хаос та дублювання однакових завдань різними фондами.

– *Проблема прозорості, безпеки та довіри.* Брак інтегрованих засобів автоматизованої перевірки призводить до появи шахрайських зборів, що підриває довгострокову довіру суспільства до волонтерського сектору. Відсутність єдиної системи фіксації волонтерського внеску ускладнює підготовку достовірної звітності для міжнародних донорів.

– *Проблема нецільового та неефективного рекрутингу.* Благодійні організації витрачають значний часовий ресурс на ручний пошук виконавців під специфічні задачі. У той же час потенційні волонтери не мають зручного інструментарію для швидкого пошуку ініціатив, які б точно відповідали їхньому профілю навичок та поточній геолокації. Додатковим деструктивним фактором є відсутність системи довгострокової мотивації, що викликає швидке емоційне вигорання учасників руху.

Рішенням зазначених проблем є розробка веб-платформи «UNITY+», яка консолідує інформаційні процеси в межах єдиного архітектурного простору. Схема учасників (стейкхолдерів) та користувачів, залучених до інформаційної взаємодії з платформою, відображає архітектурні ролі системи. До структури учасників входять:

– *Індивідуальні волонтери:* кінцеві користувачі, які використовують систему для пошуку проєктів за фільтрами навичок, ведення особистого профілю-портфоліо та трекінгу власної активності;

– *Волонтерські та громадські організації (НУО, благодійні фонди):* користувачі з правами менеджерів, які здійснюють публікацію проєктів, декомпозицію завдань на Канбан-дошках та верифікацію внеску волонтерів;

– Адміністратори платформи: системні користувачі, відповідальні за модерацію контенту, технічний супровід та верифікацію юридичного статусу організацій за кодами ЄДРПОУ;

– *Фінансові стейкхолдери та міжнародні донори:* суб'єкти, які отримують доступ до модулів автоматизованої публічної аналітики та звітності для контролю цільового використання виділених грантів;

– *Державні та регулюючі органи:* Міністерство цифрової трансформації та Міністерство соціальної політики, які виступають потенційними партнерами для подальшої інтеграції платформи з державними реєстрами та сервісами.

Взаємозв'язки та вектори інформаційного обміну між усіма учасниками та сервісами розроблюваного програмного забезпечення деталізовано на рисунку 1.1.

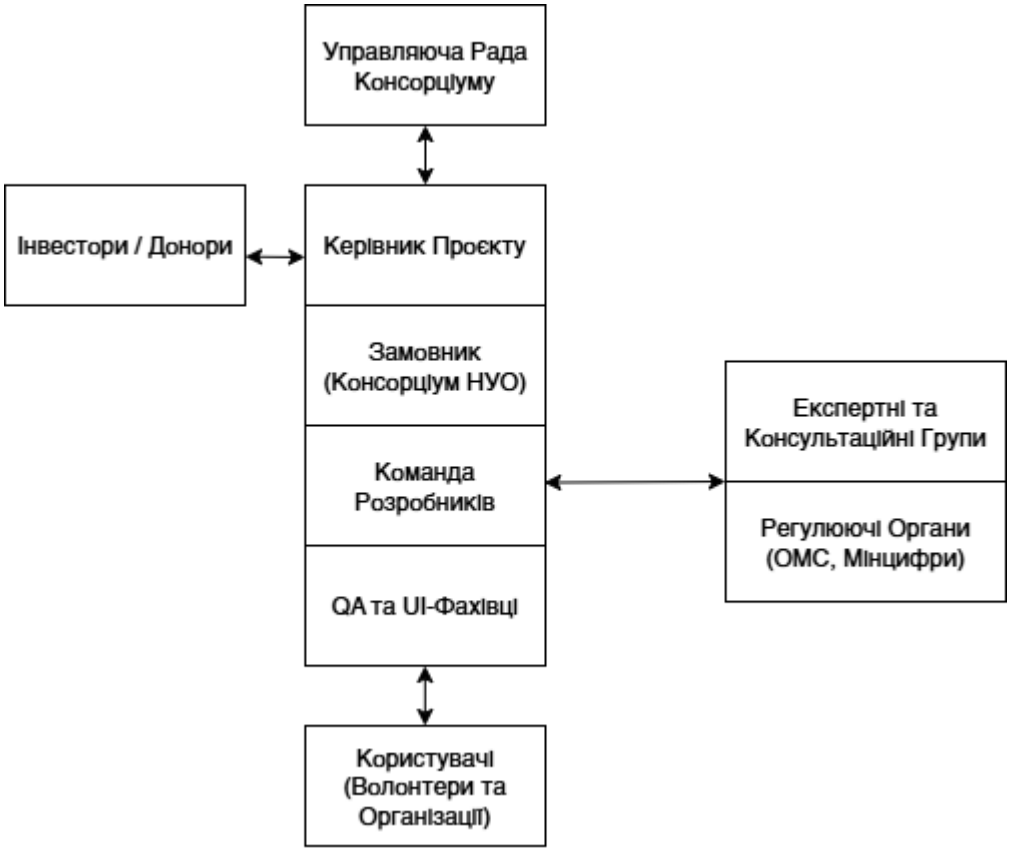

Рисунок 1.1 - Схема стейкхолдерів та учасників взаємодії веб-платформи UNITY+.

Таким чином, розробка та впровадження платформи «UNITY+» заповнює існуючі технологічні прогалини у предметній області, створюючи захищене та гнучке ІТ-рішення для ефективної структуризації та автоматизації процесів взаємодії громадянського суспільства.

1.2 Порівняльний аналіз існуючих аналогів та автоматизованих систем

Для обґрунтування актуальності розробки та доведення техніко-економічної доцільності проєктування веб-платформи «UNITY+» необхідно провести комплексний критичний аналіз наявних альтернативних інформаційних систем та технічних рішень на ринку України та світу. Головна мета аналізу полягає у виявленні функціональних та архітектурних недоліків існуючого програмного забезпечення та підтвердженні того, що жоден із доступних аналогів не здатен задовольнити комплексну потребу автоматизації волонтерського сектору.

Сучасний ринок програмного забезпечення пропонує кілька категорій систем, які частково використовуються волонтерськими організаціями (НУО) та благодійними фондами для автоматизації їхньої діяльності. Проте всі вони мають вузьку спеціалізацію та обмежену архітектуру. Розглянемо детально основні класи таких систем.

– *Спеціалізовані інформаційні платформи пошуку та оголошень (наприклад, «Волонтерська Платформа», «Helplist»).*

Ці веб-ресурси розроблені переважно як централізовані дошки оголошень, де організації можуть публікувати текстові запити про допомогу або відкриті волонтерські вакансії, а користувачі – реєструватися на конкретні події.

Технічні та функціональні обмеження: Основний архітектурний недолік таких платформ полягає у повній відсутності внутрішнього інструментарію для управління життєвим циклом проєкту після етапу рекрутингу. Організатор не має можливості декомпонувати велику гуманітарну місію на окремі операційні атоми (завдання), розподіляти їх між волонтерами всередині платформи та відстежувати прогрес у реальному часі за допомогою Канбан-дошок. Крім того, у цих системах відсутній автоматизований облік відпрацьованих волонтерських годин за часовими мітками (timestamps) та інтегрована система цифрової мотивації (гейміфікації).

– *Комерційні та адаптовані CRM-системи (наприклад, NetHunt CRM, KeepinCRM, Salesforce для НУО).*

Багато великих благодійних фондів намагаються адаптувати готові комерційні CRM-рішення під власні потреби. Ці системи фокусуються на фінансовому обліку, веденні бази даних великих донорів (fundraising), трекінгу грантових коштів та генерації офіційної фінансової звітності.

Технічні та функціональні обмеження: Головним мінусом CRM-систем є повна відсутність орієнтації на кінцевого користувача – безпосереднього волонтера. Вони не містять інструментів для створення індивідуальних публічних профілів-портфоліо волонтерів, де фіксувався б їхній «цифровий слід», перелік підтверджених навичок, історія внесків та система репутаційних відзнак (бейджів). Подібні платформи є закритими внутрішньокорпоративними системами, що унеможливлює створення на їхній базі відкритого соціального простору для взаємодії громадянського суспільства. Також висока вартість ліцензій комерційного ПЗ робить їх недоступними для середніх та малих волонтерських хабів.

– *Універсальні хмарні інструменти управління проєктами та таск-менеджери (наприклад, Trello, Asana, Jira, Google Sheets).*

Через брак профільних рішень, більшість локальних волонтерських об'єднань організовують операційную діяльність за допомогою загальних систем управління задачами. Trello або Asana дозволяють створювати картки завдань, переміщувати їх між етапами виконання та призначати відповідальних осіб.

Технічні та функціональні обмеження: Дані інструменти вимагають складної ручної синхронізації та адміністрування, оскільки вони не інтегровані з публічними реєстрами. У них немає механізмів автоматичної верифікації легітимності благодійних фондів (наприклад, через перевірку коду ЄДРПОУ), що є критичним фактором безпеки в сучасних умовах. Вони не забезпечують автоматичну генерацію публічної звітності для суспільства та міжнародних партнерів і не мають інструментів багатофакторної фільтрації волонтерських вакансій за специфічними навичками (SMM, дизайн, логістика, медицина) у поєднанні з геолокаційними мітками на інтерактивній мапі.

Для проведення детального порівняльного аналізу технічних, системних та функціональних можливостей розглянутих класів інформаційних систем із проектованою веб-платформою «UNITY+» було сформовано детермінований перелік критеріїв оцінювання. Результати аналізу зведено у таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз техніко-функціональних параметрів інформаційних систем координації волонтерської діяльності.

Критерії порівняння та оцінювання систем	Спеціалізовані ІС	Комерційні CRM	Загальні таск-менеджери	Проектована веб-платформа «UNITY+»
Архітектурний тип програмного комплексу	Багатосторінковий веб-ресурс (MPA)	Закрита хмарна SaaS-інфраструктура	Універсальний хмарний сервіс	Високотехнологічний SPA-додаток
Централізований таск-менеджмент (Канбан)	Відсутній (лише текстові оголошення)	Відсутній (орієнтація на клієнтські угоди)	Наявний (базові універсальні дошки задач)	Інтегрований спеціалізований Канбан-модуль
Автоматизована верифікація організації	Часткова (ручна перевірка модератором)	Відсутня (введення даних користувачем)	Повністю відсутня (анонімне створення)	Інтегрована верифікація (ЄДРПОУ / ліцензії)
Геоінформаційна прив'язка запитів	Текстове вказання міста / регіону	Відсутня	Відсутня	Інтерактивна картографія (мапа потреб)
Гейміфікація та мотивація (Бейджі)	Відсутня	Відсутня	Відсутня	Наявна підсистема соціального капіталу
Публічне волонтерське портфоліо	Відсутнє (лише внутрішній профіль)	Відсутнє	Відсутнє	Наявне відкрите цифрове портфоліо навичок

Продовження табл. 1.1

Автоматизований звіт для донорів	Відсутній	Наявний (фінансовий аналіз)	Відсутній (потребує ручного експорту)	Автоматична генерація публічної аналітики
Вартість розгортання та підтримки	Безкоштовно (обмежений функціонал)	Висока (щомісячна платна підписка)	Платні тарифи для великих команд	Безкоштовне хмарне BaaS-рішення

Аналіз даних, що наведені у таблиці 1.1, дозволяє зробити висновок, що жодна з існуючих інформаційних систем не забезпечує повного циклу автоматизації волонтерської діяльності. Комерційні системи є занадто закритими та коштовними для некомерційного сектору, а існуючі безкоштовні вітчизняні платформи виконують виключно пасивну інформаційну роль, залишаючи весь оперативний менеджмент, облік та комунікацію за рамками програмного продукту.

Таким чином, розроблюваним проєктом «UNITY+» повністю заповнюється існуюча прогалина на ринку IT-рішень. Платформою пропонується унікальна архітектурна концепція, що забезпечує двоспрямовану структуризацію даних. В єдиній системі, реалізованій як захищений високотехнологічний SPA-додаток, успішно поєднуються функції управління змістом волонтерських ініціатив, верифікації та автоматизованого обліку особистого внеску користувачів.

1.3 Обґрунтування вибору архітектури та технологічного стеку

Під час проектування високонавантажених інформаційних систем, орієнтованих на роботу в реальному часі з великою кількістю динамічних запитів, критично важливим є вибір оптимальної архітектурної парадигми. Для розробки волонтерської платформи «UNITY+» було обрано архітектуру SPA (Single Page Application — односторінковий веб-додаток) із чітким розділенням на клієнтську (Frontend) та серверну (Backend) частини, що взаємодіють через захищений протокол передачі даних за допомогою REST API.

Вибір SPA зумовлений необхідністю забезпечення високої швидкодії та безшовного UX/UI інтерфейсу для кінцевих користувачів. На відміну від класичних багатосторінкових додатків (MPA), де кожна дія викликає повне перезавантаження сторінки з сервера, SPA завантажує базовий HTML-код, стилі та скрипти лише один раз. Усі подальші динамічні оновлення даних (наприклад, фільтрація карток потреб чи трекінг статусів на Канбан-дошці) відбуваються шляхом асинхронних запитів у фоновому режимі, що критично знижує навантаження на мережу та забезпечує миттєвий відгук інтерфейсу додатку.

Для реалізації логіки клієнтської та серверної частин у межах єдиної екосистеми було обрано мову програмування JavaScript (стандарт ECMAScript). Вибір цієї мови обґрунтовано її асинхронною подійно-орієнтованою моделлю виконання, що базується на циклі подій (Event Loop). Це дозволяє ефективно обробляти велику кількість паралельних запитів до бази даних без блокування основного потоку виконання (I/O non-blocking інпут/аутпут). Використання єдиного стеку технологій (Single Language Stack) суттєво спрощує інтеграцію між модулями, оптимізує швидкість розробки ПЗ та полегшує супровід програмного коду системи.

Клієнтську частину системи (Frontend) реалізовано на базі компонентного підходу з використанням open-source JavaScript-бібліотеки React. Вибір даного інструменту обґрунтовано такими інженерними перевагами:

– Virtual DOM (Віртуальне дерево елементів): React створює полегшену копію реального DOM у пам'яті додатка. Під час зміни стану (state) компонентів система обчуслює мінімальну різницю (diffing) за допомогою алгоритму узгодження і оновлює лише ті елементи сторінки, які реально зазнали змін. Це гарантує стабільну роботу інтерактивної мапи та Канбан-дошки навіть за умов низької швидкості інтернет-з'єднання.

– Компонентно-орієнтована архітектурна модель: Весь веб-додаток декомпонується на незалежні, ізольовані модулі багаторазового використання (наприклад, CardNeed, Navbar, AuthForm, KanbanBoard). Такий підхід значно

спрощує масштабування системи, полегшує процес модульного тестування та дозволяє реалізувати чистий, адаптивний UI/UX дизайн.

– Декларативний підхід до розробки: Розробник описує фінальний стан і вигляд інтерфейсу користувача для кожного конкретного режиму системи, а React самостійно та оптимізовано керує процесом його відображення (рендерингу), що суттєво мінімізує виникнення логічних помилок у програмному коді клієнта.

Як серверну інфраструктуру (Backend) та хмарну систему управління базами даних (СУБД) обрано прогресивну BaaS-платформу (Backend-as-a-Service) Appwrite. Використання готового хмарного бекенд-сервісу замість написання власного серверного коду з нуля (наприклад, на Node.js чи Django) дозволяє оптимізувати архітектуру платформи «UNITY+» за рахунок таких технічних факторів:

– Інтегрований модуль автентифікації (Appwrite Auth): Забезпечує готові безпечні протоколи реєстрації, авторизації та верифікації сесій користувачів з підтримкою шифрування токенів, технології JWT (JSON Web Tokens) та інтеграції зовнішніх провайдерів OAuth.

– Документо-орієнтована хмарна СУБД (Appwrite Databases): Дозволяє проектувати гнучкі колекції даних (Users, Needs, Tasks) з можливістю динамічного розширення атрибутів, налаштуванням вбудованих індексів для прискорення пошуку та валідації типів даних безпосередньо на рівні сервера.

– Хмарне сховище (Appwrite Storage): Надає оптимізовані інструменти для завантаження, безпечного збереження та швидкої дистрибуції медіафайлів, первинних звітів, кошторисів та фотозвітів волонтерських організацій з автоматичним стисненням зображень на стороні хмари.

– Безпека та ізоляція даних на рівні документів: Платформа підтримує вбудовану систему розподілу прав доступу (Permissions) на рівні окремих записів у колекціях (наприклад, редагувати або видаляти картку потреби може лише той користувач або організація, яка виступає її безпосереднім автором), що повністю задовольняє вимоги надійності та конфіденційності.

Взаємодія між Frontend-частиною (додаток на React) та Backend-інфраструктурою (хмара Appwrite) реалізується за допомогою офіційного інструменту Appwrite Web SDK. Клієнтська частина надсилає асинхронні HTTP-запити (за допомогою методів GET, POST, PUT, DELETE) через REST API. Сервер обробляє ці запити і повертає структуровані дані у форматі JSON. Загальну концептуальну схему клієнт-серверної взаємодії розроблюваного програмного забезпечення наведено на рисунку 1.2.

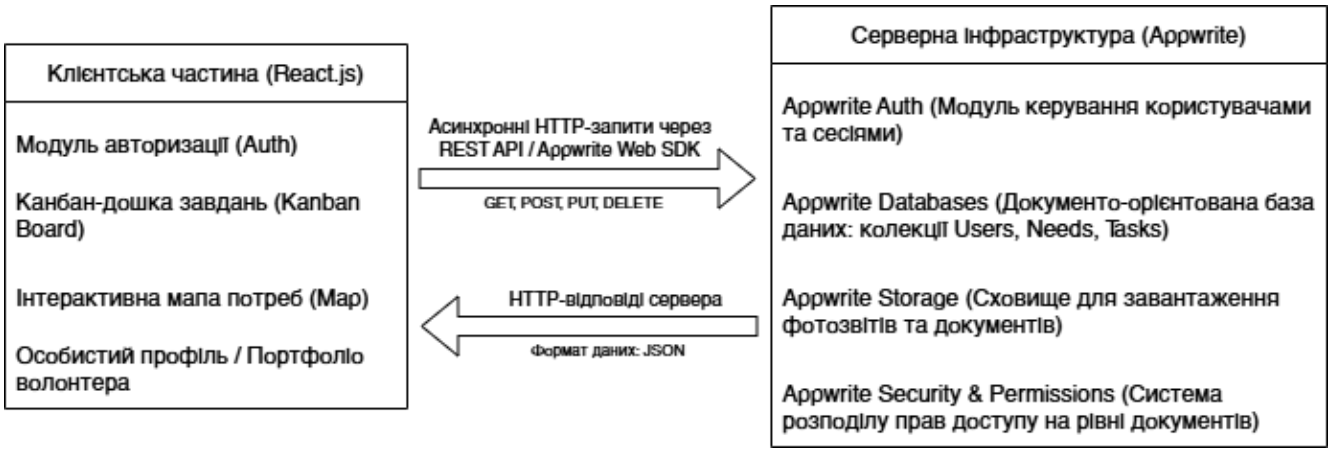

Рисунок 1.2 – Архітектурна схема клієнт-серверної взаємодії платформи UNITY+.

Поєднання технологій React та Appwrite забезпечує створення сучасного, гнучкого, безпечного та стійкого до високих навантажень IT-рішення, здатного підтримувати стабільну роботу з тисячами користувачів одночасно без втрати продуктивності інтерфейсу.

Висновки до розділу:

У першому розділі проведено комплексний системний аналіз сучасного стану автоматизації волонтерського руху, визначено ключові проблеми координації стейкхолдерів та виконано теоретичне й системне обґрунтування концепції IT-проєкту. Розглянуто особливості побудови структури та схеми учасників волонтерської діяльності, здійснено детальний порівняльний аналіз техніко-функціональних параметрів існуючих програмних аналогів. На основі виявлених

архітектурних обмежень існуючого ПЗ було науково обґрунтовано вибір клієнт-серверної архітектури односторінкового додатка (SPA) на базі компонентного підходу бібліотеки React, мови JavaScript та хмарної інфраструктури BaaS-платформи Appwrite як оптимального технологічного стеку для реалізації веб-платформи «UNITY+».

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА ТЕХНІЧНА РЕАЛІЗАЦІЯ ПЛАТФОРМИ «UNITY+»

2.1 Формулювання вимог до веб-платформи та системний аналіз

Проєктування складної клієнт-серверної системи координації волонтерської діяльності «UNITY+» потребує попереднього чіткого структурування вимог за класичною методологією системного аналізу. Для цього специфікацію вимог було розподілено на чотири ієрархічні рівні: бізнес-вимоги, вимоги користувачів, функціональні та нефункціональні вимоги, а також визначено ключові обмеження й технічні ризики проєкту.

Бізнес-вимоги (Business Requirements):

– Створення єдиного консолідованого цифрового простору для ефективної взаємодії та координації зусиль інститутів громадянського суспільства в Україні;

– Забезпечення високого рівня прозорості та інформаційної безпеки волонтерської діяльності для підвищення довіри з боку суспільства та міжнародних донорів;

– Оптимізація рекрутингових процесів із метою зменшення середнього часу закриття критичної вакансії волонтера на 10% протягом першого року експлуатації системи.

Вимоги користувачів (User Requirements):

– Роль «Волонтер»: можливість швидкого пошуку релевантних проєктів за профілем навичок та геолокацією, перегляд особистої статистики внесків та динамічного портфоліо;

– Роль «Організатор» (представники НУО та фондів): можливість реєстрації організації, публікації волонтерських ініціатив, декомпозиції проєктів на конкретні завдання, менеджменту заявок та верифікації відпрацьованого часу;

– Роль «Адміністратор платформи»: доступ до засобів системної модерації контенту та інструментів верифікації юридичного статусу організацій.

Функціональні вимоги (Functional Requirements):

– Надання двох незалежних та розгалужених шляхів реєстрації та автентифікації користувачів залежно від обраної ролі;

– Реалізація інтерактивного модуля Канбан-дошки для оперативного контролю та візуалізації життєвого циклу завдань у реальному часі;

– Автоматизований облік відпрацьованих волонтерських годин на основі системних часових міток (timestamps);

– Формування динамічного персонального портфоліо волонтера з гейміфікованою системою нарахування цифрових нагород (бейджів);

– Реалізація модуля багатофакторної фільтрації та пошуку ініціатив за категоріями, тегами компетенцій та географічними координатами;

– Інтеграція засобів завантаження та збереження фотозвітів і супровідної робочої документації (кошторисів, планів робіт).

Нефункціональні вимоги (Non-Functional Requirements):

– Технологічність: розробка системи на базі сучасного веб-стеку з використанням бібліотеки React для фронтенду та готових хмарних сервісів Appwrite SDK для бекенду;

– Масштабованість: архітектура бази даних (колекцій) повинна підтримувати стабільну обробку транзакцій та збереження даних для понад 15 000 активних користувачів;

– Надійність: забезпечення високої доступності (High Availability) системи за рахунок розгортання в оптимізованій хмарній інфраструктурі;

– Інтерфейс: реалізація адаптивного, кросбраузерного та інтуїтивно зрозумілого UI/UX дизайну для десктопних та мобільних пристроїв.

Обмеження та ризики проєкту:

– Часове обмеження: повне завершення основної фази проектування, кодування та функціонального тестування — до кінця 2026 року;

– Ресурсно-інформаційне обмеження: використання виключно офіційно верифікованих даних некомерційних організацій.

Для оптимізації процесу розробки програмного забезпечення на основі системного аналізу було проведено пріоритезацію функціональних вимог за допомогою експертних оцінок. Результати пріоритезації наведено у таблиці 2.1.

Таблиця 2.1 – Пріоритезація функціональних вимог до платформи «UNITY+»

Функціональна вимога	Опис	Пріоритет
Реєстрація та верифікація	Розмежування ролей волонтера та організації з перевіркою документів.	Високий
Керування проєктами	Можливість створення ініціативи та розбиття її на завдання.	Високий
Пошукова система	Багатофакторний фільтр для знаходження релевантних запитів.	Середній
Облік та мотивація	Автоматична фіксація годин та видача цифрових бейджів.	Середній
Публічна звітність	Генерація статистики для донорів та суспільства	Низький

Первинні результати системного аналізу, які виходять за межі безпосереднього коду, включають формування єдиного візуального бренду (назва, логотип UNITY+), розробку детальної архітектурної документації та проектування користувацьких інтерфейсів (UI/UX).

З метою верифікації сформульованих вимог та створення оптимального середовища взаємодії користувачів на етапі системного аналізу було розроблено високоточні графічні прототипи екранних форм (інтерфейсів) платформи у середовищі Figma. Повний комплекс розроблених графічних макетів представлено у ДОДАТКУ А. Нижче наведено детальний інженерний аналіз спроектованої структури користувацького інтерфейсу системи:

– *Головна сторінка інформаційної системи.* Виконує функцію агрегації публічних даних, представляючи користувачу загальну динамічну статистику внесків (години, проєкти, волонтери), та слугує основним входом до системи. Інтегрована навігаційна панель надає доступ до реєстрації, пошуку проєктів та сторінки профілю. Макет інтерфейсу наведено у ДОДАТКУ А (рисунок А.1).

– *Система реєстрації та авторизації.* Проєктується два окремих шляхи для реєстрації з метою розмежування прав доступу. Реєстрація волонтера орієнтована на персональні дані користувача (ПІБ, телефон, email), тоді як реєстрація організації вимагає введення офіційної назви та реєстраційного коду ЄДРПОУ/ІНН для подальшої верифікації. Систему авторизації також розмежовано. Спроектовані екранні форми представлено у ДОДАТКУ А (рисунки А.2–А.5).

– *Профіль волонтера.* Профіль є центральним елементом для обліку внеску та мотивації й виконує роль цифрового волонтерського портфоліо. Він відображає повний облік активності (завершені проєкти, кількість відпрацьованих годин, історію отриманих бейджів), містить динамічний розділ «Мої навички» та список «Активні завдання» з можливістю підтвердження зміни статусів виконання. Графічний прототип профілю наведено у ДОДАТКУ А (рисунок А.6).

– *Профіль організації.* Слугує адміністративною панеллю (дашбордом) для управління проєктами та ресурсами. Надає загальну аналітику залучення волонтерів, відображає внутрішній рейтинг фонду та містить інтегрований модуль «Нові заявки волонтерів» з функцією швидкого прийняття або відхилення кандидатів. Прототип дашборду наведено у ДОДАТКУ А (рисунок А.7).

– *Створення нового проєкту.* Сторінка реалізована у вигляді покрокової форми, що дозволяє організатору визначити всі вихідні параметри ініціативи: загальні відомості, вимоги та обмеження до компетенцій волонтерів, географічні або часові ліміти, а також виконати декомпозицію проєкту на окремі вимірювані завдання. Макет екранної форми наведено у ДОДАТКУ А (рисунок А.8).

– *Адміністрування та управління проєктом.* Адміністративний інтерфейс, призначений для контролю, координації та моніторингу активної ініціативи в реальному часі. Надає інструменти для керування складом команди, редагування завдань, зміни їхнього поточного статусу («Виконано», «В процесі», «Очікує»), верифікації відпрацьованих годин та завантаження супровідних документів. Прототип панелі наведено у ДОДАТКУ А (рисунок А.9).

– *Сторінка проєкту.* Інформаційний хаб, що надає волонтеру повні відомості для прийняття рішення про участь у проєкті. Відображає назву, організатора, загальний прогрес, мету, інформацію про локацію на інтерактивній карті, дедлайни, перелік специфічних вимог та доступних атомарних завдань із можливістю миттєвого підтвердження участі. Графічний макет сторінки наведено у ДОДАТКУ А (рисунок А.10).

– *Пошук проєкту.* Ключовий інструмент для швидкого рекрутингу та координації ресурсів. Сторінка інтегрує багатофакторний фільтр пошуку за категоріями, навичками та містами, динамічні теги для швидкого переходу до актуальних напрямків, інструменти сортування та інформативні картки результатів із мітками терміновості запитів. Графічний макет пошукового модуля наведено у ДОДАТКУ А (рисунок А.11).

2.2 Функціональне та процесне моделювання системи

Проєктний етап розробки архітектури веб-платформи «UNITY+» передбачає побудову комплексу взаємопов'язаних функціональних та процесних моделей. Це дозволяє абстрагуватися від конкретної програмної реалізації, виконати декомпозицію складних бізнес-процесів волонтерського руху та зафіксувати вектори трансформації інформаційних потоків між клієнтською та серверною частинами системи. Моделювання реалізовано за допомогою поєднання методологій структурного аналізу (SADT, DFD) та уніфікованої мови об'єктно-орієнтованого моделювання UML.

2.2.1 Функціональне та процесне моделювання системи

Для визначення ієрархічної структури функцій системи та правил їхнього керування було застосовано методологію структурного аналізу SADT (Structured Analysis and Design Technique) у стандарті IDEF0. На концептуальному рівні (контекстна діаграма A0) веб-платформа розглядається як єдиний функціональний блок, що трансформує вхідні запити користувачів у верифіковані соціальні результати.

З метою деталізації архітектури було виконано функціональну декомпозицію першого рівня (діаграма A0), в межах якої виділено п'ять взаємопов'язаних функціональних блоків (активностей):

A1: Реєстрація та верифікація. Процес забезпечує первинний онбординг стейкхолдерів та розмежування прав доступу на основі ролевої моделі.

– Вхід (I): реєстраційні анкети користувача (ПІБ, контактні дані, Email) та офіційні дані некомерційних організацій (назва, код ЄДРПОУ, статутні документи).

– Управління (C): алгоритми програмної перевірки валідності коду ЄДРПОУ, внутрішня політика конфіденційності платформи та правила верифікації облікових записів.

– Вихід (O): активований та верифікований профіль волонтера або підтверджений статус організації з правом публікації зборів.

– Механізм (M): адміністратор системи (модератор) та програмний модуль автентифікації на базі хмарних сервісів інфраструктури.

A2: Пошук та підбір ініціатив. Модуль призначений для забезпечення інтелектуального зведення потреб та наявних людських ресурсів.

– Вхід (I): перелік заявлених навичок волонтера, поточна геолокація користувача та багатофакторні фільтри пошуку (категорія проєкту, терміновість).

– Управління (C): атомарна база навичок (Taxonomy / теги компетенцій) та наявні обмеження проєкту (дедлайни, локація, ліміт учасників).

– Вихід (O): персоналізований список релевантних волонтерських проєктів та сформовані цифрові заявки на участь.

– Механізм (M): багатофакторний пошуковий фільтр та вбудовані алгоритми сортування результатів вибірки за коефіцієнтом релевантності.

A3: Управління волонтерськими діями. Центральний операційний блок, що автоматизує безпосередній процес виконання гуманітарних місій.

– Вхід (I): призначення волонтера на конкретну задачу всередині проєкту та вхідні робочі документи (плани робіт, інструкції, кошториси).

– Управління (C): внутрішні регламенти виконання робіт, встановлені дедлайни завдань та методологія візуального менеджменту Канбан.

– Вихід (O): динамічна зміна статусу завдання на дошці («Очікує», «В процесі», «Виконано») та готові результати роботи (документальні та фотозвіти).

– Механізм (M): програмний модуль «Канбан-дошка» та інтегрований інтерфейс адміністративної панелі організатора.

A4: Облік внеску та годин. Функція відповідає за математично точну та прозору фіксацію витраченого часу.

– Вхід (I): системні часові мітки (timestamps) про моменти початку та фактичного завершення волонтером активності за завданням.

– Управління (C): правила та критерії підтвердження робочого часу верифікованою некомерційною організацією.

– Вихід (O): накопичені та верифіковані години в персональному цифровому портфоліо волонтера.

– Механізм (M): серверне API для фіксації часу та розподілена хмарна база даних.

A5: Гейміфікація та звітність. Підсистема довгострокової мотивації та зовнішнього моніторингу результатів.

– Вхід (I): виконані умови для системних досягнень (метрики кількості годин/проєктів) та загальна накопичена статистика активності.

– Управління (C): набір логічних правил та тригерів («досягнення») для автоматичного розблокування унікальних нагород.

– Вихід (O): цифрові репутаційні бейджі, оновлене публічне портфоліо волонтера та консолідована аналітична звітність для міжнародних донорів.

– Механізм (M): внутрішній сервіс гейміфікації та генератор динамічних звітів.

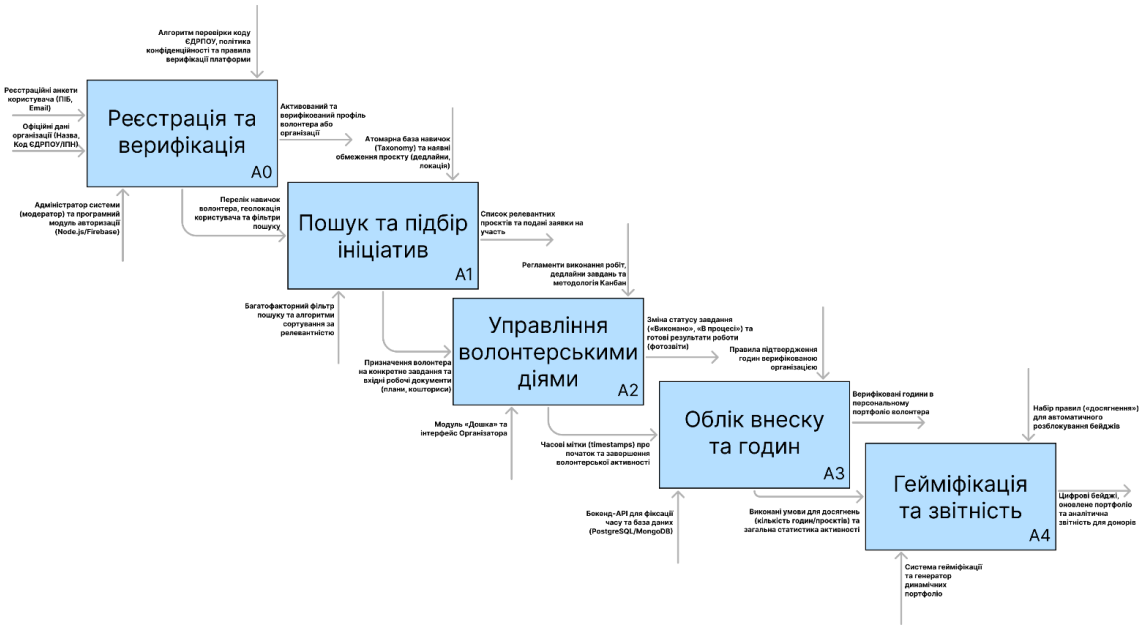

Рисунок 2.2 – Діаграма функціональної декомпозиції IDEF0 першого рівня платформи UNITY+.

2.2.2 Моделювання потоків даних (DFD)

Для аналізу процесів перетворення інформації та визначення місць її збереження було розроблено діаграму потоків даних DFD (Data Flow Diagram). Модель чітко розмежовує активні компоненти системи, сховища даних та інформаційні вектори.

Структура розробленої DFD-моделі включає такі інженерні компоненти:

Зовнішні сутності

– Волонтер: виступає первинним джерелом персональних даних і навичок, генерує запити на фільтрацію та отримує вихідні інформаційні структури про доступні проєкти;

– Організація (НУО): постачає в систему описи гуманітарних ініціатив, критерії відбору та отримує автоматизовані аналітичні звіти;

– Адміністратор: отримує системні логи та дані для модерації, видає директиви підтвердження верифікації юридичного статусу.

Накопичувачі даних (Сховища)

– D1 База профілів: документо-орієнтоване сховище, що містить ПІБ, хедшери, Email, токени авторизації, масиви навичок, верифіковані години та особисті рейтинги;

– D2 Реєстр проєктів: зберігає текстові параметри ініціатив, геокоординати для мапи, перелік завдань та їхні поточні статуси на Канбан-дошці;

– D3 Архів відзнак: бібліотека графічних іконок (бейджів) та логічних правил (умов) їхнього автоматичного нарахування.

Процеси (Перетворення даних)

– Процес 1.1 «Авторизувати користувача» (Фізична реалізація: модуль Auth веб-сервера). Трансформує вхідний потік «Реєстраційні дані» від Волонтера/Організації у захищену сесію доступу. Процес надсилає запит «Перевірка доступу» до сховища D1, отримує назад дані «Статус авторизації» та повертає користувачу токен сесії.

– Процес 2.1 «Сформувати волонтерський проєкт» (Фізична реалізація: API-контролер панелі організатора). Приймає від Організації потік «Параметри ініціативи». Після отримання від Адміністратора потоку «Статус модерації», процес трансформує ці дані у потік «Дані нового проєкту» та записує їх у сховище D2.

– Процес 3.1 «Підібрати релевантні ініціативи» (Фізична реалізація: пошуковий SQL-подібний двигун). Обробляє потік від Волонтера «Запит за навичками/локацією», асинхронно зчитує зі сховища D2 дані «Список активних вакансій», виконує їх фільтрацію та повертає Волонтеру потік «Сформована вибірка».

– Процес 4.1 «Верифікувати волонтерський внесок» (Фізична реалізація: бекенд-сервер модуля Канбан). Приймає від Волонтера потік «Звіт про виконання / Часові мітки», а від Організації — потік «Підтвердження/Відхилення». Після логічної обробки процес генерує потік «Оновлені години» та перезаписує їх у сховище D1 для оновлення цифрового портфоліо.

– Процес 5.1 «Нагородити цифровою відзнакою» (Фізична реалізація: сервіс гейміфікації / Cron-завдання). Періодично зчитує зі сховища D1 дані «Поточний прогрес волонтера», порівнює їх із даними «Критерії бейджів» зі сховища D3, і у разі виконання умов направляє Волонтеру потік «Нова відзнака», фіксуючи її в системі.

Архітектурну схему потоків даних системи DFD представлено на рисунку 2.3.

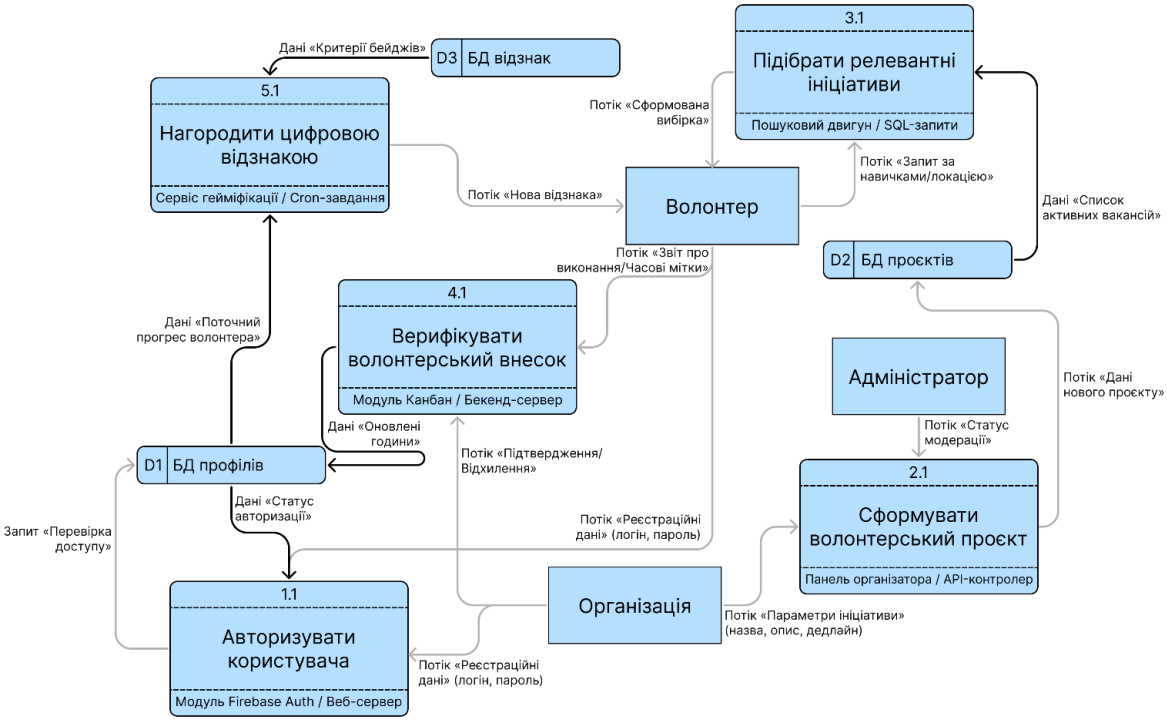

Рисунок 2.3 – Діаграма потоків даних DFD системи координації «UNITY+».

2.2.3 Об'єктно-орієнтоване моделювання за допомогою мови UML

Для формалізації функціональних меж системи та визначення детальних сценаріїв взаємодії компонентів було застосовано інструменти UML-моделювання.

Діаграма прецедентів (Use Case Diagram). Модель визначає повний набір функціональних можливостей платформи, розподілений за трьома основними дійовими особами (акторами). Для деталізації логіки та забезпечення однозначності розуміння меж системи розроблено інженерні специфікації сценаріїв для кожного актора, які наведено у таблицях 2.2, 2.3 та 2.4.

Таблиця 2.2 – Специфікація прецедентів UML для актора «Волонтер»

Use Case	Зв’язок	Опис для UNITY+
Пошук проєкту	—	Базовий функціонал перегляду доступних ініціатив.
Фільтрація за навичками	<<extend>>	Додаткова опція (розширює «Пошук проєкту»), якщо волонтер хоче знайти роботу саме за своїм профілем.
Подати заявку на участь	—	Надсилання запиту організації на приєднання до команди.
Авторизація	<<include>>	Обов'язкова умова (включається в «Подати заявку»), оскільки анонім не може стати волонтером.
Управління завданнями	—	Зміна статусів на канбан-дошці (в процесі / виконано).
Перегляд портфоліо	—	Доступ до особистої статистики та отриманих відзнак.

Таблиця 2.3 – Специфікація прецедентів UML для актора «Організатор»

Use Case	Зв’язок	Опис для UNITY+
Створення ініціативи	—	Додавання нового волонтерського проєкту на платформу.
Визначення вимог	<<include>>	Обов'язковий етап (включається в «Створення ініціативи»): вказання навичок, локації та дедлайну.
Керування заявками	—	Перегляд анкет волонтерів, прийняття або відхилення кандидатів.
Верифікація внеску	—	Підтвердження того, що волонтер дійсно відпрацював заявлені години.
Нарахування бейджів	<<extend>>	Додаткова дія (розширює «Верифікацію»), якщо волонтер виконав умови для нагороди.
Формування звітності	—	Генерація даних про ефективність проєкту для донорів.

Таблиця 2.4 – Специфікація прецедентів UML для актора «Адміністратор»

Use Case	Зв’язок	Опис для UNITY+
Верифікація організації	—	Перевірка документів та коду ЄДРПОУ перед наданням прав на створення проєктів.
Модерація контенту	—	Перевірка описів проєктів на відповідність правилам платформи.
Керування базою знань	—	Оновлення списку навичок (Tags) та категорій волонтерства.
Блокування користувачів	—	Обмеження доступу у разі порушення умов використання платформи.

Графічну діаграму варіантів використання Use Case, що об'єднує описані специфікації, наведено на рисунку 2.4.

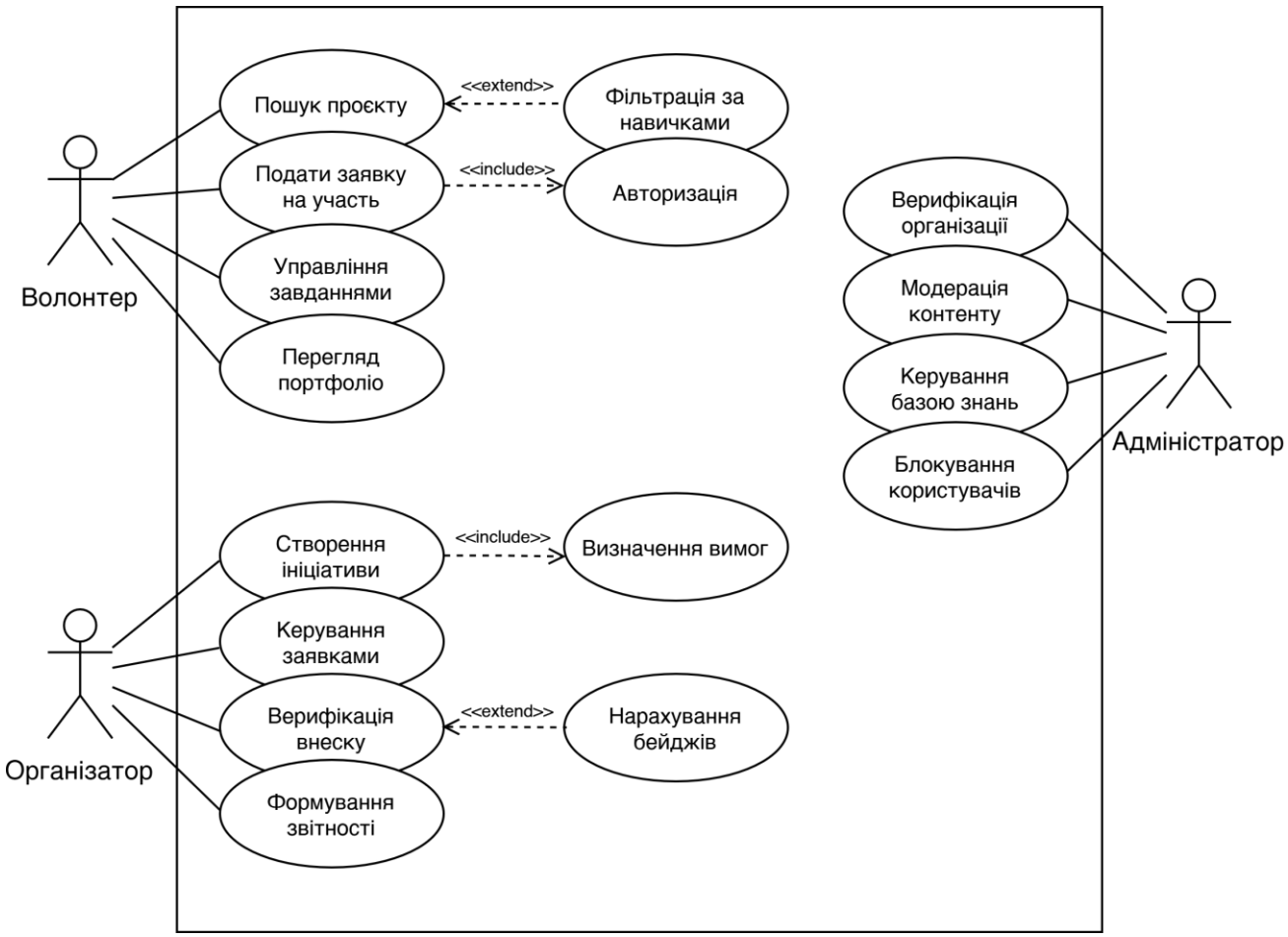

Рисунок 2.4 – UML діаграма прецедентів (Use Case) веб-платформи «UNITY+».

Діаграма послідовності (Sequence Diagram). Для формалізації динаміки системи та деталізації часового виміру взаємодії об'єктів було спроектовано модель процесу подачі заявки волонтером на участь у проєкті. На діаграмі визначено п'ять ключових ліній життя (Lifelines), які розташовані у верхній частині моделі:

– Волонтер (Actor): ініціатор транзакції, кінцевий користувач;

– UI (): фронтенд-компонент інтерфейсу сторінки проєкту, з яким взаємодіє актор;

– API (Controller): контролер серверної логіки, що приймає, валідує та розподіляє запити

– DB (Server): хмарна система управління базами даних, відповідальна за персистентне збереження стану;

– Організатор (Actor): представник благодійної організації, що приймає фінальне рішення.

Взаємодія об'єктів за часовою шкалою зверху вниз відбувається за таким інженерним алгоритмом:

– Етап ініціації сесії: Волонтер ініціює дію натисканням інтерактивної кнопки «Приєднатися». Компонент інтерфейсу UI перехоплює подію і викликає асинхронну функцію applyToProject() на рівні контролера API. На лініях життя активуються блоки фокусу контролю (Focus of Control), що візуалізують тривалість виконання операцій об'єктами.

– Етап бізнес-валідації: Контролер API надсилає синхронний запит checkEligibility() до бази даних DB для перевірки відповідності профілю волонтера (наявність блокувань, верифікація акаунту). База даних повертає статус відповіді return: OK за допомогою пунктирної стрілки зворотного зв'язку.

– Етап транзакції: Контролер викликає метод create() для сутності бази даних, створюючи новий документо-орієнтований запис «Заявка» (Application Document) зі стійким початковим станом status: Pending (на розгляді).

– Етап асинхронного сповіщення: Система надсилає подію-тригер на інтерфейс Організатора. У моделі зафіксовано часове обмеження (Time Constraint) за допомогою фігурних дужок, яке декларує, що час реакції організатора на розгляд заявки не повинен перевищувати 24 години з моменту створення таймауту.

– Етап схвалення: Організатор викликає функцію approveApplication(). Серверний контролер API надсилає запит на модифікацію стану документа у DB, змінюючи атрибут на status: Active.

– Етап завершення: Після успішного запису база даних повертає фінальне підтвердження, яке транслюється через UI на екран Волонтера. Після завершення операції сесійний деструктор викликає команду destroy() для очищення тимчасових об'єктів у пам'яті клієнта з метою оптимізації ресурсів пристрою.

Графічну схему динамічної взаємодії об'єктів UML Sequence представлено на рисунку 2.5.

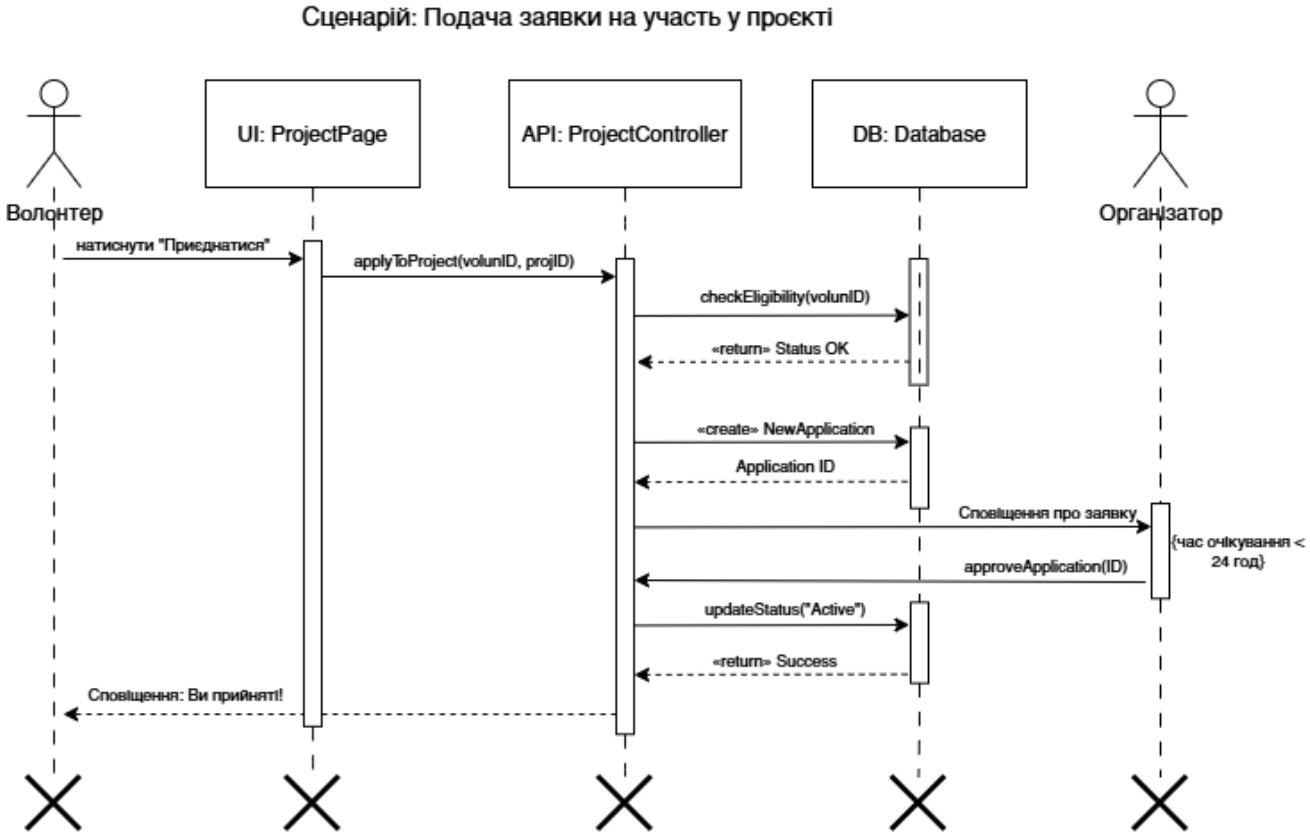

Рисунок 2.5 – UML діаграма послідовності (Sequence Diagram) процесу обробки заявки.

2.3 Проектування структури бази даних та сховища

Ефективне функціонування веб-платформи для координації волонтерської діяльності «UNITY+» безпосередньо залежить від архітектури збереження даних. Оскільки як серверну інфраструктуру обрано BaaS-платформу Appwrite, проектування бази даних базується на документо-орієнтованій (NoSQL) парадигмі, де замість класичних реляційних таблиць використовуються колекції документів у форматі JSON. Це забезпечує високу гнучкість при зміні схем даних, швидкий асинхронний доступ до інформації та автоматичне масштабування під високими навантаженнями.

2.3.1 Проектування логічної структури бази даних

Для формалізації сутностей предметної області та визначення зв'язків між ними на етапі системного проектування було побудовано діаграму «сутність-зв'язок» ERD (Entity-Relationship Diagram). Логічна структура сховища платформи UNITY+ детермінована такими основними сутностями:

- Користувач (User): ПІБ, Email, Телефон, Пароль, Роль (Волонтер/Організатор).
- Волонтер (Volunteer): Рейтинг, Години, Список навичок (Skills).
- Організація (Organization): Назва, Код ЄДРПОУ, Місія, Координатор.
- Проєкт (Project): Назва, Опис, Статус, Локація, Дедлайн.
- Завдання (Task): Назва, Статус (Виконано/В процесі), Трудомісткість.
- Відзнака (Badge): Назва, Умови отримання.

На відміну від реляційних СУБД, у документо-орієнтованій базі даних Appwrite зв'язки між сутностями реалізуються за допомогою концепцій збереження масивів унікальних ідентифікаторів (Document ID References) або впровадження документів. Конфігурацію архітектурних зв'язків системи спроектовано таким чином:

- Організація створює багато Проєктів (1:N).
- Проєкт містить багато Завдань (1:N).
- Волонтер долучається до багатьох Завдань (M:N).
- Волонтер отримує Відзнаки за виконання умов (M:N).

Графічне відображення зв'язків та логічної архітектури сховища наведено на рисунку 2.6.

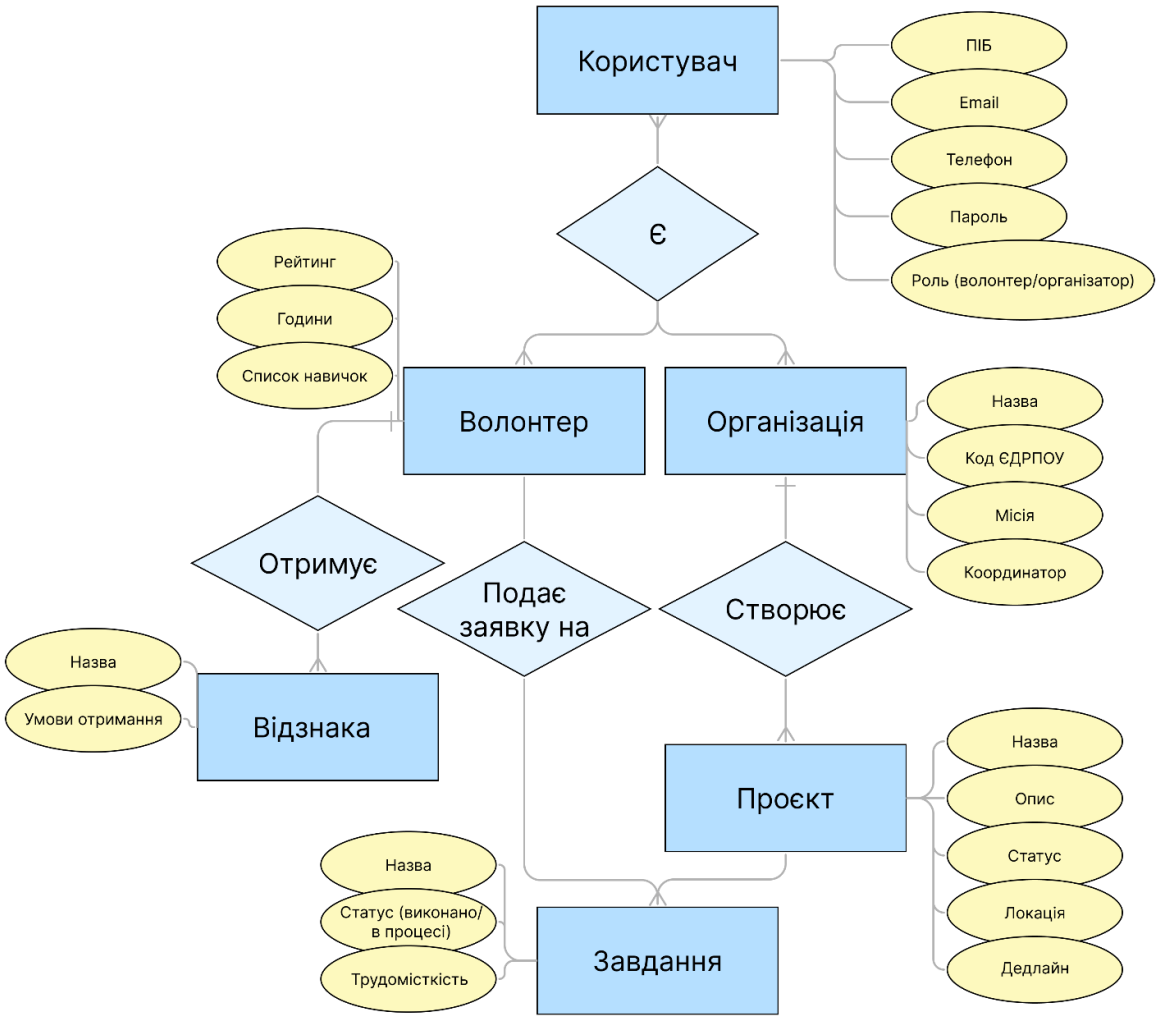

Рисунок 2.6 – Логічна схема бази даних (ERD) волонтерської платформи UNITY+.

2.3.2 Специфікація колекцій хмарної СУБД Appwrite

Для безпосереднього створення бази даних в адміністративній панелі хмари Appwrite було виконано декомпозицію логічної схеми на фізичні атрибути. Кожен атрибут отримав свій тип даних, правила валідації (Required/Nullable) та ліміти за розміром (Size). Фізичну структуру колекцій зведено в інженерні таблиці 2.5 та 2.6.

Таблиця 2.5 – Специфікація атрибутів колекції «Users»

Назва атрибута	Тип даних у СУБД	Критерії / Валідація	Системне призначення та логіка поля
$id	string	Системний ключ	Унікальний UUID документа користувача в системі Appwrite.
name	text	required	Повне ім'я волонтера або офіційна назва організації.
city	string	required, Size: 50	Географічне розташування для регіональної координації.
phone	text	Nullable	Контактний номер телефону для оперативного зв'язку.
email	email	required	Електронна пошта для авторизації та JWT-сесій.
edrpou	string	Nullable, Size: 12	Офіційний код ЄДРПОУ / ІПН (використовується лише для НУО).
role	string	required, Size: 20	Системна роль користувача (volunteer або organization).
skills[]	text	Array, Nullable	Масив тегів специфічних навичок та компетенцій волонтера.
avatar	string	Nullable, Size: 10000000	Посилання або Base64-код графічного аватара профілю.
volunteerHours	integer	Nullable	Сумарна кількість верифікованих відпрацьованих годин.
completedProjects	integer	Nullable	Кількість успішно завершених волонтерських місій.
coordinator	string	Nullable, Size: 200	ПІБ відповідального координатора від організації.
year	datetime	Nullable	Дата заснування організації або реєстрації волонтера.
mission	string	Nullable, Size: 500	Текстовий опис головної гуманітарної місії фонду
address	string	Nullable, Size: 500	Фізична адреса штабу або волонтерського хабу.

Продовження табл. 2.5

completedTasks[]	string	Array, Size: 10000	Масив унікальних ID успішно виконаних завдань.
activeTasks[]	string	Array, Size: 10000	Масив унікальних ID завдань, що перебувають у роботі.
$createdAt	datetime	Системне поле	Тимчасова мітка створення профілю користувача.
$updatedAt	datetime	Системне поле	Тимчасова мітка останньої модифікації профілю.

Таблиця 2.6 – Специфікація атрибутів колекції «Projects»

Назва атрибута	Тип даних у СУБД	Критерії / Валідація	Системне призначення та логіка поля
$id	string	Системний ключ	Унікальний UUID волонтерського проєкту.
title	string	required, Size: 200	Назва гуманітарної чи благодійної ініціативи.
description	string	required, Size: 5000	Детальний технічний опис мети та умов проєкту.
orgId	string	required, Size: 36	Зовнішній ключ (ID) організації-автора проєкту.
orgName	string	required, Size: 200	Публічна назва організації для відображення у картці.
city	string	required, Size: 100	Місто реалізації проєкту для пошукового фільтра.
address	string	required, Size: 200	Точна фізична локація для інтерактивної мапи.
status	string	required, Size: 20	Поточний стан проєкту (active, completed, draft).

Продовження табл. 2.6

volunteersNeeded	integer	required	Необхідний ліміт волонтерів для успішного старту.
volunteersJoined	integer	required	Кількість користувачів, які вже закрили вакансії.
startDate	datetime	Nullable	Офіційна дата початку виконання перших завдань.
endDate	datetime	Nullable	Запланована дата закриття та архівації проєкту.
deadline	datetime	Nullable	Крайній термін прийому нових заявок від волонтерів.
requirements[]	string	Array, Size: 20000	Масив текстових вимог до учасників місії.
tasks[]	string	Array, Size: 10000	Загальний перелік декомпонованих задач проєкту.
category	string	Nullable, Size: 50	Категорія волонтерства (відбудова, медицина тощо).
contactPhone	string	Nullable, Size: 20	Контактний телефон гарячої лінії проєкту.
contactEmail	string	Nullable, Size: 100	Контактна пошта для координації стейкхолдерів.
isUrgent	boolean	required	Прапорець терміновості (маркування тегом «Терміново!»).
volunteers[]	string	Array, Size: 100	Список ID користувачів, затверджених у команду.
skills[]	string	Array, Size: 10000	Масив ID ключових навичок, необхідних для проєкту.

Продовження табл. 2.6

image	string	Nullable, Size: 5000000	URL-посилання або Base64 банера сторінки проєкту.
isRemote	boolean	default: false	Ознака дистанційного (онлайн) формату волонтерства.
applications	string	Nullable, Size: 10000	Масив ID поданих заявок, що очікують модерації.
totalHours	double	Nullable	Загальний обсяг годин, запланований на проєкт.
completedHours	double	Nullable	Кількість фактично виконаних і підтверджених годин.
activeTasks[]	string	Array, Size: 10000	Поточні активні задачі, виведені на Канбан-дошку.
$createdAt	datetime	Системне поле	Дата та час публікації проєкту на платформі.
$updatedAt	datetime	Системне поле	Дата та час останнього оновлення даних ініціативи.

Результати декомпозиції та налаштування типів даних і обмежень для кожного окремого атрибута колекцій NoSQL бази даних було успішно зафіксовано через адміністративну веб-консоль. Приклад фізичного налаштування ліміту символів для масиву зв'язків активних завдань наведено на рисунку 2.7.

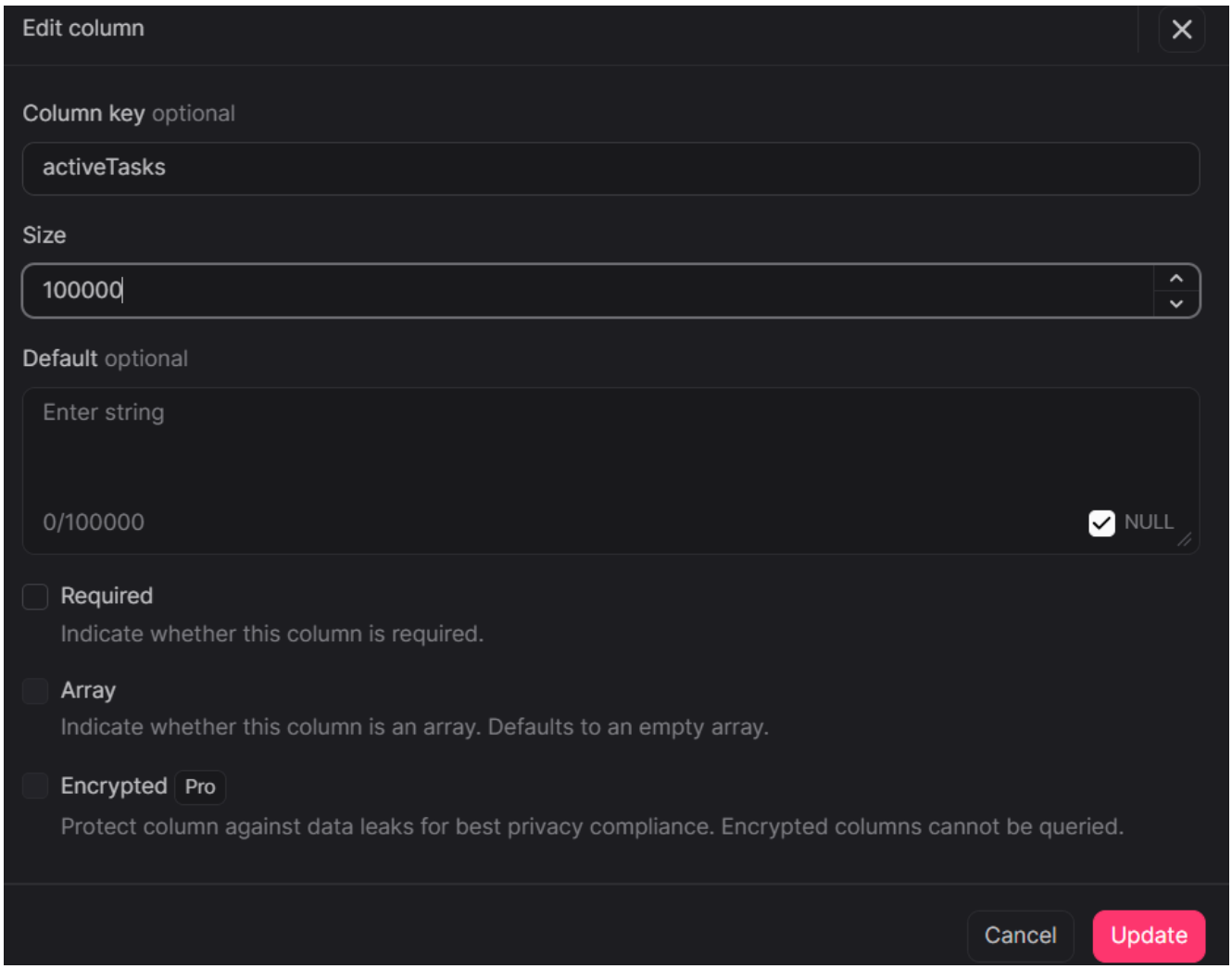

Рисунок 2.7 – Процес конфігурації NoSQL атрибута activeTasks в адміністративній панелі Appwrite Databases.

Дані конфігурації, наведені на рисунку, підтверджують забезпечення персистентного збереження списків ідентифікаторів у текстовому форматі із максимальним лімітом до 100 000 символів на документ.

2.3.3 Об'єктне проектування структури

Для проектування програмного коду клієнтської частини та опису типізації об'єктів (в межах архітектури TypeScript/JavaScript компонентів) було розроблено UML діаграму класів (Class Diagram). Модель визначає структуру об'єктів у пам'яті додатка, їхні внутрішні методи та інтерфейси взаємодії.

Опис класів та їхнього системного функціоналу детерміновано таким чином:

– *Інтерфейс IVerifiable:* Визначає обов'язковий стандарт (контракт) для об'єктів, які потребують офіційної перевірки в системі. Містить метод +verifyStatus(): boolean — повертає результат перевірки легітимності сутності (наприклад, перевірки коду ЄДРПОУ через API).

– *Клас User (Базовий):* Слугує основою для всіх типів користувачів, об'єднуючи спільні дані для авторизації та керування профілем. Містить ідентифікатор (id), повне ім'я (name), електронну пошту (email) та пароль (password). Реалізує методи оновлення профілю +updateProfile() та авторизації. Виступає батьківським класом (зв'язок узагальнення) для Volunteer та Organization.

– *Клас Volunteer:* Представляє волонтера, зберігаючи дані про його досвід, навички та рейтинг. Включає список навичок (skills), загальну кількість годин (totalHours) та поточний рейтинг (rating). Реалізує методи подачі заявок на участь у проєктах +applyToProject() та перегляду особистого портфоліо досягнень +viewPortfolio(). Наслідує всі властивості класу User та має асоціативний зв'язок із класом Project для участі в ініціативах.

– *Клас Organization:* Описує благодійну організацію чи фонд, що створює та координує волонтерські проєкти. Містить офіційний код EDRPOU, місію організації та статус верифікації (isVerified). Дозволяє створювати нові проєкти +createProject() та підтверджувати відпрацьовані волонтерами години +verifyVolunteerHours(). Наслідує клас User, реалізує інтерфейс IVerifiable та створює проєкти (асоціативний зв'язок 1..*).

– *Клас Project:* Є центральною сутністю для управління конкретною волонтерською ініціативою. Містить назву (title), детальний опис (description) та поточний статус проєкту. Дозволяє додавати специфічні вимоги до волонтерів +addRequirement() та генерувати звіти про результати +generateReport(). Жорстко володіє списком завдань через зв'язок композиції (завдання не можуть існувати без об'єкта Project). Використовує сервіс сповіщень (NotificationService).

– *Клас Task:* Описує окреме завдання (атом робіт) у межах проєкту для відображення на Канбан-дошці. Включає назву завдання (taskName), статус виконання (isCompleted) та кількість передбачених годин (assignedHours). Реалізує метод оновлення стану виконання завдання +updateStatus(). Є невід'ємною частиною композиції проєкту.

– *Клас NotificationService:* Відповідає за автоматичну розсилку сповіщень та системних алертів користувачам. Містить метод +sendAlert(User user) — надсилає повідомлення конкретному об'єкту типу User. Використовується класами Project та User для інформування про важливі події (зв'язок залежності <<use>>).

Графічну схему структури об'єктів та архітектури класів UML Class Diagram представлено на рисунку 2.8.

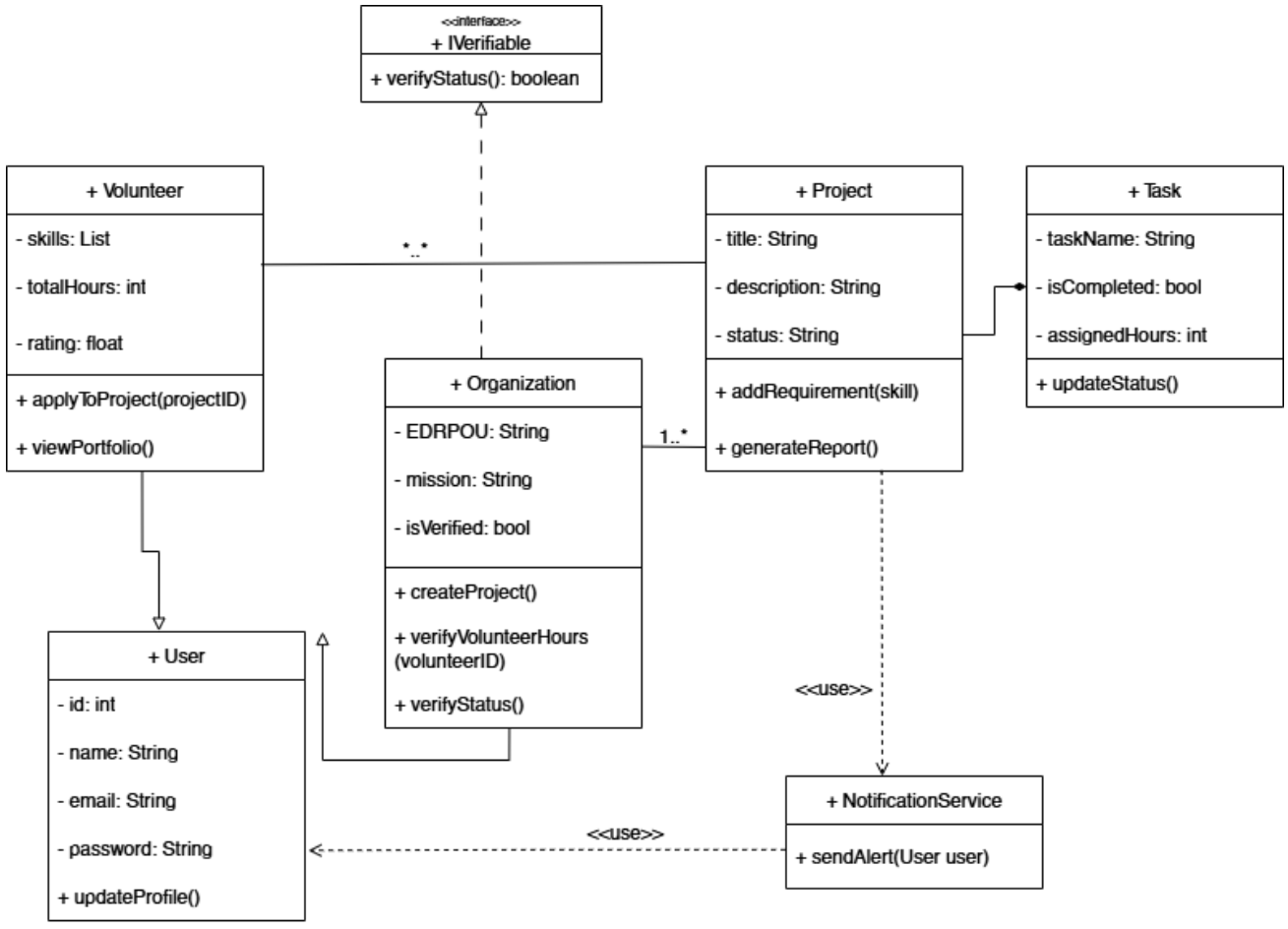

Рисунок 2.8 – UML діаграма класів (Class Diagram) веб-платформи UNITY+.

2.3.4 Оптимізація збереження медіаданих та використання Base64

Під час проектування архітектури системи «UNITY+» було проведено аналіз та оптимізацію використання хмарних ресурсів BaaS-платформи Appwrite. З метою мінімізації навантаження на сервер, виключення додаткових витрат на хмарне сховище (Appwrite Storage) та забезпечення повної техніко-економічної незалежності проєкту, було прийнято рішення відмовитися від використання фізичних хмарних бакетів для збереження файлів.

Замість цього в системі реалізовано інженерний підхід — кодування бінарних даних зображень у текстовий формат Base64 безпосередньо на стороні клієнта (Frontend) перед відправкою запиту. Це дозволило повністю інтегрувати збереження графічного контенту в існуючу структуру NoSQL документів:

– Графічні аватари користувачів обробляються на фронтенді, стискаються та записуються як ASCII-рядок в атрибут avatar (тип даних string, ліміт Size: 10000000) колекції Users;

– Банери та фотозвіти проєктів трансформуються в оптимізований текстовий формат та зберігаються в атрибут image (розмір обмежено лімітом Size: 5000000 символів) колекції Projects.

Таке архітектурне рішення дозволило об'єднати текстові й графічні потоки даних в межах єдиних асинхронних транзакцій до СУБД, суттєво прискорило швидкість розгортання системи, зменшило кількість необхідних API-запитів до бекенду та забезпечило автономну роботу платформи без підключення платних тарифних планів сховища.

2.4 Програмна реалізація клієнтської та серверної частин

Програмна реалізація інформаційної системи «UNITY+» базується на розподіленій клієнт-серверній архітектурі Single Page Application (SPA). Фронтенд-частина розроблена на базі сучасної бібліотеки компонентного підходу React (екосистема JavaScript), що забезпечує реактивне оновлення інтерфейсу

користувача в реальному часі та модульну структуру коду. Функціонал серверної частини (автентифікація, менеджмент NoSQL бази даних, JWT-сесії, безпека та фільтрація) повністю делегований готовим сервісам BaaS-платформи Appwrite (Backend-as-a-Service) за допомогою інтеграції офіційного інструментарію Appwrite Web SDK.

2.4.1 Конфігурація інфраструктури та ініціалізація Appwrite SDK

Для забезпечення безшовного асинхронного зв'язку між React-клієнтом та хмарним бекендом у структурі проєкту було ініціалізовано та налаштовано базовий конфігураційний клієнтський модуль. Зв'язок реалізується через захищений протокол HTTPS за допомогою передачі унікальних ключів ідентифікації проєкту (Project ID).

Для забезпечення масштабованості, простоти супроводу коду та чіткого розділення архітектурних шарів веб-платформи «UNITY+» було сформовано модульну структуру каталогів проєкту екосистеми React. Логічну архітектуру розподілу файлів у каталозі src представлено на рисунку 2.14.

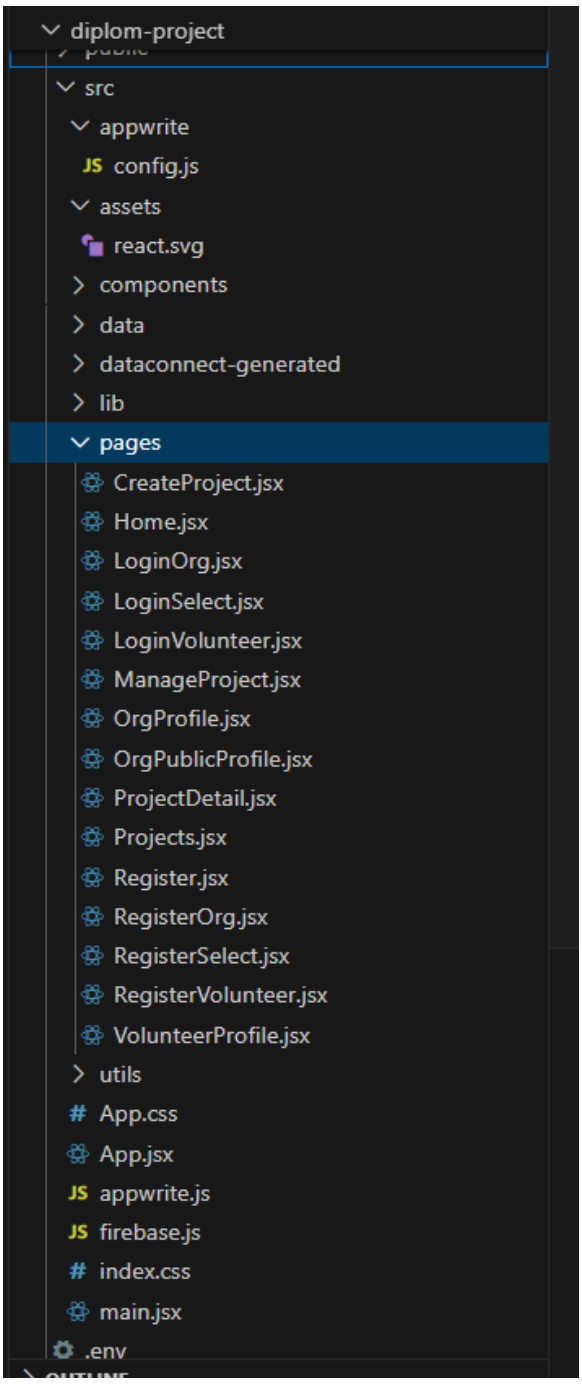

Рисунок 2.9 – Ієрархічна структура папок та компонентів вихідного коду системи UNITY+.

Відповідно до розробленої архітектури (рисунок 2.14), каталоги мають таке інженерне призначення:

– appwrite/ — містить конфігураційні скрипти ініціалізації клієнта BaaS-платформи та зв'язку з NoSQL базою даних;

– assets/ — збереження статичних графічних ресурсів, векторних іконок та логотипів;

– components/ — бібліотека атомарних UI-компонентів клієнтського інтерфейсу, що використовуються повторно;

– data/ — статичні структури даних (наприклад, задокументований довідник навичок волонтерів);

– pages/ — модулі екранних форм додатка, що відповідають за рендеринг цілісних сторінок системи (реєстрація, профілі стейкхолдерів, таск-менеджмент);

– utils/ — допоміжні функції, валідатори форм та кастомні React-хуки.

Програмний код ініціалізації системного зв'язку та підключення хмарних сервісів автентифікації (Account) та NoSQL бази даних (Databases) наведено на рисунку 2.10.

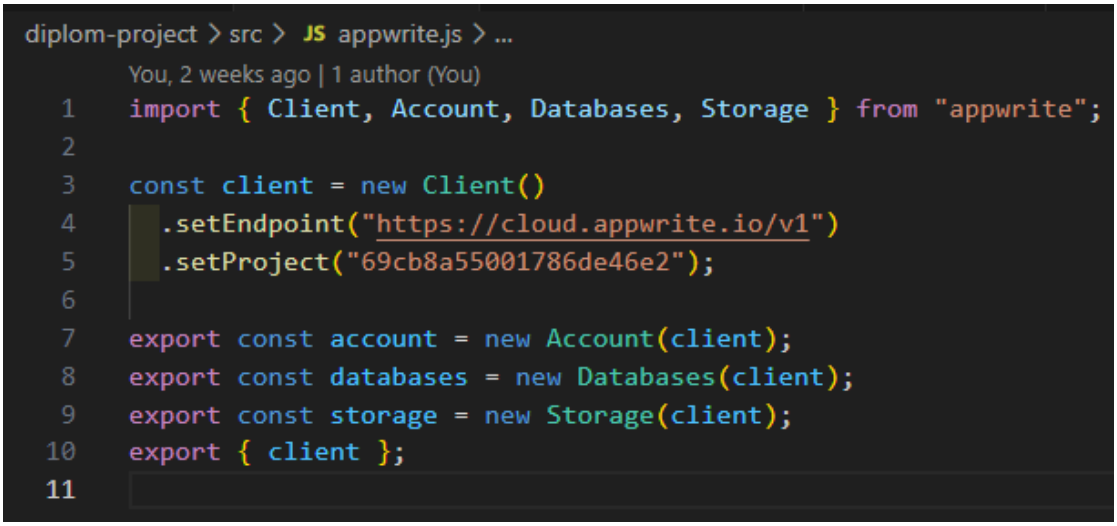

Рисунок 2.10 – Лістинг модуля ініціалізації хмарного сервісу Appwrite (src/appwrite.js)

Таке архітектурне рішення дозволяє уникнути необхідності написання та супроводу власного громіздкого серверного коду, забезпечуючи високу швидкість обробки асинхронних запитів безпосередньо з клієнтської частини додатка.

2.4.2 Реалізація архітектури керування сесіями та ролевої моделі доступу

Оскільки вебплатформа передбачає суворе розмежування користувацьких інтерфейсів, функціональних можливостей та прав доступу для двох різних типів акторів системи (volunteer та organization), виникла необхідність створення

централізованого модуля перевірки безпеки та менеджменту станів додатка. У межах розробки архітектури Single Page Application було прийнято інженерне рішення інтегрувати логіку глобального контролю сесій безпосередньо у кореневий програмний компонент додатка — src/App.jsx.

Для оптимізації кількості запитів до серверної частини BaaS-платформи Appwrite та уникнення затримок при рендерингу інтерфейсу, у системі реалізовано комбінований підхід: синхронізація поточного стану користувача (user, loading) із хмарою поєднується з персистентним збереженням ідентифікатора ролі у локальному сховищі браузера localStorage.

При кожній первинній ініціалізації додатка або зміні маршруту автоматично тригереться асинхронна функція checkUserSession(). Дана функція звертається до методу account.get() офіційного Web SDK Appwrite для перевірки валідності поточної JWT-сесії користувача. У разі успішного запиту стан user наповнюється об'єктом облікового запису, а з localStorage зчитується активна роль користувача для коректного функціонування захищених маршрутів (Protected Routes). У випадку генерації помилки (відсутність сесії або завершення терміну дії токена), система автоматично очищує локальне сховище та скидає стан користувача до значення null, блокуючи доступ до кабінетів.

Програмну логіку кореневого компонента, що відповідає за ініціалізацію хмарних сесій та глобальну клієнтську маршрутизацію SPA-додатка, наведено на рисунку 2.10.

```jsx
import React, { useEffect, useState } from "react";
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Navigate,
} from "react-router-dom";
import { account } from "./lib/appwrite";

// Компоненти
import Navbar from "./components/Navbar";
import Footer from "./components/Footer";

// Сторінки
import Home from "./pages/Home";
import Projects from "./pages/Projects";
import VolunteerProfile from "./pages/VolunteerProfile";
import OrgProfile from "./pages/OrgProfile";
import LoginVolunteer from "./pages/LoginVolunteer";
import LoginOrg from "./pages/LoginOrg";
import RegisterVolunteer from "./pages/RegisterVolunteer";
import RegisterOrg from "./pages/RegisterOrg";
import LoginSelect from "./pages/LoginSelect";
import RegisterSelect from "./pages/RegisterSelect";
import CreateProject from "./pages/CreateProject";
import ManageProject from "./pages/ManageProject";
import ProjectDetails from "./pages/ProjectDetail";
import OrgPublicProfile from "./pages/OrgPublicProfile";
import ForgotPassword from "./components//ForgotPassword";
import ResetPassword from "./components/ResetPassword";

function App() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [role, setRole] = useState(localStorage.getItem("role"));

  const checkUserSession = async () => {
    try {
      const currentUser = await account.get();
      console.log("👤 Поточний користувач:", currentUser);
      setUser(currentUser);

      const savedRole = localStorage.getItem("role");
      if (savedRole) {
        setRole(savedRole);
      }
    } catch (error) {
      console.log("ℹ Користувач не авторизований");
      setUser(null);
      localStorage.removeItem("role");
      setRole(null);
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    checkUserSession();
  }, []);

  useEffect(() => {
    const handleStorageChange = () => {
      const savedRole = localStorage.getItem("role");
      if (savedRole) {
        setRole(savedRole);
        account
          .get()
          .then(setUser)
          .catch(() => setUser(null));
      } else {
        setRole(null);
        setUser(null);
      }
    };

    window.addEventListener("storage", handleStorageChange);

    const interval = setInterval(() => {
      const savedRole = localStorage.getItem("role");
      if (savedRole !== role) {
        if (savedRole) {
          setRole(savedRole);
          account
            .get()
            .then(setUser)
            .catch(() => setUser(null));
        } else {
          setRole(null);
          setUser(null);
        }
      }
    }, 1000);
```

```
    return () => {
      window.removeEventListener("storage", handleStorageChange);
      clearInterval(interval);
    };
  }, [role]);

  if (loading)
    return (
      <div className="min-h-screen flex items-center justify-center font-bold text-[32px] text-[#212121]">
        <div className="flex flex-col items-center gap-4">
          <div className="w-12 h-12 border-4 border-[#FFE24E] border-t-transparent rounded-full animate-spin"></div>
          <p>Завантаження Unity+...</p>
        </div>
      </div>
    );

  const getProfileRoute = () => (role === "org" ? "/org-profile" : "/profile");

  return (
    <Router>
      <div className="flex flex-col min-h-screen font-['Manrope'] bg-[#f1f1f1]">
        <Navbar user={user} userRole={role} />
        <main className="flex-grow">
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/create-project" element={<CreateProject />} />
            <Route path="/projects" element={<Projects />} />
            <Route path="/projects/:id" element={<ProjectDetails />} />
            <Route path="/manage-project/:id" element={<ManageProject />} />
            <Route path="/organization/:orgId" element={<OrgPublicProfile />} />

            <Route
              path="/login"
              element={
                !user ? <LoginSelect /> : <Navigate to={getProfileRoute()} />
              }
            />
            <Route
              path="/register"
              element={
                !user ? <RegisterSelect /> : <Navigate to={getProfileRoute()} />
              }
            />

            <Route
              path="/login-volunteer"
              element={!user ? <LoginVolunteer /> : <Navigate to="/profile" />}
            />
            <Route
              path="/register-volunteer"
              element={
                !user ? <RegisterVolunteer /> : <Navigate to="/profile" />
              }
            />

            <Route
              path="/login-org"
              element={!user ? <LoginOrg /> : <Navigate to="/org-profile" />}
            />
            <Route
              path="/register-org"
              element={!user ? <RegisterOrg /> : <Navigate to="/org-profile" />}
            />

            <Route
              path="/profile"
              element={
                user ? <VolunteerProfile /> : <Navigate to="/login-volunteer" />
              }
            />
            <Route
              path="/org-profile"
              element={user ? <OrgProfile /> : <Navigate to="/login-org" />}
            />

            <Route path="*" element={<Navigate to="/" />} />
            <Route path="/forgot-password" element={<ForgotPassword />} />
            <Route path="/reset-password" element={<ResetPassword />} />
          </Routes>
        </main>
        <Footer />
      </div>
    </Router>
  );
}

export default App;

```

Рисунок 2.11 – Лістинг модуля кореневого маршрутизатора та глобального контролю сесій користувачів (src/App.jsx)

Завдяки такій програмній структурі, опис клієнтського роутингу та ізоляція приватних кабінетів (компоненти VolunteerProfile та OrgProfile) відбуваються динамічно на рівні декларативного опису шляхів Routes. Це повністю виключає можливість несанкціонованого переходу неавторизованих користувачів до адміністративних модулів керування проєктами.

2.4.3 Програмна логіка підсистем обліку годин та гейміфікації

Автоматизація трекінгу діяльності та нарахування досягнень є ключовим інноваційним компонентом платформи «UNITY+».

– Облік волонтерських годин: реалізовано за допомогою фіксації системних часових міток. Коли організатор змінює статус завдання на Канбан-дошці на значення done, на стороні клієнта запускається асинхронний метод, який вираховує різницю між датами, бере значення передбаченої трудомісткості з поля assignedHours, надсилає асинхронний запит до Appwrite Databases та оновлює атомарне поле volunteerHours у документі відповідного волонтера.

– Модуль гейміфікації та бейджів: функціонує як набір предикативних логічних правил на фронтенді додатка. У момент успішного оновлення профілю система перевіряє поточне значення volunteerHours. Якщо воно перевищує встановлені тригери (наприклад, 10, 50 або 100 годин), користувачу автоматично розблоковується відповідний цифровий бейдж, а графічна іконка досягнення підтягується в його публічне портфоліо.

2.4.4 Тестування, життєвий цикл розробки та розгортання

Життєвий цикл створення програмного продукту UNITY+ включав кілька послідовних етапів верифікації та тестування:

– Альфа-тестування (Внутрішнє): виконувалося безпосередньо автору проєкту під час кодування за допомогою інструментів розробника (React Developer Tools, консоль логування Appwrite). На цьому етапі було виявлено та

усунено критичні помилки асинхронної синхронізації даних при паралельному оновленні масивів завдань.

– Бета-тестування (Цільове): проводилося із залученням фокус-групи представників місцевих волонтерських організацій та активістів. Тестування дозволило оптимізувати юзабіліті адаптивного інтерфейсу та виправити баги валідації форм при введенні кодів ЄДРПОУ.

– Розгортання (Deployment): весь вихідний код клієнтської та конфігураційної частин веб-платформи UNITY+ було розміщено у віддаленому репозиторії на платформі GitHub (monamrrr/unity-plus). Для забезпечення публічного доступу до платформи фронтенд-частина додатка була успішно скомпільована та розгорнута безпосередньо з репозиторію на хмарі стабільного продакшн-сервера за допомогою автоматизованого налаштування процесів CI/CD (Continuous Integration / Continuous Deployment), реалізованих шляхом безшовної інтеграції GitHub та хмарної консолі Appwrite. Це гарантує автоматичне тригерення процесів збирання проєкту та миттєве оновлення інтерфейсу користувача в браузері при кожному внесенні змін (push) до головної гілки репозиторію..

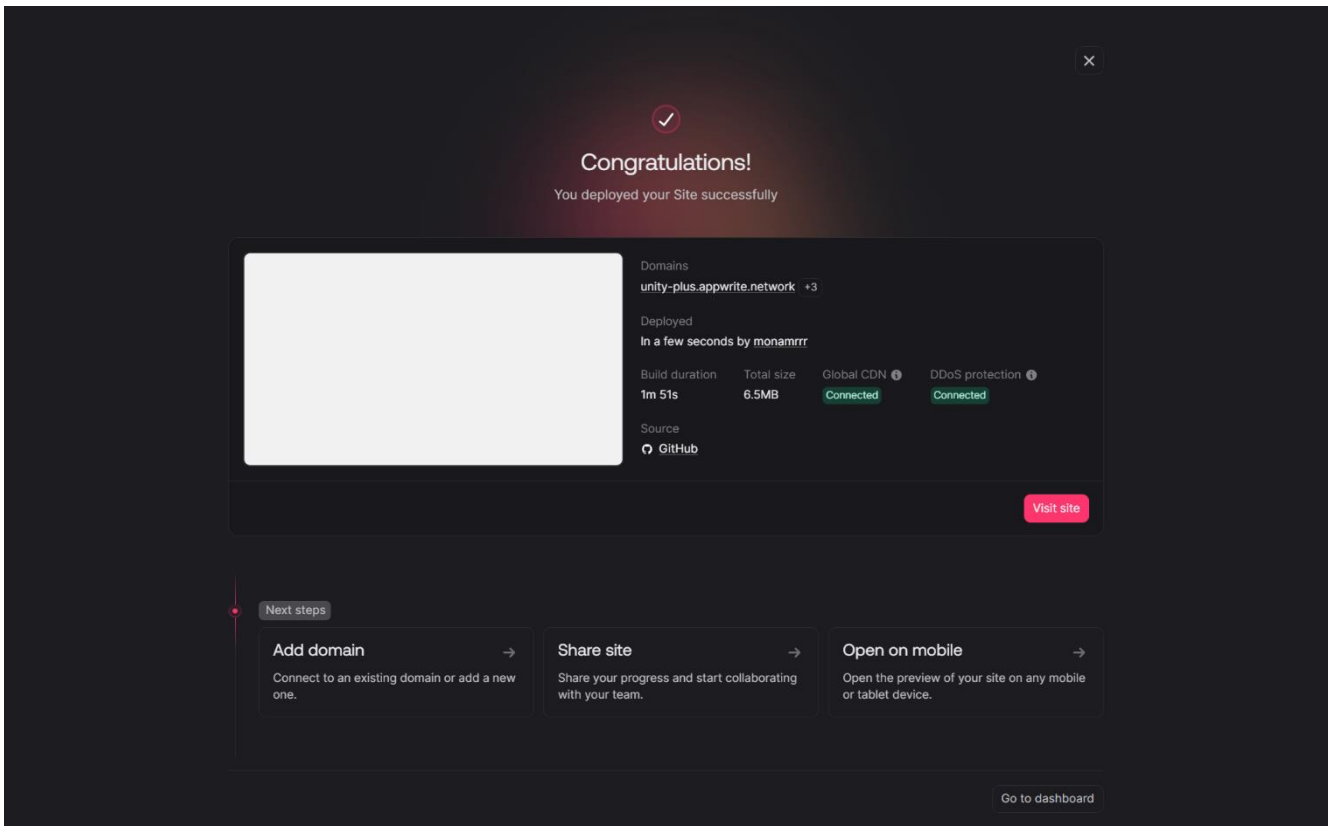

Рисунок 2.12 – Інтерфейс успішного розгортання веб-платформи UNITY+ у хмарній консолі Appwrite.

Як видно з рисунку 2.12, загальний розмір скомпільованого продукту склав 6.5 МБ, а процес автоматичного розгортання вихідного коду з GitHub-репозиторію на продакшн-домені unity-plus.appwrite.network зайняв 1 хвилину та 51 секунду, що підтверджує високу оптимізацію клієнтського коду додатка.

2.5 Розробка інтерфейсу користувача (UI/UX) та тестування

Фінальний етап технічної реалізації веб-платформи «UNITY+» охоплює розробку адаптивного клієнтського інтерфейсу (UI/UX) та проведення комплексного функціонального тестування з метою перевірки відповідності створеного програмного забезпечення сформульованим технічним вимогам.

2.5.1 Інженерні рішення UI/UX дизайну

Під час перенесення графічних прототипів із середовища Figma в архітектуру

React-компонентів, особливу увагу було приділено проектуванню користувацького досвіду (UX) для десктопних систем. Інтерфейс платформи базується на компонентному підході, де кожен елемент є ізольованим модулем, що повторно використовується.

Основними інженерними рішеннями при розробці UI/UX стали:

– Оптимізація під десктопні пристрої: Оскільки координація великої кількості волонтерських завдань та робота з Канбан-дошками вимагає значного екранного простору, інтерфейс системи повністю оптимізовано для моніторів та ноутбуків. Це забезпечує безперешкодний одночасний вивід аналітики, фільтрів та карток проєктів на одному екрані без втрати читабельності даних.

– Зниження когнітивного навантаження: Важливі аналітичні блоки (метрики годин, прогрес-бари благодійних зборів) винесені на верхні рівні ієрархії сторінок. Для інтерфейсу організації застосовано зручну дашборд-структуру, яка дозволяє координувати волонтерів та таски в межах одного робочого вікна браузера.

– Асинхронний відгук (Micro-interactions): Взаємодія з NoSQL бекендом Appwrite супроводжується миттєвим візуальним фідбеком — скелетними екранами завантаження (Skeletons), динамічними анімаціями зміни статусів карток Канбан та інтерактивними спливаючими сповіщеннями (Toasts) у разі помилок авторизації або успішного закриття тасок.

Практична програмна реалізація десктопного користувацького інтерфейсу дозволила трансформувати графічні прототипи у готові інтерактивні сторінки в межах логічного користувацького сценарію. Первинна взаємодія користувача з системою починається з форми реєстрації (рисунок 2.13).

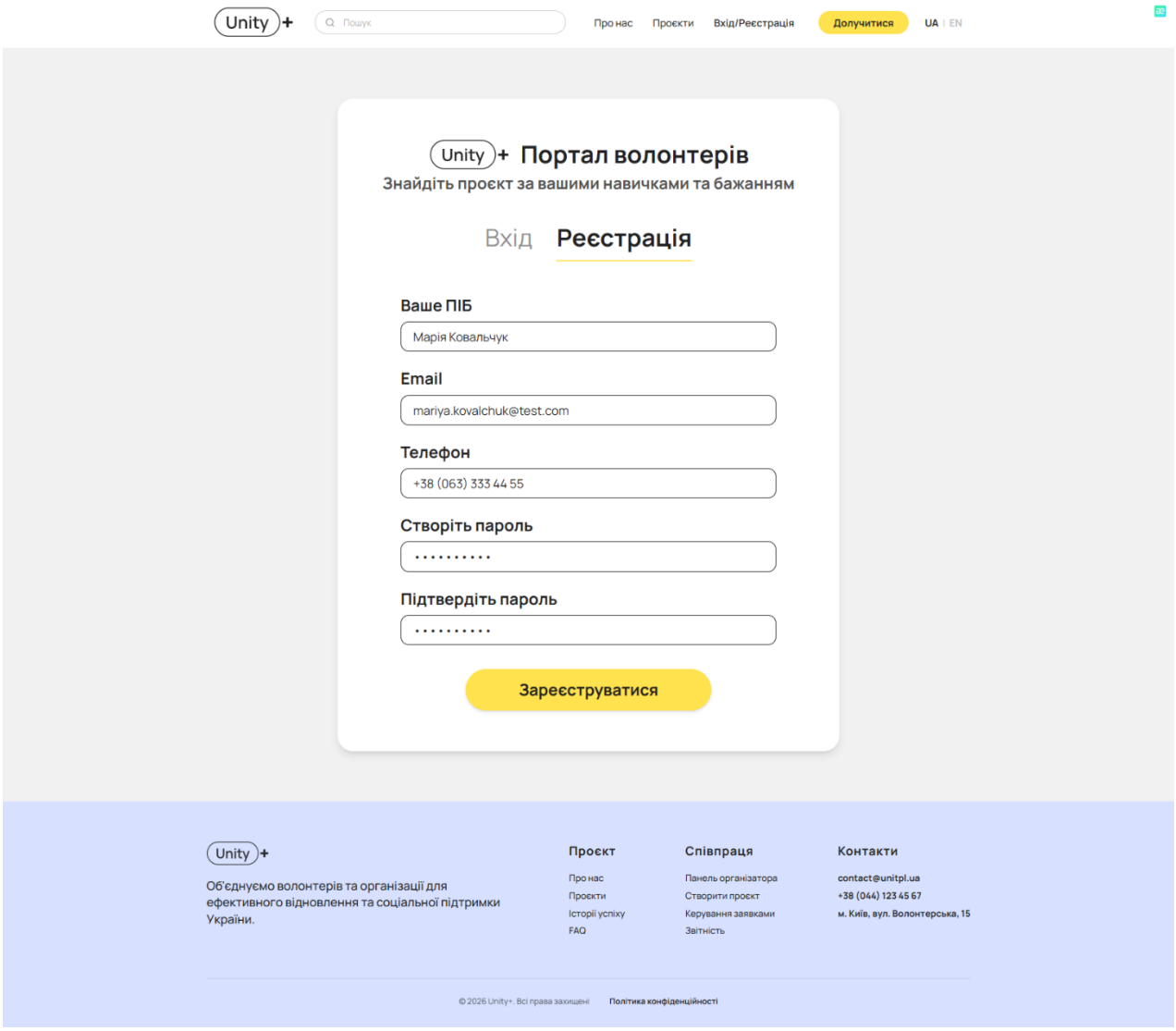

Рисунок 2.13 – Екранна форма реалізованого модуля реєстрації нових користувачів.

Після успішного онбордингу некомерційна організація отримує можливість створення та покрокової декомпозиції гуманітарних місій на конкретні завдання за допомогою форми, інтерфейс якої представлено на рисунку 2.14.

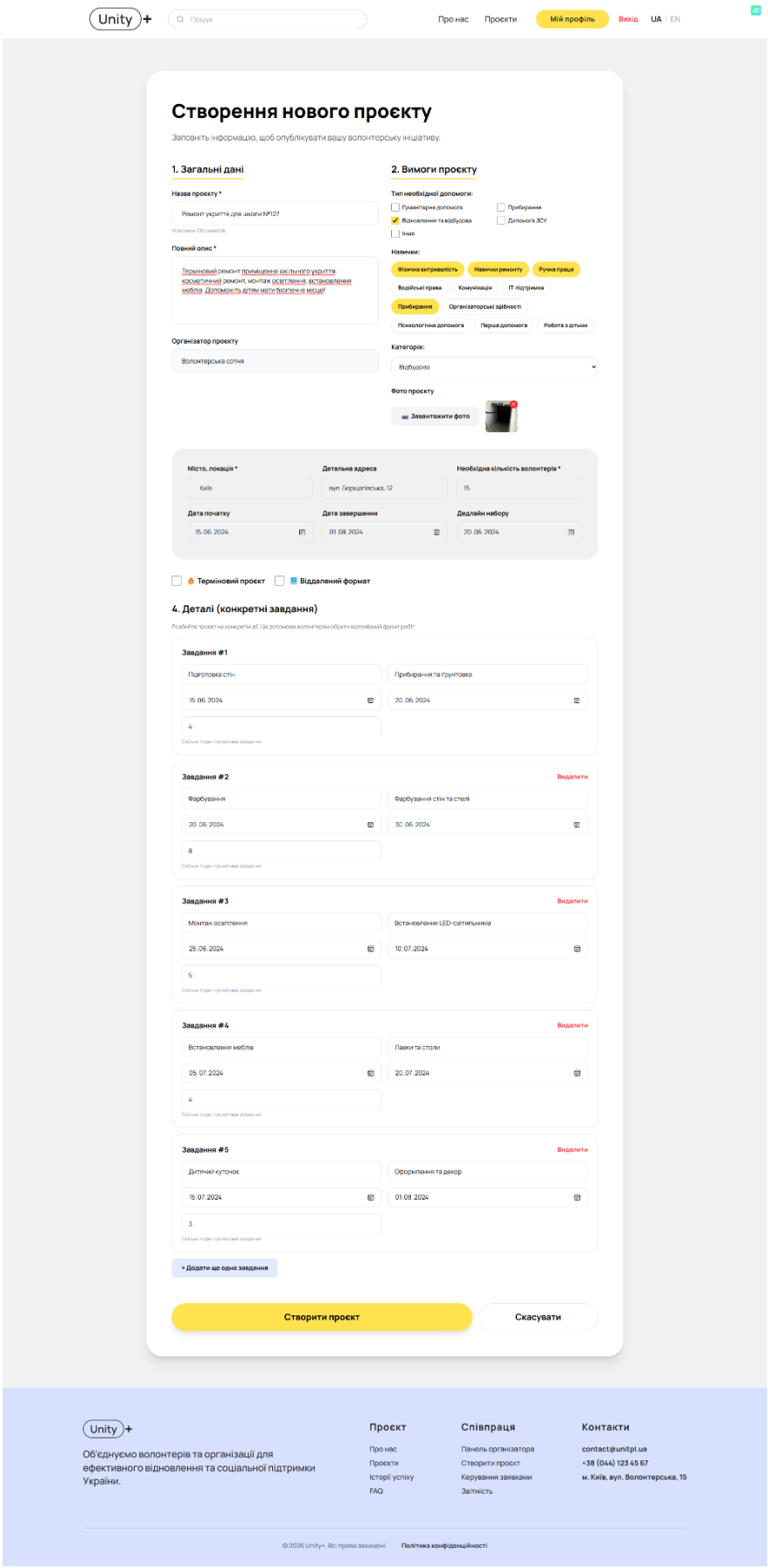

Рисунок 2.14 – Реалізований інтерфейс покрокової форми ініціалізації та декомпозиції волонтерського проєкту.

Опублікована ініціатива автоматично стає доступною для пошуку та відображається у вигляді детального інформаційного хабу проєкту для волонтерів (рисунок 2.15), де інтегровано карту Leaflet/OpenStreetMap та модулі подачі заявок на атомарні задачі.

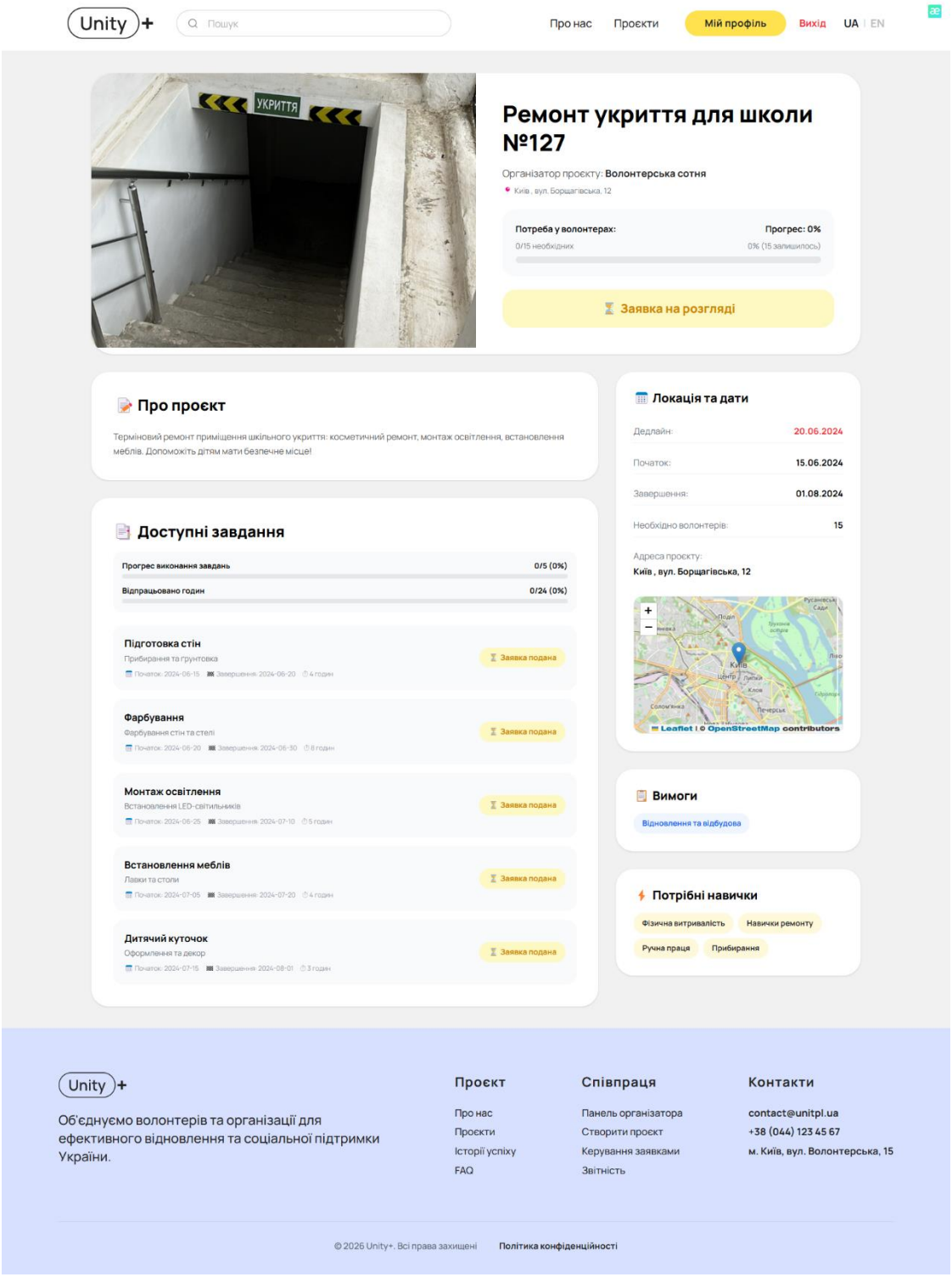

Рисунок 2.15 – Інтерфейс реалізованої екранної форми інформаційного хабу проєкту для волонтера.

Оперативний моніторинг поточної діяльності, облік залучених волонтерів, розгляд нових анкет та перегляд автоматизованої внутрішньої аналітики фонду здійснюється через особистий кабінет координатора (рисунок 2.16).

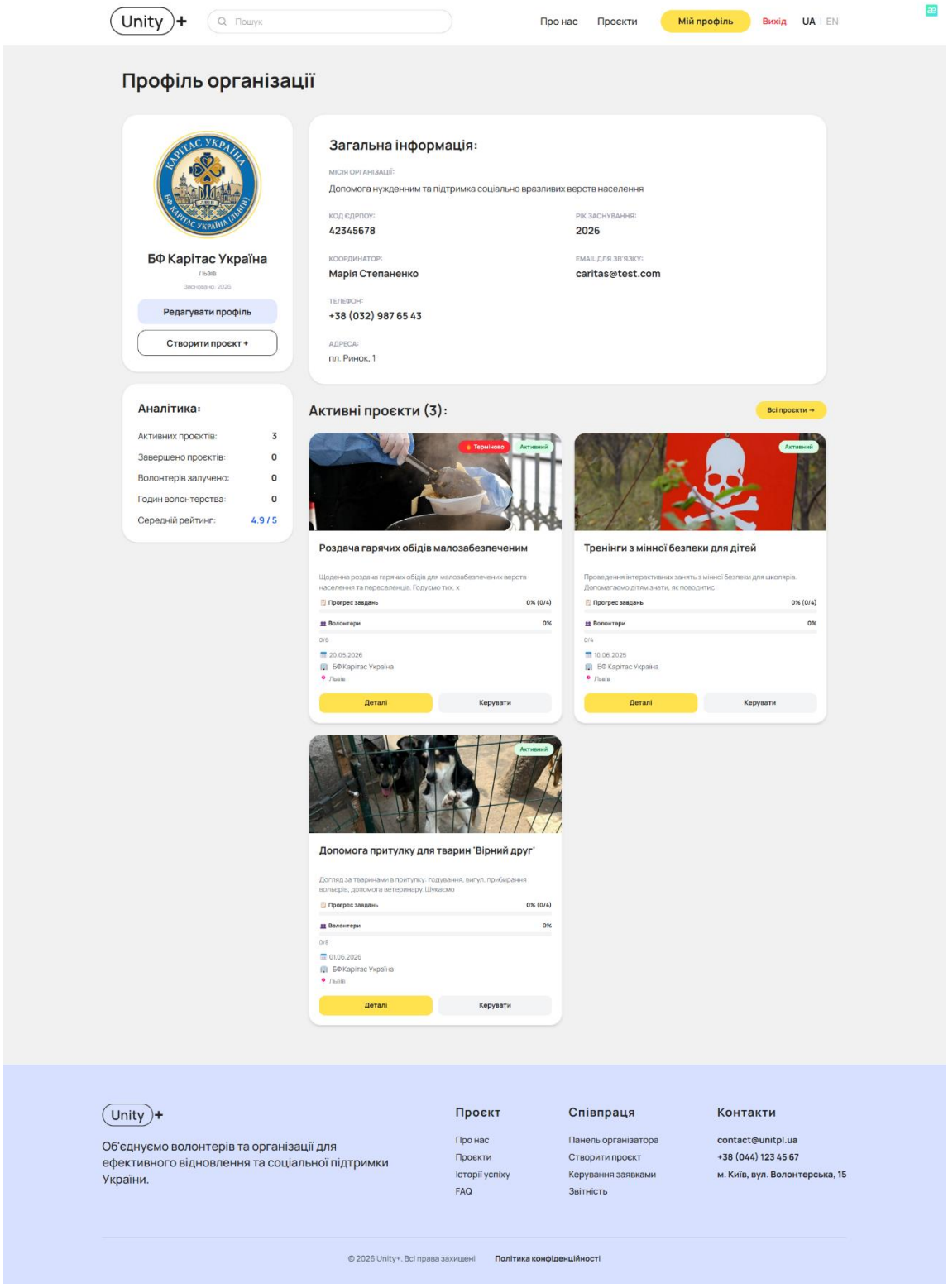

Рисунок 2.16 – Функціонування адміністративного дашборду організації з блоками аналітики та картками активних місій.

2.5.2 Функціональне тестування системи та верифікація вимог

З метою підтвердження надійності, валідності та безпеки обробки інформаційних потоків у системі було проведено функціональне модульне та інтеграційне тестування за методом «чорної скриньки» (Black Box Testing). У ході тестування імітувалися реальні сценарії використання платформи різними типами користувачів на основі вищеописаних екранних форм (рисунки 2.10–2.13).

Для верифікації критичних вузлів системи (модуля автентифікації сесій, системи розмежування ролей, логіки валідації юридичних даних та трекінгу Канбан-завдань) було розроблено серію тест-кейсів. Детальні специфікації та результати проведення випробувань зведено в інженерні таблиці 2.8, 2.9 та 2.10.

Таблиця 2.8 – Тест-кейс №1: Реєстрація організації з валідацією коду ЄДРПОУ.

Компонент тесту	Технічні параметри та опис виконання операції
Об'єкт тестування	Модуль реєстрації нового облікового запису класу organization (рисунок 2.10).
Вхідні дані	name: "БФ Світанок", email: "bf@test.com", role: "organization", edrpou: "12345" (некоректна довжина).
Кроки виконання	1. Перейти на сторінку реєстрації організації. 2. Заповнити текстові поля вхідними даними. 3. Натиснути кнопку ініціалізації транзакції "Зареєструвати".
Очікуваний результат	Система перехоплює подію, спрацьовує фронтенд-валідатор довжини рядка. Запит до Appwrite не відправляється. Під полем коду з'являється червоний алерт: "Код ЄДРПОУ повинен містити 8 або 12 цифр".
Фактичний результат	Поведінка інтерфейсу повністю відповідє очікуваній. Помилка валідації відпрацювала коректно. Транзакція заблокована.
Статус тесту	Успішно (Passed)

Таблиця 2.9 – Тест-кейс №2: Розмежування ролей при автентифікації сесії

Компонент тесту	Технічні параметри та опис виконання операції
Об'єкт тестування	Глобальний провайдер контексту AuthContext та хук useAuth().
Вхідні дані	Обліковий запис користувача з атрибутом role: "volunteer".
Кроки виконання	1. Авторизуватися в системі під профілем волонтера. 2. Спробувати виконати прямий перехід за захищеною URL-адресою адміністративного дашборду організації: /dashboard/organization.
Очікуваний результат	Метод checkUserStatus() зчитує роль документа. Спрацьовує роутер безпеки (Protected Routes). Система блокує рендеринг адмінки та автоматично перенаправляє (Redirect) користувача на сторінку публічного пошуку проєктів.
Фактичний результат	Доступ заблоковано, виконано миттєвий редірект на сторінку пошуку. Права доступу ізольовані на рівні JWT-сесії.
Статус тесту	Успішно (Passed)

Таблиця 2.10 – Тест-кейс №3: Автоматичний трекінг таоблік волонтерських годин

Компонент тесту	Технічні параметри та опис виконання операції
Об'єкт тестування	Інтеграційний модуль Канбан-дошки та підсистема оновлення лічильників профілю (рисунок 2.12).
Вхідні дані	ID активного завдання; поле assignedHours: 5. Початкове значення volunteerHours у профілі волонтера: 12.

Продовження табл. 2.10

Кроки виконання	1. Увійти під роллю організації-координатора. 2. Перетащити картку завдання в колонку "Виконано" (status: "done"). 3. Перевірити оновлення документа волонтера-виконавця в СУБД.
Очікуваний результат	Асинхронний тригер відправляє запит databases.updateDocument(). Значення трудомісткості додається до поточних годин волонтера. Нове значення в полі volunteerHours має становити: $12 + 5 = 17$.
Фактичний результат	Запит виконано успішно. В консолі Appwrite та в особистому кабінеті волонтера зафіксовано нове значення — 17 годин.
Статус тесту	Успішно (Passed)

Результати проведеного функціонального тестування підтвердили повну працездатність веб-платформи «UNITY+», стабільність асинхронної взаємодії React-компонентів із NoSQL базою даних Appwrite, коректність реалізації ролевої моделі безпеки та високу точність засобів автоматизації обліку волонтерської діяльності.

Висновки до розділу:

У даному розділі було проведено об'єктно-орієнтоване моделювання та технічну реалізацію веб-платформи «UNITY+». Виконано декомпозицію бізнес-процесів за допомогою діаграм SADT, Use Case, DFD та Sequence UML, а також спроектовано NoSQL архітектуру бази даних під хмару Appwrite Cloud з імплементацією Base64-кодування для медіаданих. На основі бібліотеки React та Appwrite Web SDK розроблено програмні модулі JWT-авторизації з глобальним контекстом (AuthContext), Канбан-дошки та гейміфікованого обліку годин, а також

здійснено оцінку адаптивного UI/UX дизайну та функціональне тестування критичних модулів платформи.

ВИСНОВКИ

У ході дослідження було проведено комплексну оцінку, проектування та програмну реалізацію інженерних рішень для створення сучасної клієнт-серверної веб-платформи координації волонтерської діяльності «UNITY+».

На основі отриманих результатів сформульовано такі висновки:

– Системний аналіз предметної області підтвердив, що використання розрізнених засобів комунікації призводить до деструктуризації даних, а впровадження єдиного інтернет-додатка типу SPA на базі стеку React + Appwrite дозволяє підвищити швидкість взаємодії та координації.

– Процесне моделювання за допомогою методологій SADT, DFD та UML дало змогу формалізувати життєвий цикл волонтерських ініціатив та забезпечити прозору архітектуру взаємодії акторів системи з NoSQL сервісами бази даних.

– Спроектована структура NoSQL колекцій хмарної СУБД Appwrite Cloud та об'єктна архітектура класів із реалізацією інтерфейсу IVerifiable та Base64-кодуванням забезпечила надійне та оптимізоване збереження профілів користувачів і карток потреб.

– Програмна реалізація клієнтської частини на базі бібліотеки React з використанням глобального контексту автентифікації AuthContext та модулів Канбан-дошки й гейміфікованого обліку годин підтвердила високу швидкодію, адаптивність десктопного інтерфейсу та стабільність розробленої логіки.

– Результати функціонального тестування довели відповідність розробленої веб-платформи критеріям надійності, безпеки збереження JWT-сесій та масштабованості.

Таким чином, робота підтвердила ефективність сучасних веб-технологій та хмарних BaaS-сервісів для автоматизації соціально значущих процесів і окреслила шляхи подальшого вдосконалення автоматизованих систем координації волонтерської діяльності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Чиннов К. А. Проектування архітектури сучасних веб-платформ типу SPA з використанням компонентного підходу. Прикладна інформатика. 2023. Т. 45. № 2. С. 61–72.
2. Савельєв А. О., Романов В. В. Особливості використання хмарних BaaS-платформ (Backend-as-a-Service) при розробці клієнт-серверних додатків. Комп'ютерні системи та мережі. 2024. № 12. С. 44–51.
3. Коваленко М. В., Сидоренко О. П. Оптимізація збереження медіаресурсів у документо-орієнтованих базах даних за допомогою бінарного кодування. Вісник ХНУ. Серія: Технічні науки. 2025. № 3. С. 102–111.
4. Ткаченко Д. І. Механізми забезпечення безпеки сесій у веб-додатках на основі токенів JWT. Кібербезпека та захист інформації. 2024. Вип. 8. С. 15–23.
5. Литвин В. В., Пасічник В. В. Проектування інформаційних систем та управління NoSQL базами даних у хмарних середовищах. Штучний інтелект. 2023. № 4. С. 88–95.
6. React Documentation: Official guide and API reference. URL: https://react.dev/ (дата звернення: 10.04.2026).
7. Appwrite Cloud Documentation: Docs for Databases, Auth and Storage APIs. URL: https://appwrite.io/docs (дата звернення: 12.05.2026).
8. Base64 Encoding for Data Optimization in Web Applications. Mozilla Developer Network (MDN). URL: https://developer.mozilla.org/en-US/docs/Glossary/Base64 (дата звернення: 20.05.2026).
9. NoSQL Databases: An Overview of Document-Oriented Systems and Solutions. URL: https://www.couchbase.com/resources/nosql-databases/ (дата звернення: 18.04.2026).
10. Goodliffe P. Code Craft: The Practice of Writing Excellent Code. San Francisco : No Starch Press, 2022. 560 p.

11. Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 «Комп’ютерні науки». Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.
12. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ : ДП «УкрННЦ», 2015. 26 с.
13. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні вимоги та правила складання. Київ : ДП «УкрННЦ», 2016. 16 с.
14. ДСТУ 3582:2013. Інформація та документація. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ : ДП «УкрННЦ», 2013. 23 с.

ДОДАТОК А

Ілюстративний матеріал програмної реалізації системи

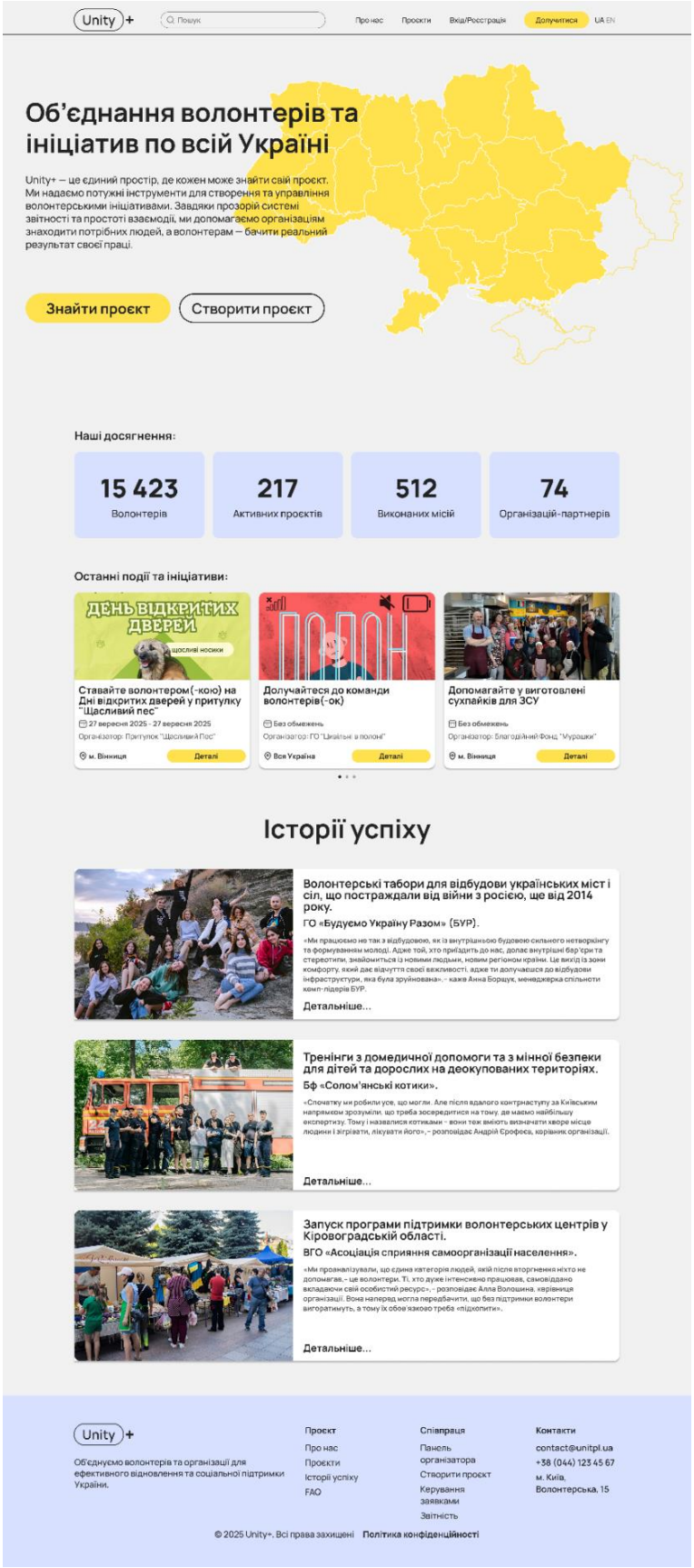

Рисунок А.1 – Спроектований UI/UX інтерфейс головної сторінки платформи UNITY+.

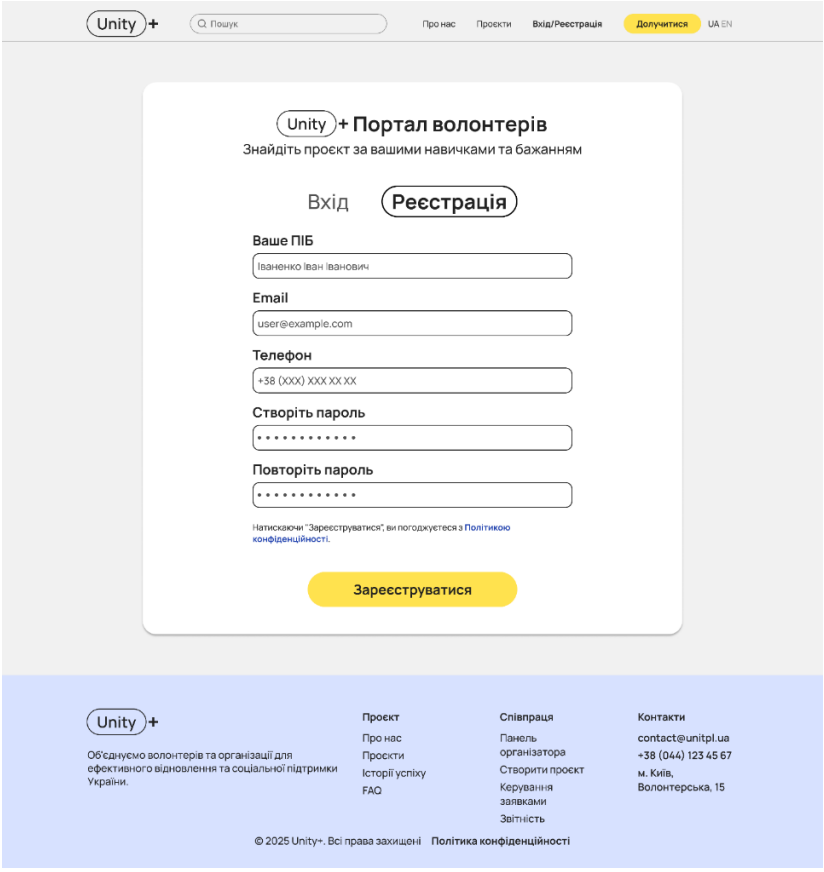

Рисунок А.2 – Екранна форма сторінки реєстрації нового волонтера.

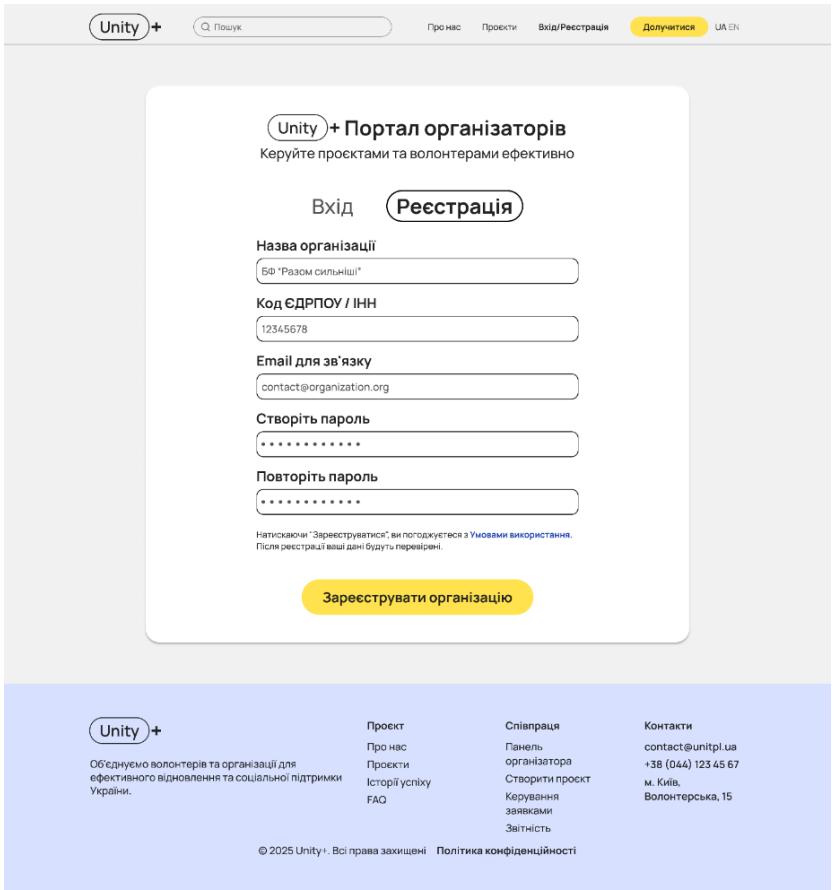

Рисунок А.3 – Екранна форма сторінки реєстрації волонтерської організації.

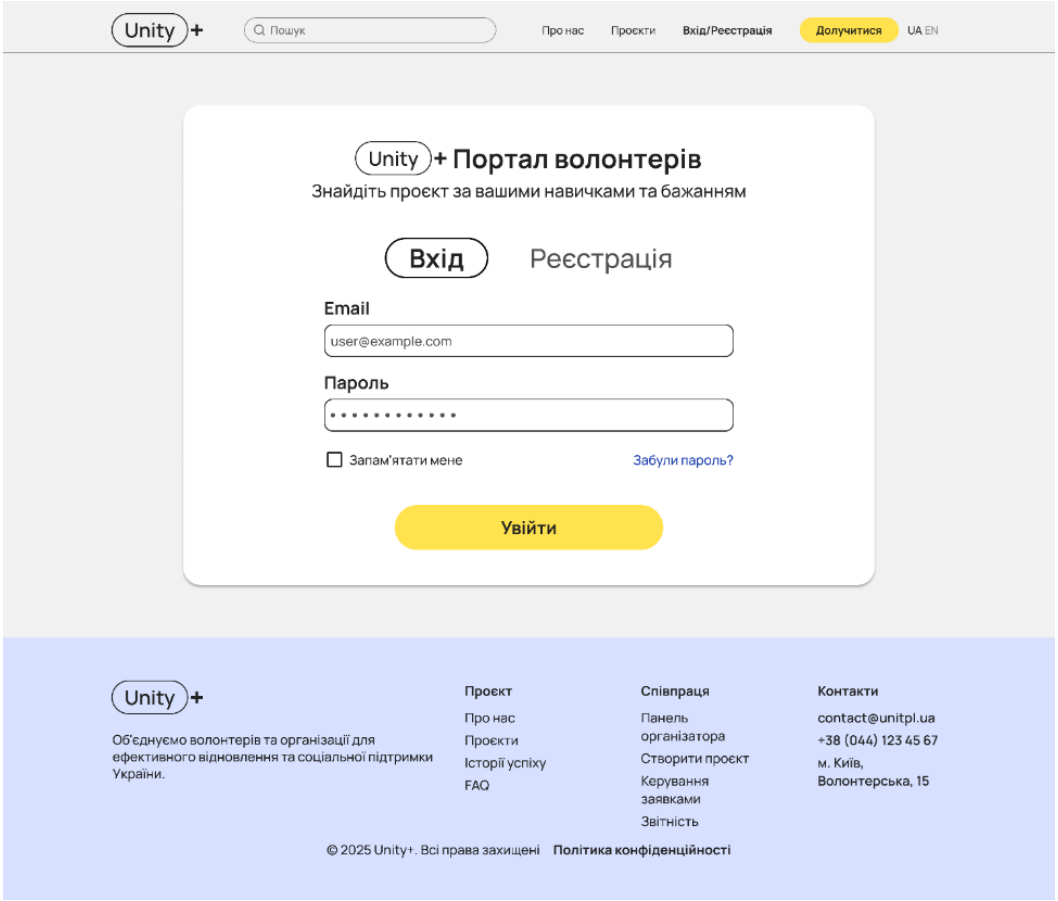

Рисунок А.4 – Екранна форма сторінки входу (авторизації) для волонтера.

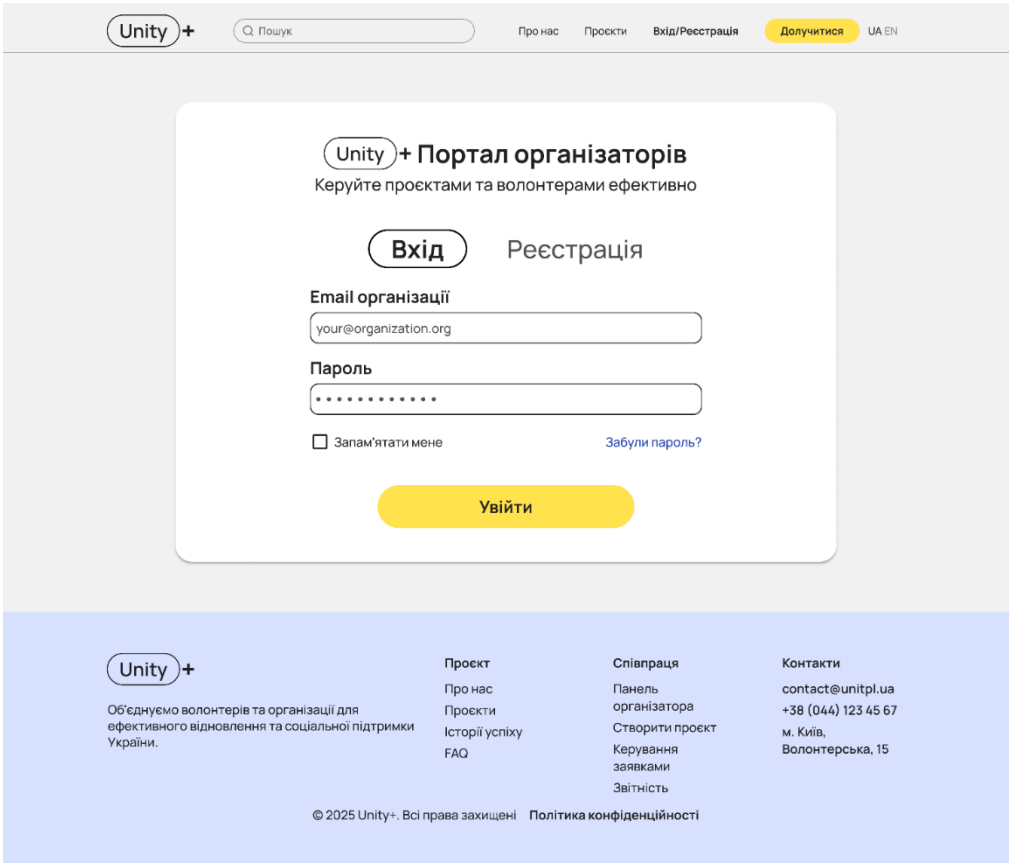

Рисунок А.5 – Екранна форма сторінки входу (авторизації) для організації.

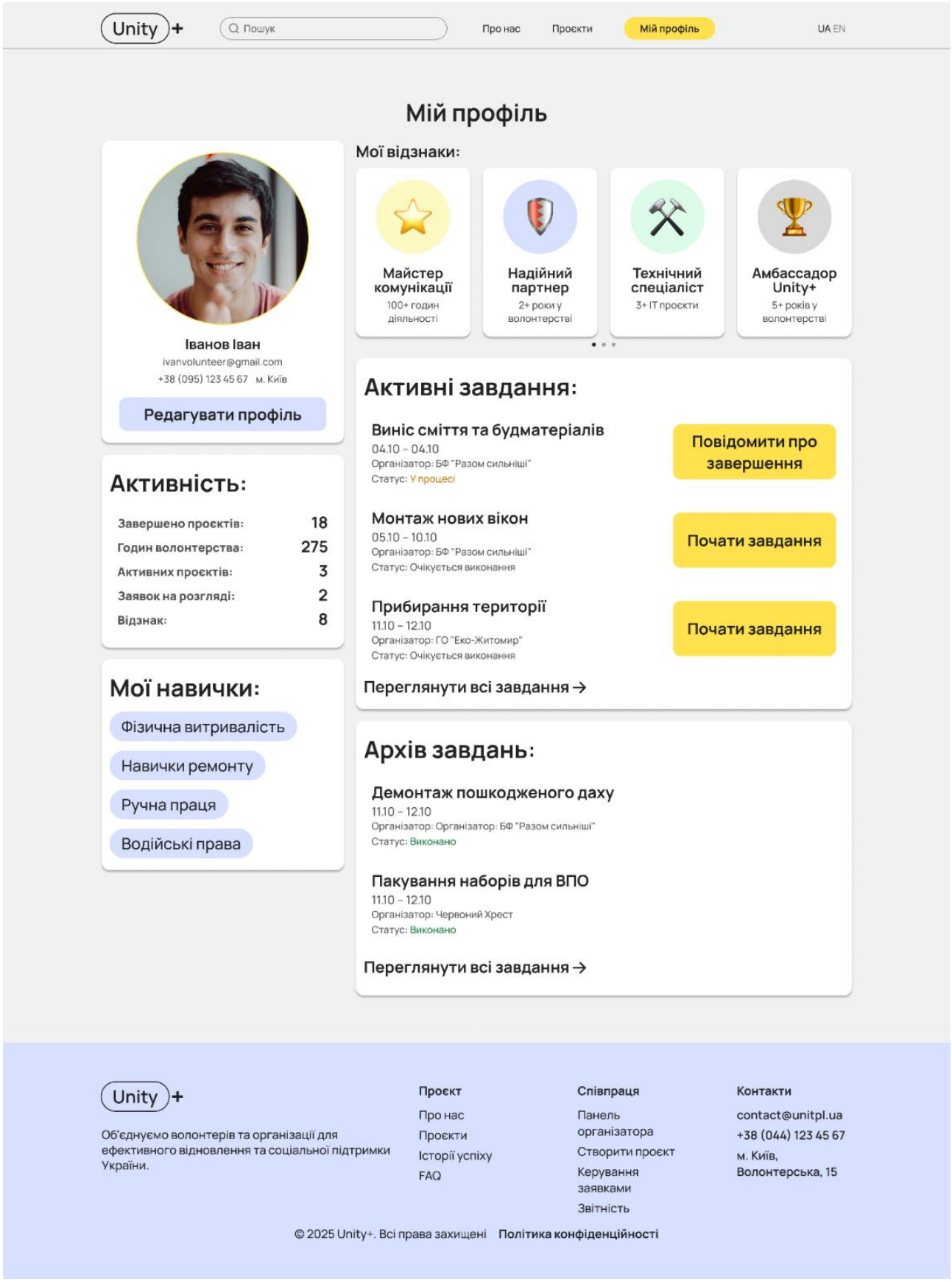

Рисунок А.6 – Інтерфейс сторінки «Мій профіль» (цифрове портфоліо) для волонтера.

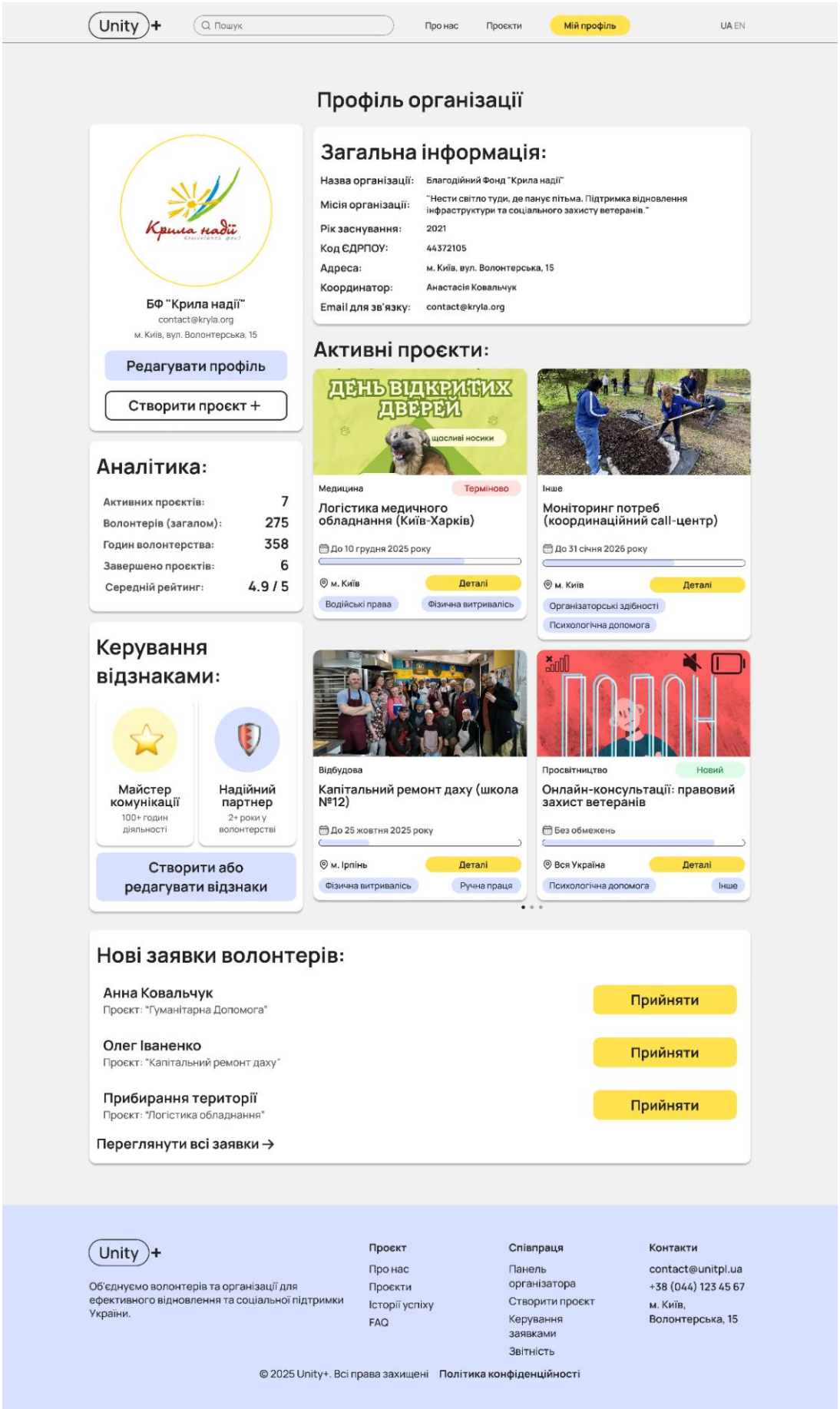

Рисунок А.7 – Адміністративний інтерфейс персонального профілю (дашборду) організації.

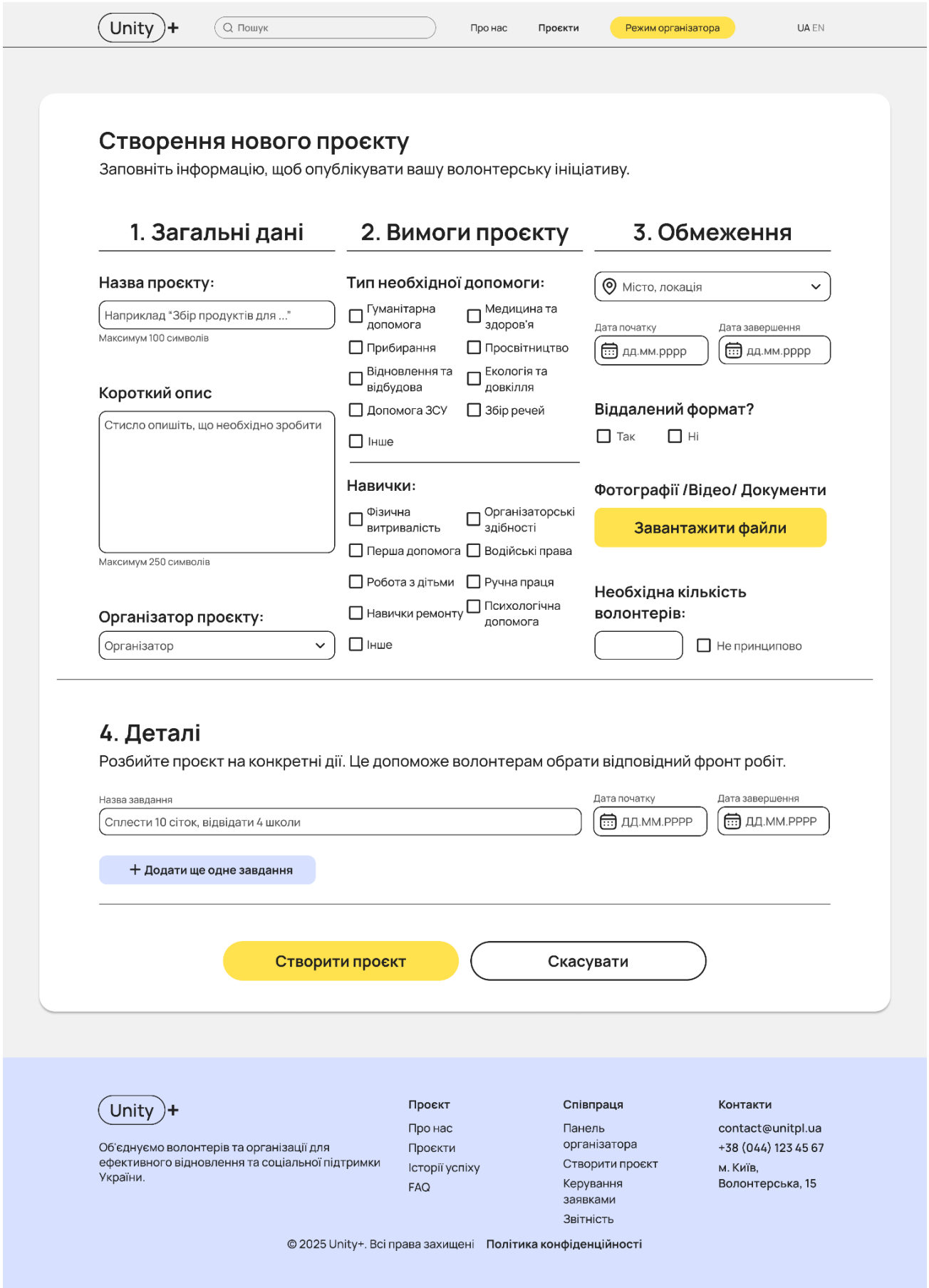

Рисунок А.8 – Інтерфейс форми покрокового створення та публікації нового проєкту.

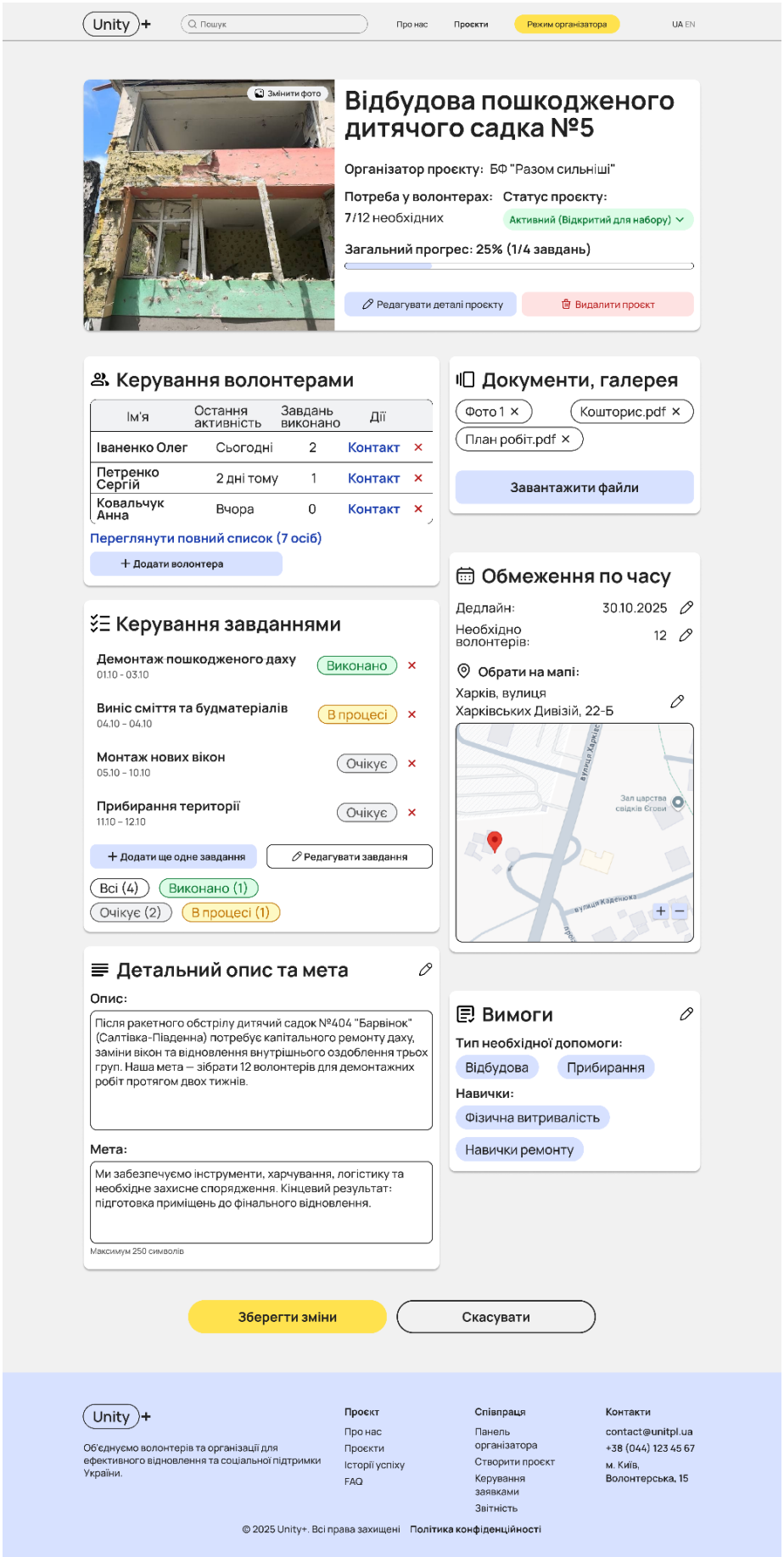

Рисунок А.9 – Екранна форма панелі оперативного управління та таск-менеджменту проєкту.

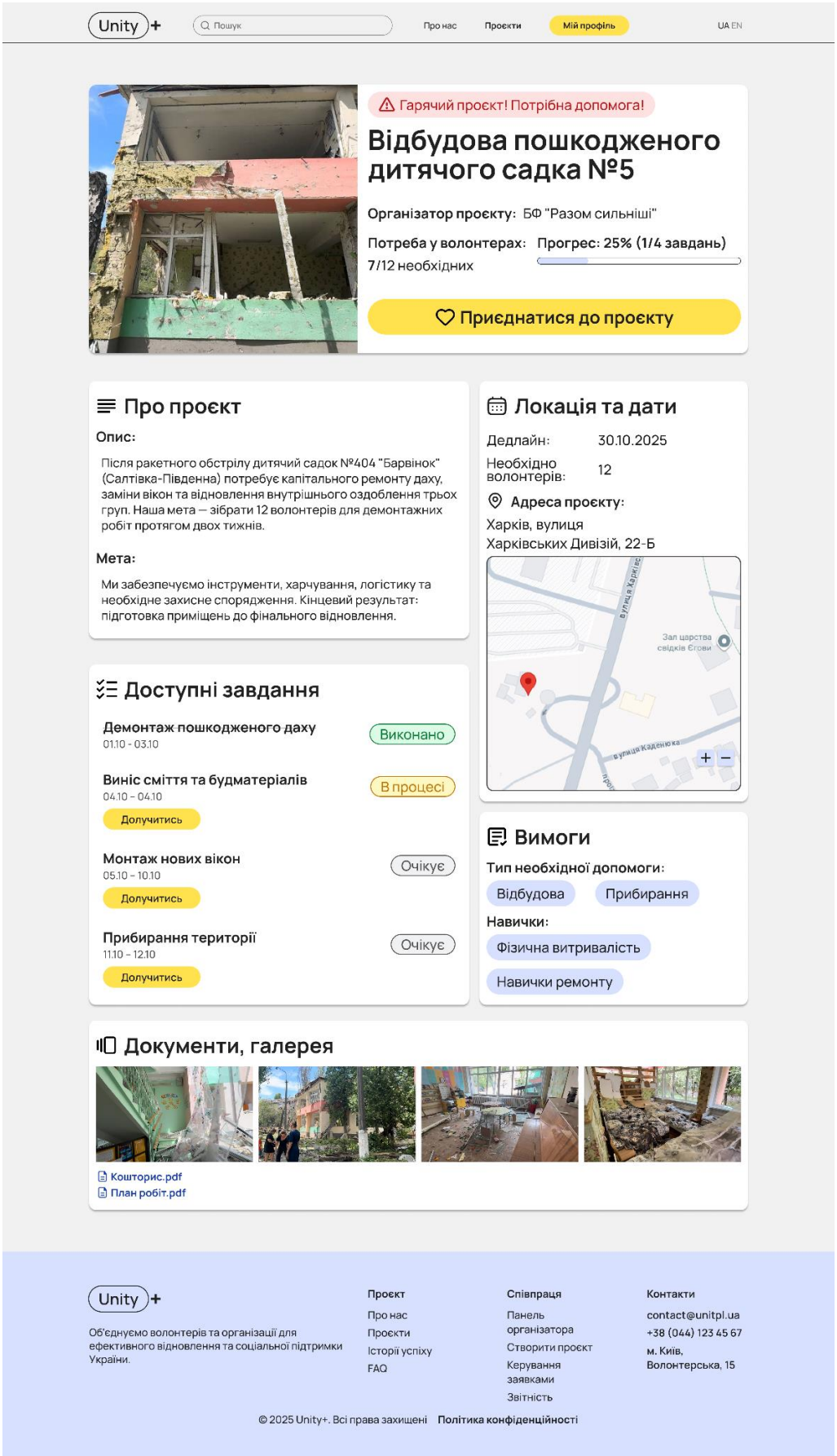

Рисунок А.10 – Інформаційний хаб сторінки волонтерського проєкту для кінцевих користувачів.

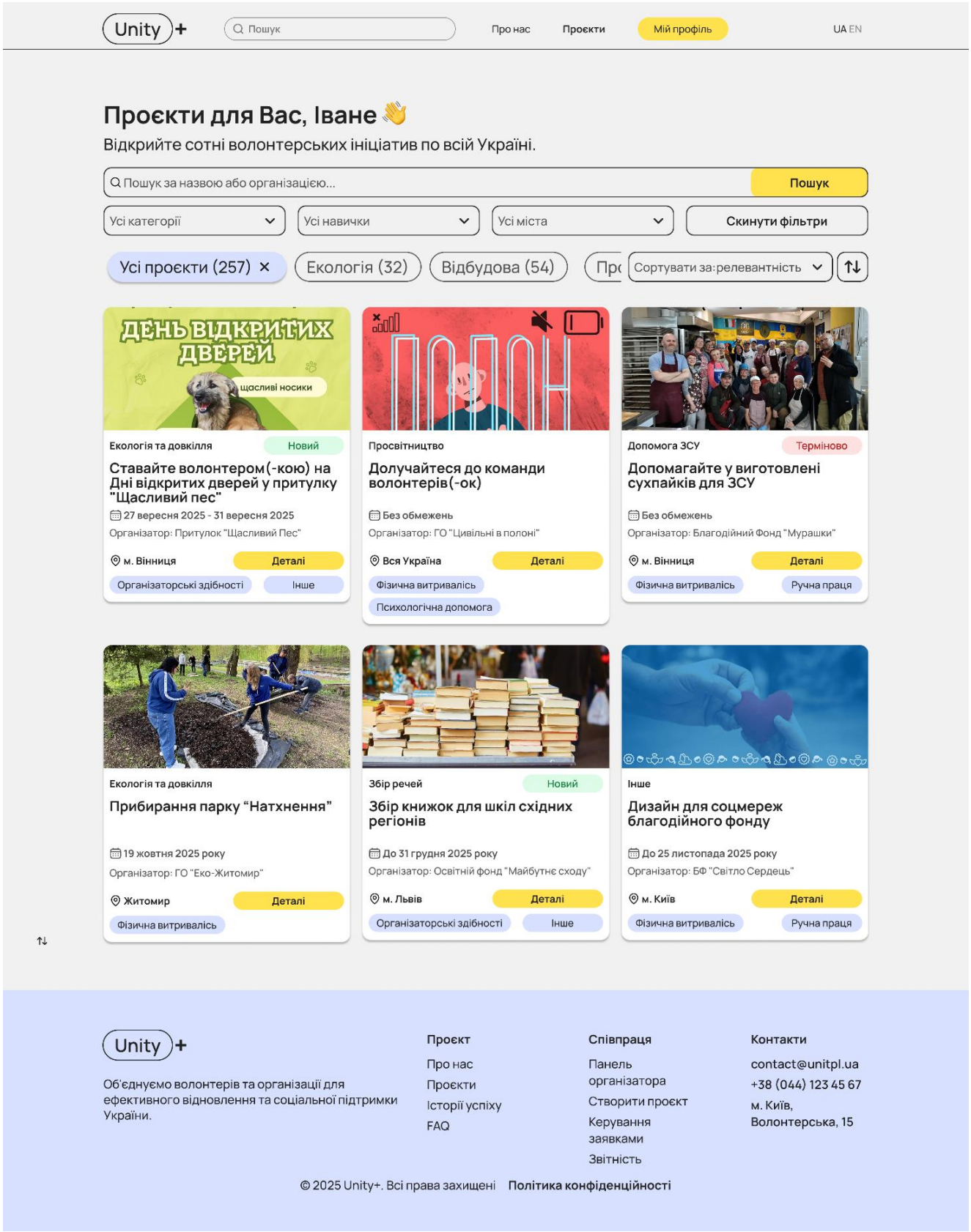

Рисунок А.11 – Інтерфейс інтелектуального модуля пошуку та багатофакторної фільтрації проєктів.

ДОДАТОК Б

Програмна реалізація обраних архітектур

Листинг Б.1 – Конфігураційний модуль ініціалізації клієнтського хмарного SDK BaaS Appwrite (src/appwrite.js)

```
import { Client, Account, Databases, Storage } from "appwrite";

const client = new Client()
  .setEndpoint("https://cloud.appwrite.io/v1")
  .setProject("69cb8a55001786de46e2");

export const account = new Account(client);
export const databases = new Databases(client);
export const storage = new Storage(client);
export { client };
```

Листинг Б.2 – Модуль кореневого маршрутизатора та конфігурації захищених клієнтських шляхів SPA-додатка (rc/App.jsx)

```
import React, { useEffect, useState } from "react";
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Navigate,
} from "react-router-dom";
import { account } from "./lib/appwrite";

// Компоненти
import Navbar from "./components/Navbar";
import Footer from "./components/Footer";

// Сторінки
import Home from "./pages/Home";
import Projects from "./pages/Projects";
import VolunteerProfile from "./pages/VolunteerProfile";
import OrgProfile from "./pages/OrgProfile";
import LoginVolunteer from "./pages/LoginVolunteer";
import LoginOrg from "./pages/LoginOrg";
import RegisterVolunteer from "./pages/RegisterVolunteer";
import RegisterOrg from "./pages/RegisterOrg";
import LoginSelect from "./pages/LoginSelect";
import RegisterSelect from "./pages/RegisterSelect";
import CreateProject from "./pages/CreateProject";
import ManageProject from "./pages/ManageProject";
import ProjectDetails from "./pages/ProjectDetail";
import OrgPublicProfile from "./pages/OrgPublicProfile";
import ForgotPassword from "./components//ForgotPassword";
import ResetPassword from "./components/ResetPassword";
function App()
{
   const [user, setUser] = useState(null);
   const [loading, setLoading] = useState(true);
   const [role, setRole] = useState(localStorage.getItem("role"));
   const checkUserSession = async () => {
      try
```

```
    {
      const currentUser = await account.get();
      console.log("👤 Поточний користувач:", currentUser);
      setUser(currentUser);

      const savedRole = localStorage.getItem("role");
      if (savedRole)
      {
        setRole(savedRole);
      }
    }
    catch (error)
    {
      console.log("ℹ️ Користувач не авторизований");
      setUser(null);
      localStorage.removeItem("role");
      setRole(null);
    }
    finally
    {
      setLoading(false);
    }
};

useEffect(() => {
  checkUserSession();
}, []);

useEffect(() => {
  const handleStorageChange = () => {
    const savedRole = localStorage.getItem("role");
    if (savedRole)
    {
      setRole(savedRole);
      account
        .get()
        .then(setUser)
        .catch(() => setUser(null));
    }
    else
    {
      setRole(null);
      setUser(null);
    }
  };

  window.addEventListener("storage", handleStorageChange);

  const interval = setInterval(() => {
    const savedRole = localStorage.getItem("role");
    if (savedRole !== role)
    {
      if (savedRole)
      {
        setRole(savedRole);
        account
          .get()
          .then(setUser)
          .catch(() => setUser(null));
      }
      else
      {
        setRole(null);
        setUser(null);
```

```
                }
            }
        }, 1000);

        return () => {
            window.removeEventListener("storage", handleStorageChange);
            clearInterval(interval);
        };
    }, [role]);

    if (loading)
        return (

            < div className = "min-h-screen flex items-center justify-center font-bold text-[32px] text-[#212121]" >

                < div className = "flex flex-col items-center gap-4" >

                    < div className = "w-12 h-12 border-4 border-[#FFE24E] border-t-transparent rounded-full animate-spin" ></
div >

                    < p > Завантаження Unity + ...</ p >

                </ div >

            </ div >
        );

    const getProfileRoute = () => (role === "org" ? "/org-profile" : "/profile");

    return (

        < Router >

            < div className = "flex flex-col min-h-screen font-['Manrope'] bg-[#f1f1f1]" >

                < Navbar user ={ user}
    userRole ={ role} />

                < main className = "flex-grow" >

                    < Routes >

                        < Route path = "/" element ={< Home />} />

                        < Route path = "/create-project" element ={< CreateProject />} />

                        < Route path = "/projects" element ={< Projects />} />

                        < Route path = "/projects/:id" element ={< ProjectDetails />} />

                        < Route path = "/manage-project/:id" element ={< ManageProject />} />

                        < Route path = "/organization/:orgId" element ={< OrgPublicProfile />} />


                        < Route
                        path = "/login"
                        element ={
                !user ? < LoginSelect /> : < Navigate to ={ getProfileRoute()} />
                        }
                        />
                        < Route
                        path = "/register"
                        element ={
```

```
        !user ? < RegisterSelect /> : < Navigate to ={ getProfileRoute()} />
            }
          />

          < Route
            path = "/login-volunteer"
            element ={ !user ? < LoginVolunteer /> : < Navigate to = "/profile" />}
          />
          < Route
            path = "/register-volunteer"
            element ={
        !user ? < RegisterVolunteer /> : < Navigate to = "/profile" />
            }
          />

          < Route
            path = "/login-org"
            element ={ !user ? < LoginOrg /> : < Navigate to = "/org-profile" />}
          />
          < Route
            path = "/register-org"
            element ={ !user ? < RegisterOrg /> : < Navigate to = "/org-profile" />}
          />

          < Route
            path = "/profile"
            element ={
        user ? < VolunteerProfile /> : < Navigate to = "/login-volunteer" />
            }
          />
          < Route
            path = "/org-profile"
            element ={ user ? < OrgProfile /> : < Navigate to = "/login-org" />}
          />

          < Route path = "*" element ={< Navigate to = "/" />} />
          < Route path = "/forgot-password" element ={< ForgotPassword />} />
          < Route path = "/reset-password" element ={< ResetPassword />} />
        </ Routes >
      </ main >
      < Footer />
    </ div >
  </ Router >
 );
}

export default App;
```

Лістинг Б.3 – Програмний модуль десктопного інтерфейсу та автентифікації сесій користувачів класу волонтер (src/pages/LoginVolunteer.jsx)

```
import React, { useState } from "react";
import { useNavigate, Link } from "react-router-dom";
import { account, databases } from "../lib/appwrite";
export default function LoginVolunteer() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  const DATABASE_ID = "69cb8d970013413c3913";
```

```
const COLLECTION_ID = "users";

const handleLogin = async (e) => {
  e.preventDefault();
  setError("");
  setLoading(true);

  try {
    await account.createEmailPasswordSession(email, password);
    console.log("✅ Сесія створена")
    const user = await account.get();
    console.log("👤 Користувач:", user.$id);
    const userDoc = await databases.getDocument(
      DATABASE_ID,
      COLLECTION_ID,
      user.$id
    );
    console.log("📄 Документ користувача:", userDoc);
    if (userDoc.role === "volunteer") {
      localStorage.setItem("role", "volunteer");
      localStorage.setItem("userId", user.$id);
      localStorage.setItem(
        "userData",
        JSON.stringify({
          name: userDoc.name,
          city: userDoc.city,
          phone: userDoc.phone,
          email: userDoc.email,
          role: userDoc.role,
        })
      );
      console.log("✅ Дані збережено в localStorage");
      window.location.href = "/profile";
    } else {
      setError(
        `Цей акаунт зареєстрований як ${
          userDoc.role || "організація"
        }, а не волонтер.`
      );
      await account.deleteSession("current");
      setLoading(false);
    }
  } catch (err) {
    console.error("❌ Помилка логіну:", err);
    if (err.code === 401) {
      setError("Невірний email або пароль.");
    } else if (err.code === 404) {
      setError("Користувача з таким email не знайдено.");
    } else {
      setError("Помилка входу. Спробуйте пізніше.");
    }
    setLoading(false);
  }
}
return (
  <div className="min-h-screen bg-[#f1f1f1] flex flex-col items-center py-[100px] font-['Manrope']">
    <div className="w-[1000px] bg-white rounded-[32px] shadow-lg p-[80px] flex flex-col items-center">
      <div className="flex flex-row items-center justify-center mb-[4px] gap-4">
        <img
          src="/logo.svg"
          alt="Unity+"
          className="w-[168px] h-[58px] object-contain"
        />
        <h1 className="font-bold text-[48px] text-[#212121] leading-tight">
```

```
            Портал волонтерів
          </h1>
        </div>
        <h1 className="text-[32px] font-bold text-[#212121] mb-12 text-center">
          Знайдіть проєкт за вашими навичками та бажанням
        </h1>
        <div className="flex gap-12 mb-16 text-[48px]">
          <button className="text-[#212121] border-b-4 border-[#FFE24E] pb-2 font-bold transition-all">
            Вхід
          </button>
          <Link
            to="/register-volunteer"
            className="text-[#515151] hover:text-[#212121] transition-all"
          >
            Реєстрація
          </Link>
        </div>
        {error && (
          <div className="bg-red-50 border border-red-200 rounded-2xl p-4 mb-6 w-[748px]">
            <p className="text-red-500 text-[20px] font-bold text-center">
              {error}
            </p>
          </div>
        )}
        <form onSubmit={handleLogin} className="space-y-[30px] w-[748px]">
          <div>
            <label className="block mb-2 font-bold text-[32px] text-[#212121]">
              Email
            </label>
            <input
              type="email"
              required
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              placeholder="user@example.com"
              className="w-full h-[60px] px-6 bg-white border-2 border-slate-900 rounded-2xl text-[24px] font-medium text-
[#515151] outline-none focus:border-[#FFE24E] transition-all"
            />
          </div>
          <div>
            <label className="block mb-2 font-bold text-[32px] text-[#212121]">
              Пароль
            </label>
            <input
              type="password"
              required
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              placeholder="••••••••••••"
              className="w-full h-[60px] px-6 bg-white border-2 border-slate-900 rounded-2xl text-[24px] outline-none
focus:border-[#FFE24E] transition-all"
            />
          </div>
          <div className="flex items-center justify-between w-full mt-6">
            <div className="flex items-center gap-4">
              <input
                type="checkbox"
                id="remember-vol"
                className="w-6 h-6 border-2 border-slate-900 rounded cursor-pointer accent-[#FFE24E]"
              />
              <label
                htmlFor="remember-vol"
                className="text-[20px] font-medium text-[#212121] cursor-pointer"
              >
```

```
                Запам'ятати мене
              </label>
            </div>
            <Link
              to="/forgot-password"
              className="text-[20px] text-[#1e40af] hover:font-bold transition-all"
            >
              Забули пароль?
            </Link>
          </div>
          <div className="flex justify-center mt-12">
            <button
              disabled={loading}
              type="submit"
              className={`w-[490px] h-[82px] bg-[#FFE24E] rounded-full font-bold text-[32px] text-[#212121] shadow-md
hover:scale-105 transition-all flex items-center justify-center ${
                loading ? "opacity-50 cursor-not-allowed" : ""
              }`}
            >
              {loading ? (
                <div className="flex items-center gap-2">
                  <div className="w-6 h-6 border-2 border-[#212121] border-t-transparent rounded-full animate-spin"></div>
                  Вхід...
                </div>
              ) : (
                "Увійти"
              )}
            </button>
          </div>
        </form>
      </div>
    </div>
  );
}
```