Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра

(бакалавра, спеціаліста, магістра)

Студента Білашука Степана Олексійовича
(ПІБ)

академічної групи 126-20-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

за освітньо-професійною програмою «Інформаційні системи та технології»
(офіційна назва)

на тему Розробка адміністративної частини інформаційної системи "Каса взаємодопомоги"
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікавційної роботи	проф. Коротенко Г.М.			
розділів:				

Рецензент	доц. Ширін А.Л.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри

____інформаційних технологій____
___та комп'ютерної інженерії____
(повна назва)

_____ __Гнатушенко В.В._____
(підпис) (прізвище, ініціали)

«_____»_____2024_року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____бакалавр_____
(бакалавра, спеціаліста, магістра

студенту _____Білашуку С.О._____ **академічної групи** ____126-20-1_____
(прізвище та ініціали) (шифр)

спеціальності _____126 « Інформаційні системи та технології »_____

за освітньою-професійною програмою _____

_____«Інформаційні системи та технології»_____

на тему ___Розробка адміністративної частини інформаційної системи _____

_____"Каса взаємодопомоги"_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024 р. № 469-с

Розділ	Зміст	Термін виконання
Розділ 1	Ознайомлення з матеріалами сайтів, літературою та виконання пошуку технологій і формування відповідних рішень	01.02.2024 – 30.03.2024
Розділ 2	Реалізація проектних рішень	01.04.2024 – 20.06.2024

Завдання видано _____ _____Коротенко Г.М._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____01.02.2024 р._____

Дата подання до екзаменаційної комісії ____02.07.2024 р.____

Прийнято до виконання _____ _____Білашук С.О._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 83 сторінки, 28 рисунків, 1 таблиця, 3 додатка, 19 джерел.

Об’єкт розробки: адміністративна частина інформаційної системи «Каса взаємодопомоги».

Мета кваліфікаційної роботи: розробка програмного забезпечення адміністративної частини інформаційної системи «Каса взаємодопомоги».

У першому розділі звертається увага на актуальність проблеми, проводиться аналіз існуючих аналогів інформаційної системи «Каса взаємодопомоги» і визначаються мета та задачі кваліфікаційної роботи. Крім того, розглядаються технології, які будуть використовуватись для реалізації проекту.

Другий розділ присвячений проектуванню адміністративної частини інформаційної системи «Каса взаємодопомоги», де наведено діаграми, створені з використанням нотації UML, для опису його функціональності та взаємодії з користувачем.

У наступному розділі детально описується реалізація застосунку. Виконується розробка самого вебдодатку адміністративної частини «Каси взаємодопомоги» з використанням інструментів Nuxt3 та Vue.js, і мов JS та TypeScript. Також описується порядок взаємодії з графічним інтерфейсом користувача.

Результатом проведеної роботи є створений вебзастосунок адміністративної частини інформаційної системи «Каса взаємодопомоги». Цей вебзастосунок дозволяє здійснювати управління життєвим циклом благодійних програм та обліковими записами учасників руху взаємодопомоги.

Ключові слова: АДМІНІСТРУВАННЯ, КАСА ВЗАЄМОДОПОМОГИ, ВЕБЗАСТОСУНОК, ФРЕЙМВОРК, NUXT, VUE.JS, JS, TYPESCRIPT.

ABSTRACT

Explanatory note: 83 pages, 28 fig., 1 tab, 3 appendices, 19 sources.

The object of development: an administrative part of the information system "Mutual Assistance Fund".

The purpose of the qualification work: development of software for an administrative part of the information system "Mutual Assistance Fund".

The first section draws attention to the relevance of the problem, analyzes the existing analogs of the information system "Mutual Assistance Fund", and defines the purpose and objectives of the qualification work. In addition, the technologies that will be used to implement the project are considered.

The second section is devoted to the design of an administrative part of the information system "Mutual Assistance Fund", where diagrams are provided using the UML notation to describe its functionality and user interaction.

The next section describes the implementation of the web application in detail. The development of the web application of the administrative part of the "Mutual Assistance Fund" using Nuxt3 and Vue.js tools, and the JS and TypeScript languages is done. The order of interaction with the graphical user interface is also described.

The result of the work is the created web application of the administrative part of the information system "Mutual Assistance Fund". This web application allows managing the life cycle of charitable programs and the accounts of members of the mutual aid movement.

Keywords: ADMINISTRATION, MUTUAL ASSISTANCE FUND, WEB APPLICATION, FRAMEWORK, NUXT, VUE.JS, JS, TYPESCRIPT.

ЗМІСТ

ВСТУП

Фонд взаємодопомоги — це продукт розподілу ризиків, який базується на взаємній підтримці рівних учасників. Кожен учасник щомісяця сплачує внесок у нейтральний пул для збору капіталу, який використовується, коли учасник потребує підтримки для покриття своїх медичних витрат. Ця схема призначена для зменшення фінансового тягаря тих, хто отримав травму або хворобу, надаючи їм фінансову допомогу для покриття дорогих рахунків за медичну допомогу та відновлення здоров’я.

У рамках даної кваліфікаційної роботи було поставлено завдання розробити адміністративну частину інформаційної системи «Каса взаємодопомоги». «Каса взаємодопомоги» є інформаційною системою, яка автоматизує процес функціонування руху взаємодопомоги.

Цей проект має на меті розробити зручний та ефективний механізм адміністрування «Каси взаємодопомоги». Користувачі зможуть керувати життєвим циклом благодійних програм фонду, обробляти заявки від учасників програм. Для ролі супер адміністратора передбачено функціонал для керування обліковими записами користувачів інформаційної системи.

У процесі розробки адміністративної частини інформаційної системи буде використана сучасна вебтехнологія, що дозволить створити потужний та функціональний інструмент для адміністраторів каси взаємодопомоги. Також будуть застосовані методи авторизації, щоб забезпечити розділення доступу до чутливих даних.

Основною метою цієї розробки є створення зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам швидко оброблювати звернення учасників фонду, керувати рухом грошових коштів та зберігати інформацію про благодійні програми.

Розроблений вебзастосунок буде враховувати сучасні вимоги до дизайну та функціональності, що забезпечить приємний та зручний досвід користувачів. Додатково будуть застосовані методи валідації та перевірки

введених даних, щоб уникнути помилок та забезпечити коректність інформації, яку надсилає користувач.

Очікується, що розроблена адміністративна частина інформаційної системи «Каса взаємодопомоги» покращить доступність та зручність процесу функціонування руху фінансової підтримки, сприяючи розвитку благодійних ініціатив членів громади.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Взаємодопомога є інструментом, який люди використовували протягом століть для підтримки своїх громад. Це широка історична практика, яка виходить за межі державних програм допомоги та відкидає некомерційну модель, яка може бути неефективною та часто передбачає відповідні вимоги для заповнення усунення прогалин у державних службах і створення процвітаючих спільнот. Взаємодопомога є колективною координацією для задоволення потреб один одного, як правило, через усвідомлення того, що існуючі системи не задовольнять їх. Взаємодопомога має багато форм, але всі вони ґрунтуються на ідеї, що суспільні системи зміняться не покладаючись на «символічні акти», але шляхом створення мереж спільнот, які підтримують одне одного. Це спільне зобов'язання задовольняти потреби одне одного через канали, які не залежать від держави. Взаємодопомога може здійснюватися через пожертвування коштів низовим організаціям чи проектам [1].

Найпоширеніший і найважливіший час для взаємодопомоги – це під час широкомасштабних громадських криз, стихійних лих, військових дій. Пандемія COVID-19 стимулювала участь українців у діяльності взаємодопомоги, але найбльшої актуальності ця ініціатива набула з початком війни, коли складний суспільний стан (перебої з постачанням електроенергії та інших комунальних послуг, постійна загроза життю та здоров'ю) поєднався важким становищем окремих громадян, які втратили майно, заощадження, стале джерело прибутку, зазнали травм чи ушкоджень.

Однією з ключових ідей взаємодопомоги є значущість потреб кожної людини, і що одержувачі допомоги не повинні підтверджувати відповідність вимогам або показувати, як вони використовують кошти, які вони отримують.

На початковому етапі свого розвитку каса взаємодопомоги може залучати учасників шляхом об’єднання людей за місцем роботи, навчання, проживання, або наявних соціальних зв’язків. Функціонування ініціативи невеликого масштабу підтримується за допомогою груп чи каналів у месенжджерах та соціальних мережах [2]. Однак з часом, зростанням капіталу і розширенням кола учасників, така організація відчуває потребу у програмному сервісі, веборієнтованому або мобільному застосунку. Це стає необхідною умовою для поліпшення сервісу та надання послуг в сучасному світі. Розроблений застосунок повинен надавати потужний та ефективний функціонал для всіх осіб, які залучені до каси взаємодопомоги, як для учасників ініціативи, так і для персоналу, що здійснює адміністрування та керування проектом. Адміністративна частина інформаційної системи, що розробляється у даній кваліфікаційній роботі, спрямована на підтримку діяльності працівників фонду, і призначена спеціально для конкретної організації "Каса взаємодопомоги".

1.2 Аналіз аналогів інформаційної системи «Каса взаємодопомоги»

Перед початком проектування застосунку необхідно провести пошук і аналіз існуючих аналогічних програмних продуктів, що належать до категорії інформаційних систем керування фондами взаємодопомоги. В українському сегменті інтернету було знайдено декілька вебсайтів, що мають подібне призначення, серед них:

1. Каса взаємодопомоги організації «Ліга майстрів» [3].

«Ліга майстрів» - це спільнота професіоналів сфери будівництва України. На сайті організації є розділ «Каса взаємодопомоги», у якому розміщена загальна інформація про ініціативу, розміри внесків, умови участі (рис. 1.1). Також доступний перегляд щомісячної звітності про використання коштів каси.

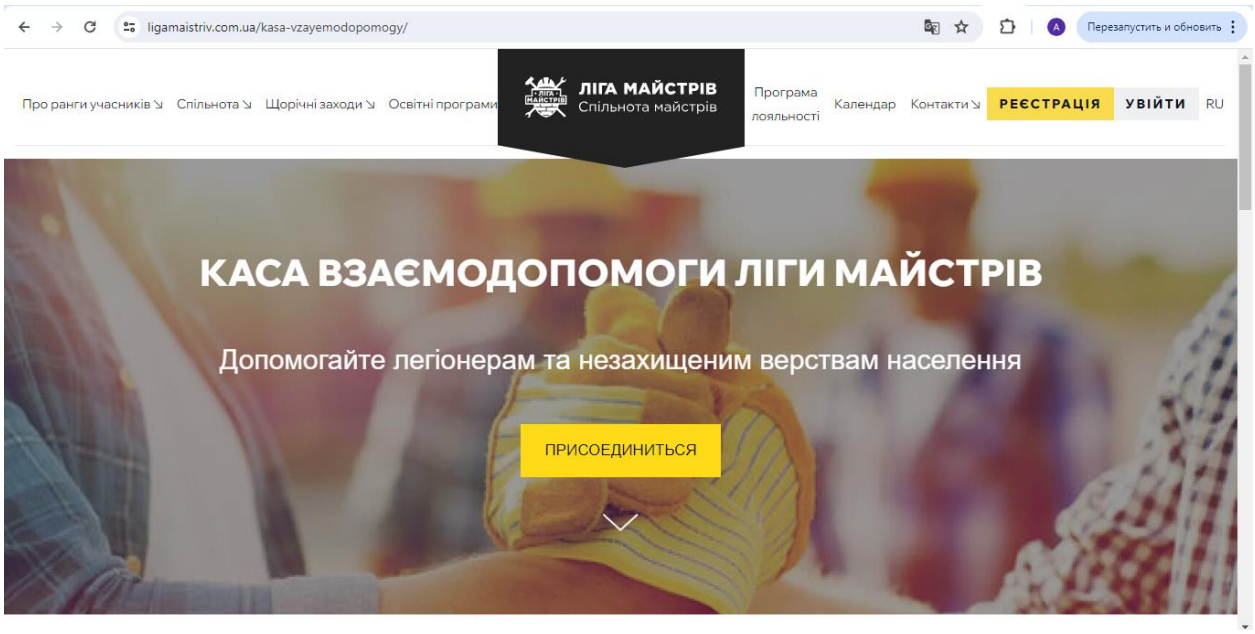

Рис. 1.1. Каса взаємодопомоги «Ліги майстрів»

Для приєднання до проекту необхідно натиснути для кнопку «Приєднатися», яка знаходиться в нижній частині екрану (рис. 1.2).

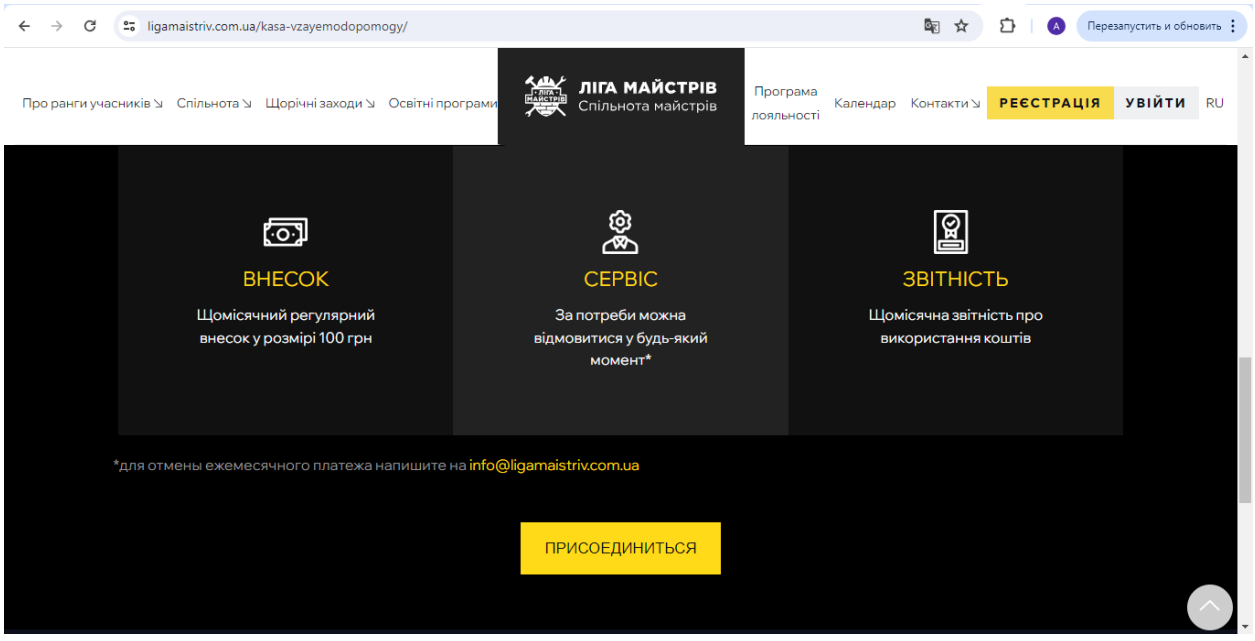

Рис. 1.2. Функція приєднання до проекту

Проте участь у касі взаємодопомоги можлива лише для дійсних членів «Лігі майстрів», яка є закритою професійною спільнотою із платним

членством, тому даний вебсайт не може розглядатися як універсальний інструмент для підтримки діяльності ініціатив взаємодопомоги.

2. «Допомога справою» [4]

Це вебсайт благодійного фонду м. Запоріжжя, який реалізує багато благодійних програм, зокрема, в сфері медицини та здоров'я, екології, соціального захисту, та інші (рис. 1.3).

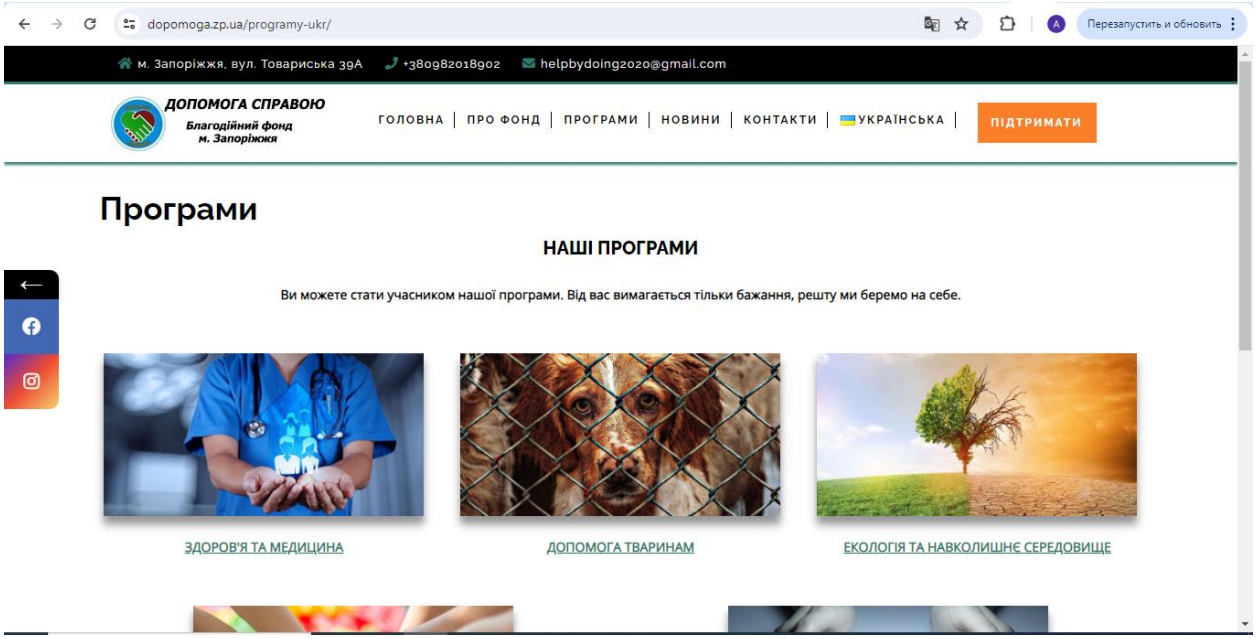

Рис. 1.3. Перелік благодійних програм

За допомогою вебсайту можна зробити разову пожертву, або настроїти щомісячний платіж на будь-яку суму (рис. 1.4).

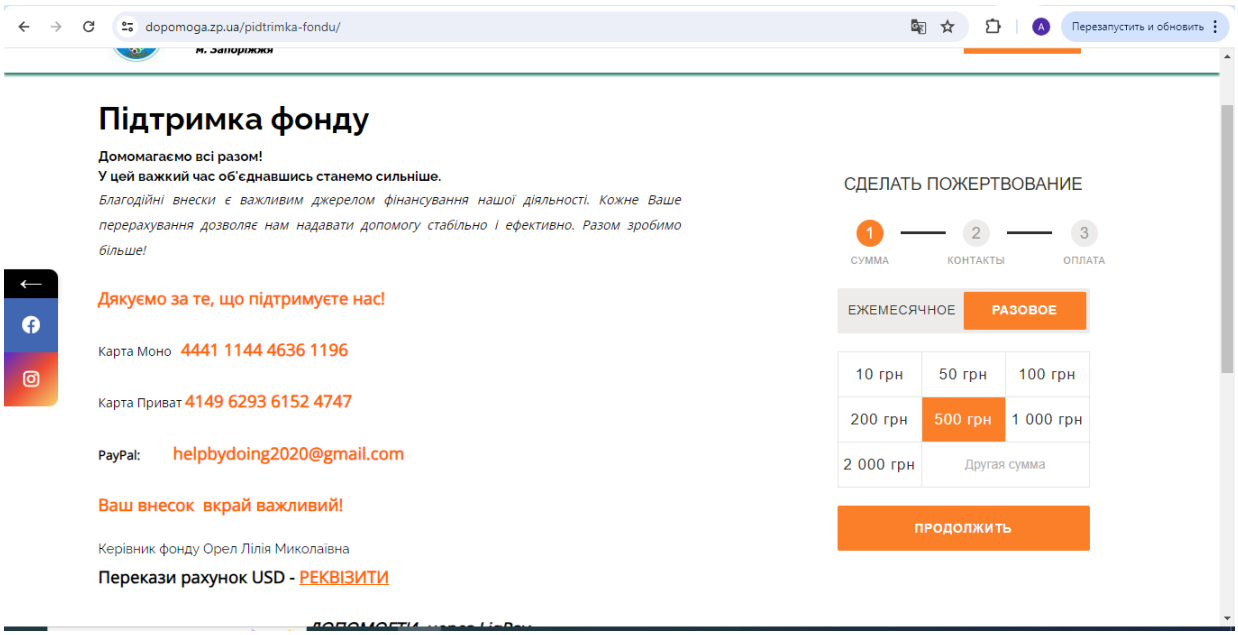

Рис. 1.4. Сторінка платежів

Також є можливість подати заявку на отримання допомоги, заповнивши форму на сторінці «Отримати підтримку» (рис. 1.5). Для подання запиту необхідно вказати ім’я, контактні дані, та описати ситуацію, яка є причиною прохання про допомогу.

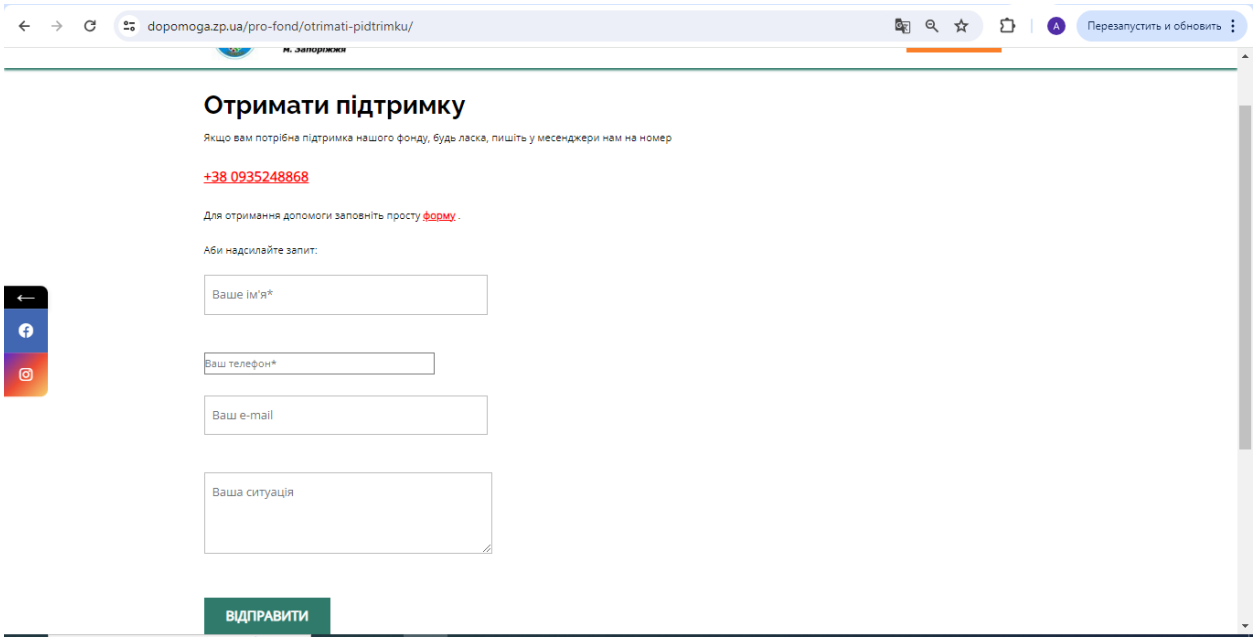

Рис. 1.5. Заява на отримання допомоги

На сторінці «Звіти» можна подивитися відео-звіти по зеваршеним програмам.

Даний вебзастосунок надає інструменти для переказу коштів та виконання запитів на отримання допомоги, але він фактично не є спільною касою, бо внестки є благодійними пожертвами, і не носять обов’язковий чи систематичний характер.

Розглянуті вебсайти реалізують окремі інструменти керування благодійними програмами, такі як вступ до спільноти («Ліга майстрів»), або подання заяв на отримання допомоги («Допомога справою»), але жоден з них не реалізує повною мірою концепцію каси взаємодопомоги. Таким чином, можна зробити висновок, що на даний момент не існує спеціалізованих програмних продуктів, призначених безпосередньо для управлінням процесом функціонування руху взаємодопомоги. Тому актуальною задачею є

розробка адміністративної частини інформаційної системи «Каса взаємодопомоги», яка б забезпечивала потужний та функціональний інструмент для адміністраторів спільного некомерційного фонду взаємодопомоги.

1.3 Мета та задачі адміністративної частини інформаційної системи «Каса взаємодопомоги»

Мета даної кваліфікаційної роботи полягає у розробці адміністративної частини інформаційної системи «Каса взаємодопомоги». Цей застосунок спростить процесс управління касою взаємодопомоги. Він надасть можливість адміністраторам фонду швидко та ефективно обробляти запити учасників благодійних програм.

Також, враховуючи основну мету цього застосунку, адміністративна частина «Каси взаємодопомоги» буде використовуватись для керування рухом коштів та життєвим циклом благодійних програм фонду для задовільнення потреб його членів.

Розроблений застосунок має забезпечувати виконання наступних функцій:

- перегляд списку благодійних програм;
- додавання нових програм;
- редагування інформації про наявні програми;
- перегляд списку учасників благодійних програм, облікові записи яких потребують обробки;
- пошук облікових записів учасників благодійних програм;
- обробка інформації про облікові записи учасників програм;
- перегляд файлів, завантажених учасниками програм;
- перегляд запитів від учасника благодійних програм та можливість обробки цих запитів;
- перегляд виплат за запитами учасника;

- перегляд запитів від учасників загалом по фонду з можливістю фільтрації;
- перегляд інформації про рух коштів загалом по фонду;
- виконання всіх операцій, окрім входу в систему, у якості авторизованого користувача системи;

За умови авторизації у якості суперадміністратора:

- перегляд списку користувачів вебзастосунка;
- можливість додавання нових користувачів;
- можливість редагування інформації про користувача, зміни пароля та блокування користувача.

Адміністративна частина «Каси взаємодопомоги» має бути розроблена відповідно до сучасних вимог дизайну та функціональності. Програмне забезпечення повинно підтримувати виконання таких дій:

- Реагування на дії користувача в онлайн режимі;
- Авторизація;
- Автозбереження будь-яких змін, здійснених користувачем;
- Здійснення максимально швидкої навігації по сайту;
- Коректна візуалізація елементів вебзастосунку.

Для досягнення поставленої мети і відповідно до вимог, необхідно виконати наступні завдання:

- здійснити експертний аналіз предметної області;
- вибрати необхідні інструменти для створення вебзастосунку;
- розробити артітектуру програмного продукту;
- розробити відповідний функціонал вебзастосунку;
- провести тестування вебзастосунку.

Щоб забезпечити ефективну реалізацію проекту та виконання всіх вимог, необхідно мати детальне технічне завдання. Це є необхідною умовою для успішної реалізації проекту та досягнення поставлених цілей.

1.4 Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра було розглянуто поняття інформаційних систем керування фондами взаємодопомоги, а також вивчено їх особливості та складові. Було проведено аналіз двох вебсайтів благодійних ініціатив.

В результаті огляду цих сайтів були виявлені наступні недоліки:

– відсутність інструментів для управління обліковими записами користувачів;

– немає керування процесом виплат;

– не реалізовано повною мірою концепцію каси взаємодопомоги як спільного грошового фонду, який виділяю кошти за запитами учасників.

Вищезазначені недоліки обумовлюють необхідність розробки адміністративної частини інформаційної системи «Каса взаємодопомоги», яка б забезпечивала потужний та функціональний інструмент для адміністраторів спільного некомерційного фонду взаємодопомоги.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АДМІНІСТРАТИВНОЇ ЧАСТИНИ ІНФОРМАЦІЙНОЇ СИСТЕМИ «КАСА ВЗАЄМОДОПОМОГИ»

2.1 Функціонал та алгоритми функціонування застосунку

Процес проектування є однією з ключових фаз у створенні вебзастосунку, під час якого визначаються принципи побудови архітектури та алгоритми функціонування програми [5].

Поведінка програми визначається через взаємодію об'єктів, які утворюють її, і представляється за допомогою різних типів діаграм, таких як діаграма комунікації, діаграма послідовності, діаграма діяльності та діаграма кінцевого автомата [6].

Діаграма варіантів використання є динамічною діаграмою або діаграмою поведінки в рамках UML [5]. Вона використовується для моделювання функціональності системи, зосереджуючись на взаємодії акторів і сценаріїв використання. Варіанти використання представляють собою набір дій та функцій, які система повинна виконувати. "Актори" в цьому контексті можуть бути людьми, організаціями або програмними засобами, які виконують певні ролі в системі.

Для кожного актора визначаються сценарії їх взаємодії з вебзастосунком. Для створення діаграми варіантів використання застосунку були визначені наступні актори [6]:

– Адміністратор, який представляє працівника каси взаємодопомоги і відповідає за обробку запитів учасників та управління благодійними програмами;

– Суперадміністратор, який представляє адміністратора інформаційної системи, та відповідає за створення та керування обліковими записами користувачів застосунку;

– Учасник, який представляє рядового користувача – учасника ініціативи «Каса взаємодопомоги».

Після визначення списку акторів, наступним кроком є формування переліку сценаріїв, який включає:

– Додавання нової програми;
– Редагування запису програми;
– Підтвердження учасника;
– Пошук наявного учасника;
– Редагування запису учасника;
– Редагування запитів;
– Перегляд запитів;
– Підтвердження або відхилення запиту;
– Перегляд файлів запиту;
– Додавання виплат за запитом;
– Додавання інформації про виплати або поповнення;
– Додавання нового користувача;
– Редагування наявного користувача;
– Зміна паролю користувача;
– Блокування користувача.

На рисунку 2.1 наведена діаграма варіантів використання для адміністративної частини «Каси взаємодопомоги».

Для кожного варіанту використання створюється діаграма послідовності, яка відображає процес взаємодії елементів системи, тут вони також називаються акторами. У діаграмі послідовності для варіанту використання «Створення запиту» присутні три актори: Учасник, Адміністратор, та Застосунок. Ці актори є ініціаторами взаємодії в системі.

Рис. 2.1. Діаграма варіантів використання

Діаграма містить вісім повідомлень, які є закінченими фрагментами інформації, що передаються від одного об'єкта до іншого:

1. Авторизація;
2. Ідентифікація;
3. Вибір програми;
4. Надсилання запиту на підтвердження;
5. Перевірка поданих персональних файлів;
6. Перевірка факту вибору програми;
7. Заповнення полів;
8. Підтвердження.

Ці повідомлення не лише містять певну інформацію, а й очікують її виконання з боку отримувача. Вони зображені горизонтальними стрілками, які з'єднують об'єкти лініями життя. Лінії життя можуть бути прямими пунктирними лініями, які виходять з усіх об'єктів паралельно від осі часу,

або фокусами управління - вертикальними прямокутниками, що відображають активність об'єктів. Початок фокусу управління відповідає початку активності, а завершення - її закінченню. Якщо стрілка повідомлення виходить з фокусу управління актора, це означає, що цей актор відправляє повідомлення іншому актору. Якщо стрілка повідомлення входить до його фокусу управління, це означає, що повідомлення надійшло до нього від іншого актора. Якщо стрілка перетинає фокус управління, це означає, що повідомлення проходить повз і не адресоване даному актору.

На рисунку 2.2 наведено діаграму послідовності, що відображає процес створення учасником запиту на отримання допомоги.

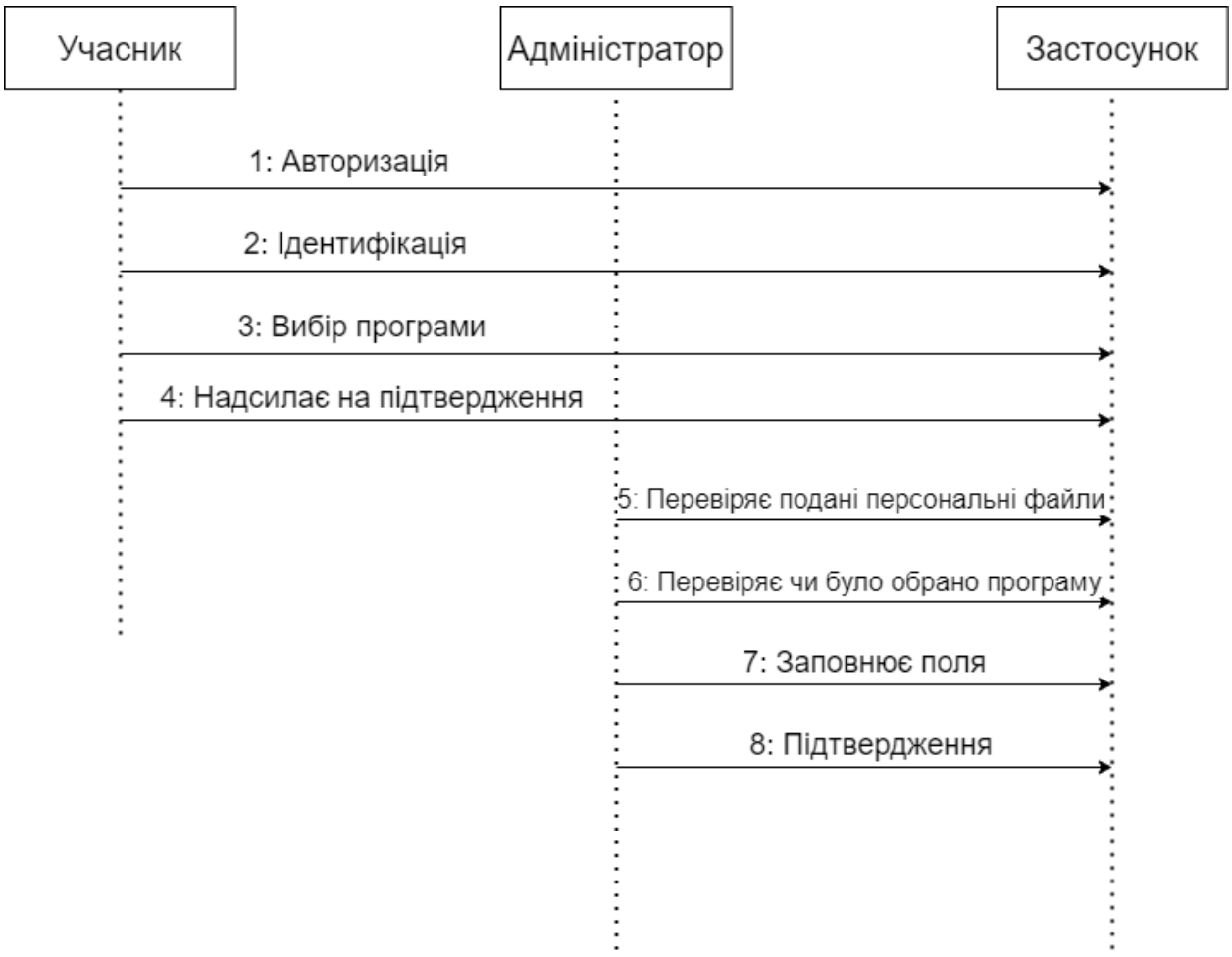

Рис. 2.2. Діаграма послідовності створення запиту на отримання виплат

2.2 Вибір технологій та мов програмування для створення програмного продукту

Для створення вебзастосунку були обрані фреймворки Vue.js та Nuxt.js, і мови програмування JavaScript та TypeScript, також мова розмітки стилів CSS3.

Vue.js [7] (вимовляється як /vjuː/) — це фреймворк JavaScript для створення інтерфейсів користувача. Він створений на основі стандартних HTML, CSS і JavaScript і забезпечує декларативну модель програмування на основі компонентів, яка допомагає вам ефективно розробляти інтерфейси користувача будь-якої складності.

Основні властивості Vue:

- декларативне відображення: Vue розширює стандартний HTML синтаксисом шаблону, який дозволяє декларативно описувати вивід HTML на основі стану JavaScript;
- реактивність: Vue автоматично відстежує зміни стану JavaScript і ефективно оновлює DOM, коли відбуваються зміни.

Vue — це фреймворк та екосистема, яка охоплює більшість загальних функцій, необхідних для розробки інтерфейсу. Але Інтернет надзвичайно різноманітний, і вебзастосунки можуть кардинально відрізнятися за формою та масштабом. Зважаючи на це, Vue розроблено таким чином, щоб бути гнучким і поступово адаптуватися. Залежно від конкретного випадку Vue можна використовувати різними способами:

- покращення статичного HTML без етапу створення;
- вбудовування як вебкомпонентів на будь-якій сторінці;
- односторінкова програма (SPA);
- Fullstack / Server-Side Rendering (SSR);
- Jamstack / генерація статичного сайту (SSG);
- орієнтація на комп’ютер, мобільний пристрій, WebGL і навіть термінал.

Nuxt.js [8] — це прогресивний фреймворк на основі Vue.js для створення сучасних вебдодатків. Він заснований на офіційних бібліотеках Vue.js (vue, vue-router і vuex) і потужних інструментах розробки (webpack, Babel і PostCSS). Мета Nuxt — зробити веб-розробку потужною та продуктивною, маючи на увазі чудовий досвід розробника.

Nuxt — це фреймворк, розроблений, щоб реалізувати потужну архітектуру, яка відповідає офіційним рекомендаціям Vue. Його можна використовувати поступово, щоб створювати все: від статичних цільових сторінок до складних корпоративних вебдодатків. Універсальний за своєю природою, він підтримує різні цілі (серверні, безсерверні або статичні), а рендеринг на стороні сервера можна перемикати.

Розширюваний за допомогою потужної екосистеми модулів, він дозволяє легко підключати кінцеві точки REST або GraphQL, фреймворки CMS, CSS тощо. Підтримка PWA та AMP – це лише один модуль від проекту Nuxt.

NuxtJS є гнучкою основою проекту Vue.js, створюючи структуру для впевненого створення будь-якого проекту.

Особливості Nuxt:

- запис файлів Vue (*.vue);
- автоматичне розділення коду;
- візуалізація на стороні сервера;
- потужна система маршрутизації з асинхронними даними;
- обслуговування статичних файлів;
- ES2015+ транспіляція;
- об’єднання та мінімізація JS і CSS;
- керування елементом <head> (<title>, <meta> тощо);
- гаряча заміна модуля в розробці;
- попередній процесор: Sass, Less, Stylus тощо;
- готові заголовки HTTP/2;

– розширення за допомогою модульної архітектури.

TypeScript [9] — це строго типізована мова програмування, яка базується на JavaScript. TypeScript додає до JavaScript додатковий синтаксис для підтримки більш тісної інтеграції з редактором. Фіксує помилки на ранніх стадіях редактора. Код TypeScript перетворюється на JavaScript, який виконується будь-де, де працює JavaScript: у браузері, на Node.js або Deno та у програмах. TypeScript розуміє JavaScript і використовує визначення типу, щоб надати потужні інструменти без додаткового коду.

Для написання коду вебзастосунка адміністративної частини «Каси взаємодопомоги» буде використовуватися Typescript, тому що він має низку вагомих переваг. Згідно з публікацією [11], Typescript може допомогти зменшити кількість помилок у розробці на 15%. Це також корисно для роботи з іншими розробниками, оскільки кожна властивість компонента, атрибут класу та аргумент функції можуть мати свій тип, визначений у коді, який служить документацією. З цієї та інших причин Typescript стає все більш популярним.

2.3 Архітектура адміністративної частини «Каси взаємодопомоги»

Архітектура адміністративної частини «Каси взаємодопомоги» основана на стандартній архітектурі застосунку Nuxt, яка представлена на рис. 2.3. Ця діаграма заснована на посібнику Vue SSR і розширена з урахуванням Nuxt.js [12]. У лівій частині діаграми можна побачити різні папки в розділі «Код програми» та те, як цей код упакований Nuxt і об’єднаний Webpack.

Також у верхній частині діаграми зображені модулі, сервер проміжного програмного забезпечення (serverMiddleware), плагіни, та сховище даних (Store).

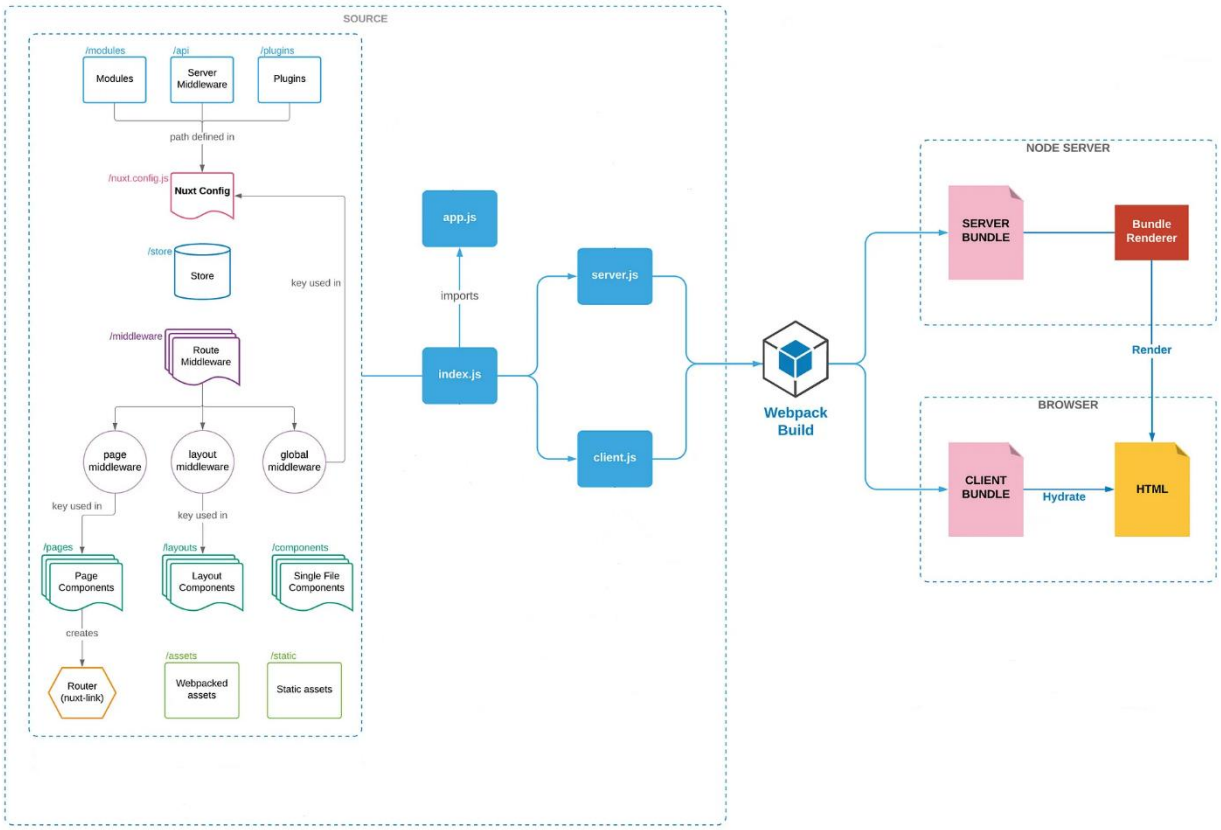

Рис. 2.3. Архітектура універсального застосунку в Nuxt.js [12]

Vuex Store (/store)

Nuxt постачається попередньо запакованим із Vuex, але він не активується, доки не буде створено сховище Vuex у каталозі /store і не створено само сховище.

Це особливий каталог для будь-якого проекту, керованого даними. Тут можна створити сховище даних, а також визначити дію nuxtServerInit.

```
const createStore = () => {
return new Vuex.Store({
...})
}
```

Коли користувач спочатку отримує доступ до програми, це допомагає заповнити/оновити сховище. Він також підтримує стан даних у всій програмі.

Проміжне програмне забезпечення маршруту Route Middleware (/middleware)

У Nuxt доступні три різні види проміжного ПЗ маршруту. Усі вони визначені в одному центральному місці — у каталозі /middleware.

Звідси їх можна використовувати у такі способи:

Глобальне проміжне програмне забезпечення (Global middleware) — запис через конфігурацію Nuxt, впливає на всі маршрути;

```
// nuxt.config.js export default {
  router: {
    middleware: 'authenticated'
  },
}
```

Проміжне програмне забезпечення макета (Layout middleware) - вхід за допомогою макетів, впливає на групу відповідних маршрутів, тобто певні сторінки, доступ до яких мають лише автентифіковані користувачі;

```
// layouts/default.vueexport default {
  middleware: 'authenticated-basic-plan-user'
}
```

Проміжне програмне забезпечення сторінки (Page middleware) - вхід через компонент сторінки, впливає на один маршрут.

```
/
/ pages/index.vueexport default {
  middleware: 'subscribed'
}
```

Наведені вище типи проміжного програмного забезпечення працюють саме в такому порядку, тобто їхні пріоритети не підлягають зміні.

Компоненти /Vue

У проекті Nuxt є три каталоги, де створюються файли .vue.

1. Компоненти сторінки (/pages)

Це найважливіший каталог з усіх, у яких розміщено представлення програм і маршрути. Створені тут компоненти Vue.js безпосередньо перетворюються на маршрути програми.

Корисність компонентів сторінки полягає в динамічних маршрутах. Їх можна використовувати для створення зручних для SEO та орієнтованих на дані URL-адрес. Динамічні маршрути генеруються на основі структури каталогу в /pages.

Крім того, Nuxt додає три спеціальні методи до компонентів сторінки, які більше ніде недоступні. Це validate(), asyncData() і fetch().

```
// pages/index.vue export default {  validate() {
    // validates dynamic URL parameters
    // verifies the availability of the data
  },

  asyncData() {
    // sets component data
  },  fetch() {
    // doesn't set component data, but
    // fetches further contextual data
  }}
```

2. Компоненти макета (/layouts)

Компоненти макета забезпечують структурні аспекти програми. Тут створюються загальні компоненти, які є на всіх сторінках (наприклад, головне меню, додаткове меню, верхній, нижній колонтитул тощо). Вони знаходяться в каталозі /layouts. Також треба додати <nuxt/> в область основного вмісту макета.

```
<template>
<div>
<nuxt/>
</div>
</template>
```

Використовуючи проміжне програмне забезпечення для маршрутизації та зберігаючи стан даних із компонентом макета, можна створювати представлення застосунку для будь-якої кількості типів користувачів із різноманітними сценаріями.

3. Компоненти Vue.js (/components)

Це звичайні, але універсальні компоненти Vue. Вони створюються в каталозі /components. Вони не наповнені спеціальними методами, такими як компоненти Page, але дозволяють структурувати та організувати бізнес-логіку застосунку. Вони також приховують важку розмітку від компонентів сторінки та макета. Це робить кодову базу більш керованою.

Активи

Об’єкти в веб-пакеті (/assets)

Такі активи, як файли JavaScript, спеціальні шрифти та файли CSS, обробляються Webpack за допомогою спеціальних завантажувачів (завантажувач CSS, завантажувач файлів, завантажувач URL-адрес тощо) залежно від типів файлів. Наприклад, якщо CSS написано у форматі .scss або .less, тоді Webpack обробить ці файли за допомогою спеціального завантажувача та виведе скомпільований файл .css, який може використовувати браузер.

Також можна налаштувати nuxt.config.js, щоб наказати Webpack мінімізувати й оптимізувати зображення в папці активів у рамках процесу збирання. Після того як Webpack обробить файли, він додає хеш-код — наприклад, index.4258e3668a44556dd767.js — до оброблених елементів, що допомагає довгостроково кешувати динамічні активи та очищати кеш.

Статичні ресурси (/static)

Активи, які не змінюються, можна безпечно помістити у статичну папку. Webpack ігнорує статичну папку й не оброблятиме нічого в ній.

Модулі, серверне програмне забезпечення та плагіни

Усі вони визначені (за їхнім шляхом) у конфігурації Nuxt. Вони доступні глобально в програмі Nuxt.

Модулі (/modules)

Свіжа програма Nuxt попередньо запакована з Vue, Vue Router, Vuex, Vue Server Rendered і Vue Meta за замовчуванням.

Серверне проміжне програмне забезпеченя serverMiddleware (/api)

serverMiddleware можна вважати точкою розширення програми. Вони особливі, тому що мають доступ до основного екземпляра фреймворку підключення. Оскільки Nuxt використовує підключення для доставки програми, це дозволяє підключити спеціальні функції до базового конвеєра запитів як проміжне програмне забезпечення.

serverMiddlewareвикористовується для:

- створення кінцевої точки API для підключення до зовнішніх програм;
- створення кінцевої точки API для надсилання електронної пошти користувачам із програми Nuxt;
- отримання доступу до запиту та змінення його будь-яким способом, навіть до того, як він досягне Nuxt.

Попередньо заповнених порожніх папок для serverMiddleware і модулів немає, їх потрібно створити, коли це необхідно.

Плагіни (/plugins)

Але в більшості випадків зовнішні пакунки або бібліотеки Vue додаються як плагіни Nuxt. Вони включаються у Nuxt, просто використовуючи синтаксис Vue.use(). Деякі з основних плагінів, які є дуже популярними, це Vue Bootstrap, перевірка форм, набори іконок і axios. Також можна писати спеціальні плагіни та додавати їх у кореневій папці програми. Вони будуть доступні глобально у програмі Nuxt.

Пакування, збірка і розгортанння

Програма створюється за допомогою npm run build. Nuxt пакує програмний код застосунку.

Як показано на рисунку 2.4, index.js є основною точкою входу, яка імпортує app.js.

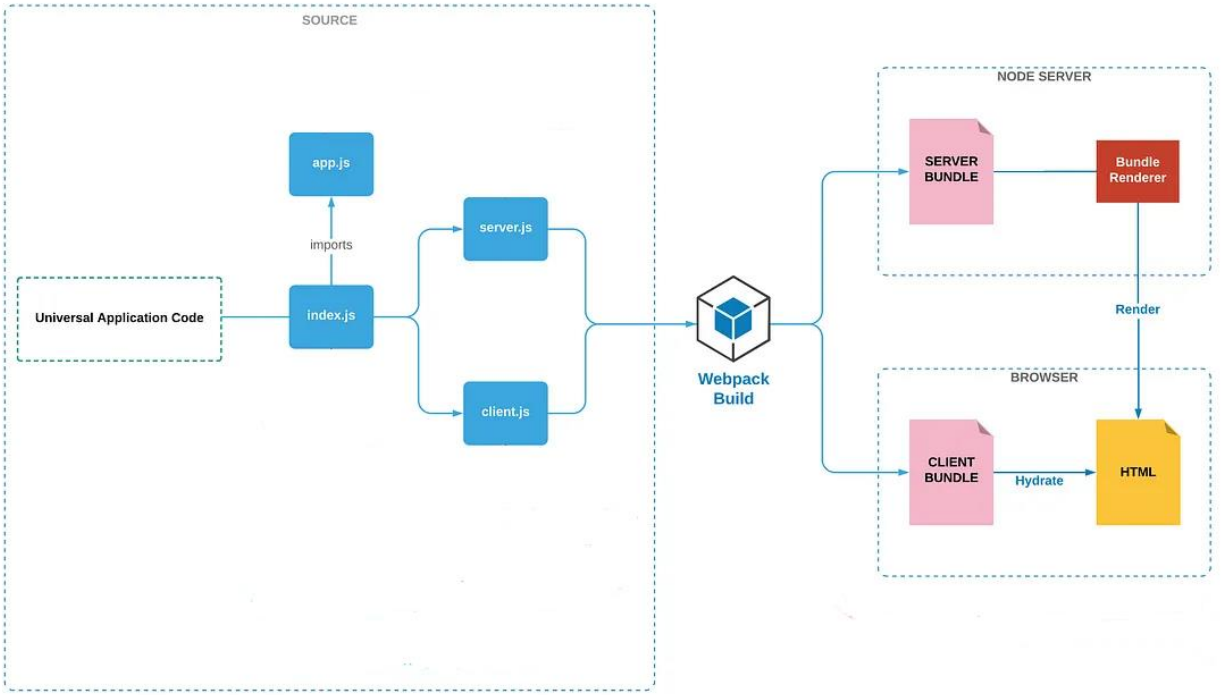

Рис. 2.4. Процеси пакування та збірки застосунку [12]

App.js визначає кореневий екземпляр Vue, оскільки .nuxt/App.js - це компонент Vue.

Після визначення цього кореневого екземпляра Vue запис клієнта (client.js) створює на його основі новий екземпляр і монтує його до елемента DOM. Кінцеві користувачі бачать новий екземпляр програми у своїх браузерах. Хоча запис сервера (server.js) створює новий екземпляр програми для кожного запиту.

Далі Webpack об’єднує програму так, що код виконується як на стороні клієнта, так і на стороні сервера. Серверний пакет рендерить серверну частину, а клієнтський пакет містить статичну розмітку HTML у браузері. Він перетворює його на динамічний DOM, який може реагувати на зміни даних на стороні клієнта.

Nuxt робить усе це «із коробки», тож розробнику не треба писати це налаштування вручну. Останні два етапи — пакування та комплектування — інкапсулюються Nuxt, що дозволяє зосередитися на коді програми, який в кінцевому підсумку доставляє остаточний застосунок.

Nuxt також має власний життєвий цикл. Діаграма на рис. 2.5 показує, що відбувається, коли запускається програма Nuxt і перед тим, як запускається програма vue.

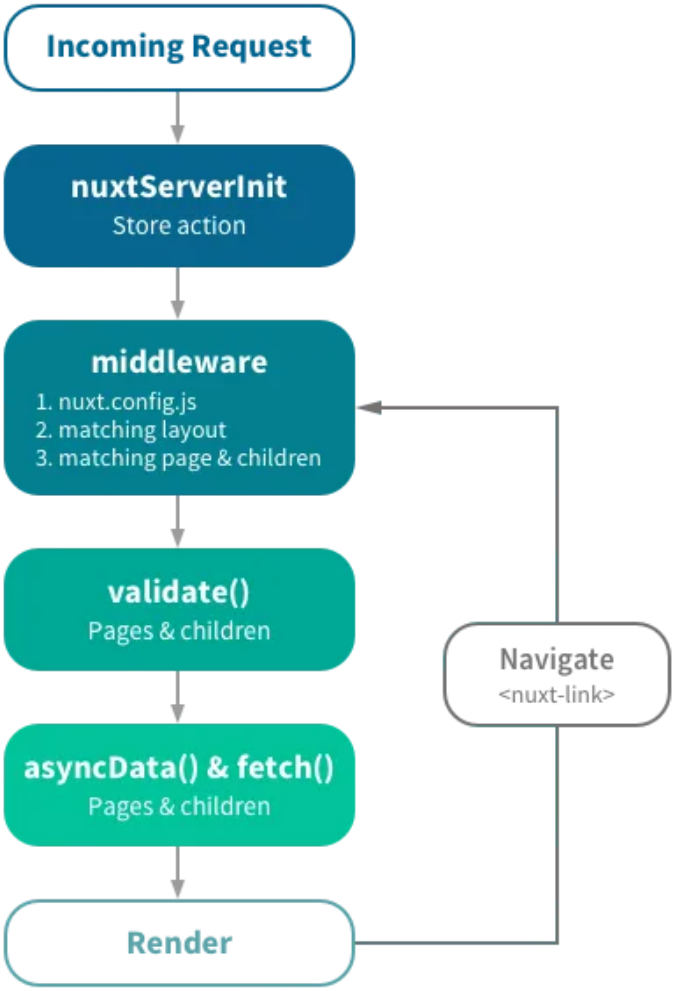

Рис. 2.5. Життєвий цикл Nuxt [8]

Перевірка (Validate)

Усі дані, надані користувачем, мають бути перевірені. Nuxt пропонує для цього метод перевірки. Метод перевірки має бути встановлено в експортованому об’єкті файлів dot vue. У життєвому циклі Nuxt перевірка буде викликана після проміжного програмного забезпечення та перед asyncData.

Метод перевірки призначений для перевірки значень, а не для скидання значень. Наприкінці він повинен повернути або true, або false. Якщо він

повертає true, файл ідентифікатора підкреслення буде завантажено нормально. Якщо він повертає false, Nuxt безпосередньо відобразить попередньо визначену сторінку помилки. Але URL-адреса в адресному полі не зміниться.

Проміжне ПЗ (Middleware):

По суті, проміжне програмне забезпечення — це заздалегідь визначена функція. Її можена застосувати до певної сторінки або до всіх сторінок. У життєвому циклі Nuxt проміжне програмне забезпечення викликається перед методом перевірки. Оскільки проміжне програмне забезпечення викликається на дуже ранній стадії, його можна використовувати для багатьох речей, як-от перевірка статусу входу, перевірка даних, наданих користувачем, тощо.

Одне проміжне програмне забезпечення має бути визначене в одному файлі JS. Каталог проміжного програмного забезпечення призначений для зберігання файлів проміжного програмного забезпечення.

Щоб створити файл Middleware, спочатку треба створити файл JS у каталозі проміжного програмного забезпечення. У файл JS експортується за замовчуванням анонімна функція, яка буде функцією проміжного програмного забезпечення. Nuxt призначить контекст анонімній функції як значення аргументу. Контекст надає функції проміжного програмного забезпечення майже повний доступ до всієї програми.

Context.params і context.query можуть отримати значення, передані в URL: Context.store отримує VueX, а Context.error() може допомогти запустити сторінку помилки та надіслати значення на сторінку помилки.

Проміжне програмне забезпечення може бути прив’язане до певної сторінки або до всіх сторінок. Якщо необхідно прив’язати Middleware до певної сторінки, треба додати властивість проміжного програмного забезпечення до цієї сторінки. Значенням властивості проміжного ПЗ є масив, кожен елемент якого має одне ім’я Middleware.

AsyncData

Метод asyncData — це те, що використовується для зв’язку з серверною програмою та базою даних. У схемі Nuxt asyncData буде викликано після validate(), але перед рендерингом Vue. Як і validate(), asyncData також отримає контекст як значення аргументу.

Іншою важливою особливістю asyncData є те, що він буде викликатися автоматично. Це робить його ідеальним кандидатом для отримання початкових даних із серверної програми та бази даних.

AsyncData не містить this, тому що він викликається після методу перевірки, але перед рендерингом Vue. Це означає, що до моменту виклику методу asyncData компонент vue ще не створено. У результаті this в asyncData не отримає екземпляр компонента.

Найважливішим у методі asyncData є те, що результат, який він повертає, автоматично об’єднується з даними vue. Це вирішило проблему відсутності this, який би вказував на екземпляр компонента. Властивості, повернуті методом asyncData, можна безпосередньо відобразити в шаблоні.

Fetch ()

У Vue початкові дані отриму.nmcz за допомогою створеного хука. У Nuxt є два варіанти: метод fetch і метод nuxtServerInit. Обидва ці два методи будуть викликані автоматично, але вони будуть викликані в інший час.

Метод fetch встановлено у файлах сторінки. Так само, як проміжне програмне забезпечення та asyncData, метод fetch також отримує контекст як значення аргументу.

Context.params і context.query надають методу fetch доступ до даних, переданих в URL. Вони дозволяють виконувати вибірку на основі даних, переданих URL-адресою. Наприклад, можна передати ідентифікатор в URL-адресу, метод fetch може надіслати цей ідентифікатор серверній програмі та відповідним чином запитати базу даних.

Context.error() вмикає метод fetch для запуску сторінки помилки, якщо щось піде не так.

Після того, як метод fetch отримав дані з серверної програми та бази даних, Context.store.commit() дозволяє методу fetch запускати метод мутації VueX, який може встановлювати властивості стану VueX.

nuxtServerInit ()

fetch() — не єдиний спосіб отримати вихідні дані, також можна використовувати nuxtServerInit. NuxtServerInit — це метод дії VueX. На відміну від звичайного методу дії, nuxtServerInit буде викликано автоматично. У життєвому циклі Nuxt метод nuxtServerInit буде викликано на самому початку.

Звичайний метод дії викликається методом диспетчеризації та має два аргументи. Перше — контекст VueX, а друге — значення, передане методом відправки. Подібно до fetch і asyncData, nuxtServerInit також отримає контекст. Отже, nuxtServerInit має два контексти. Перший – це контекст VueX, а другий – контекст Nuxt.

Можемо отримати доступ до методу мутації VueX через перший аргумент: vuex_context.commit(). Якщо щось піде не так, необхідно викликати сторінку помилки за допомогою методу error. Доступ до методу помилки можна отримати через другий аргумент: nuxt_context.

За замовчуванням компоненти Vue створюють і обробляють DOM у браузері як вихідні дані. Однак також можна відобразити ті самі компоненти в рядки HTML на сервері, надіслати їх безпосередньо в браузер і, нарешті, «гідратувати» статичну розмітку в повністю інтерактивну програму на клієнті.

Відображену на сервері програму Vue.js також можна вважати «ізоморфною» або «універсальною» в тому сенсі, що більшість коду вашої програми виконується як на сервері, так і на клієнті.

Порівняно з клієнтською односторінковою програмою (SPA), перевага візуалізації на стороні сервера (SSR) полягає в наступному:

- швидший час доступу до вмісту: це більш помітно на повільному Інтернеті чи повільних пристроях. Для відображення відображеної на сервері

розмітки не потрібно чекати, поки весь JavaScript буде завантажено та виконано, тому користувач швидше побачить повністю оброблену сторінку. Крім того, вибірка даних виконується на стороні сервера для першого відвідування, яке, швидше за все, має швидше з'єднання з базою даних, ніж клієнт. Загалом це призводить до покращення показників Core Web Vitals, кращої взаємодії з користувачем і може бути критичним для додатків, де час до вмісту безпосередньо пов'язаний із коефіцієнтом конверсії;

- уніфікована ментальна модель: можна використовувати ту саму мову та ту саму декларативну, компонентно-орієнтовану ментальну модель для розробки всієї програми, замість того, щоб перемикатися між бекенд-системою шаблонів і інтерфейсом;
- краще SEO: сканери пошукової системи безпосередньо бачитимуть повністю відтворену сторінку.

2.4 Висновки до розділу 2

У другому розділі кваліфікаційної роботи було проведено проєктування адміністративної частини «Каси взаємодопомоги». Спочатку була розглянута структура та функціонал застосунку. Для моделювання поведінки програми було використано різні типи UML діаграм. Також були розглянуті різні варіанти використання програми, і була розроблена діаграма варіантів використання. Було проведено опис акторів та опис варіантів використання програми.

Було розглянуто різні фреймворки, і було обрано Vue.js та Nuxt.js для створення програмного коду застосунку. Також був виконаний огляд мов сценаріїв та програмування, і прийнято рішення використовувати JavaScript (JS) та TypeScript під час розробки проекту. Для реалізації візуальних складових застосунку було обрано мову розмітки стилів CSS3.

У якості основи для архутектури застосунку була обрана архітектура універсального застосунку в Nuxt.js. Також було вирішено використати

технологію візуалізації на стороні сервера (SSR), яка покликана надати застосунку переваги у швидкості відображення вмісту, уніфікованому процесі написання коду, та SEO оптимізації.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО КОМПОНЕНТУ АДМІНІСТРАТИВНОЇ ЧАСТИНИ «КАСИ ВЗАЄМОДОПОМОГИ»

3.1 Структура програмного продукту

Виходячи з матеріалів попередніх розділів та спираючись на результати вивчення документації для Nuxt3, було розроблено вебзастосунок, що реалізує функції адміністративної частини інформаційної системи благодійного фонду взаємодопомоги. Структура взаємодії елементів застосунку наведена на рисунку 3.1.

Після завантаження вебзастосунку з вебсервера благодійного фонду на клієнтський пристрій, подальший обмін інформацією між застосунком та сервером відбувається з використанням компонента маршрутизації HTTP запитів, в який завантажується файл проміжного шару auth.ts. Програмний код проміжного шару виконується до того, як відбувається побудова дерева DOM кожної конкретної сторінки. За допомогою програмного коду, розміщеного у проміжному шарі, можна виконувати додаткову маршрутизацію запитів залежно від того, чи авторизований користувач в поточний момент часу, чи ні.

Відповідно до рекомендацій з побудови Nuxt-застосунків авторизовану та анонімну частини сайту розділено на найвищому рівні процесу побудови дерева DOM. Це означає, що фактично елементи дерева head та body для авторизованої та анонімної частин сайту є різними. Це відбувається шляхом створення окремих файлів шарів нижнього рівня “default layout” та “anonymous layout”. В свою чергу, кожен з файлів шарів нижнього рівня інтегрує в себе компоненти заголовків та меню (за необхідності), і надалі це виступає підґрунтям для відображення сторінок сайту. При цьому в авторизованій частині компонент меню враховує, чи є поточний користувач суперадміністратором, чи ні.

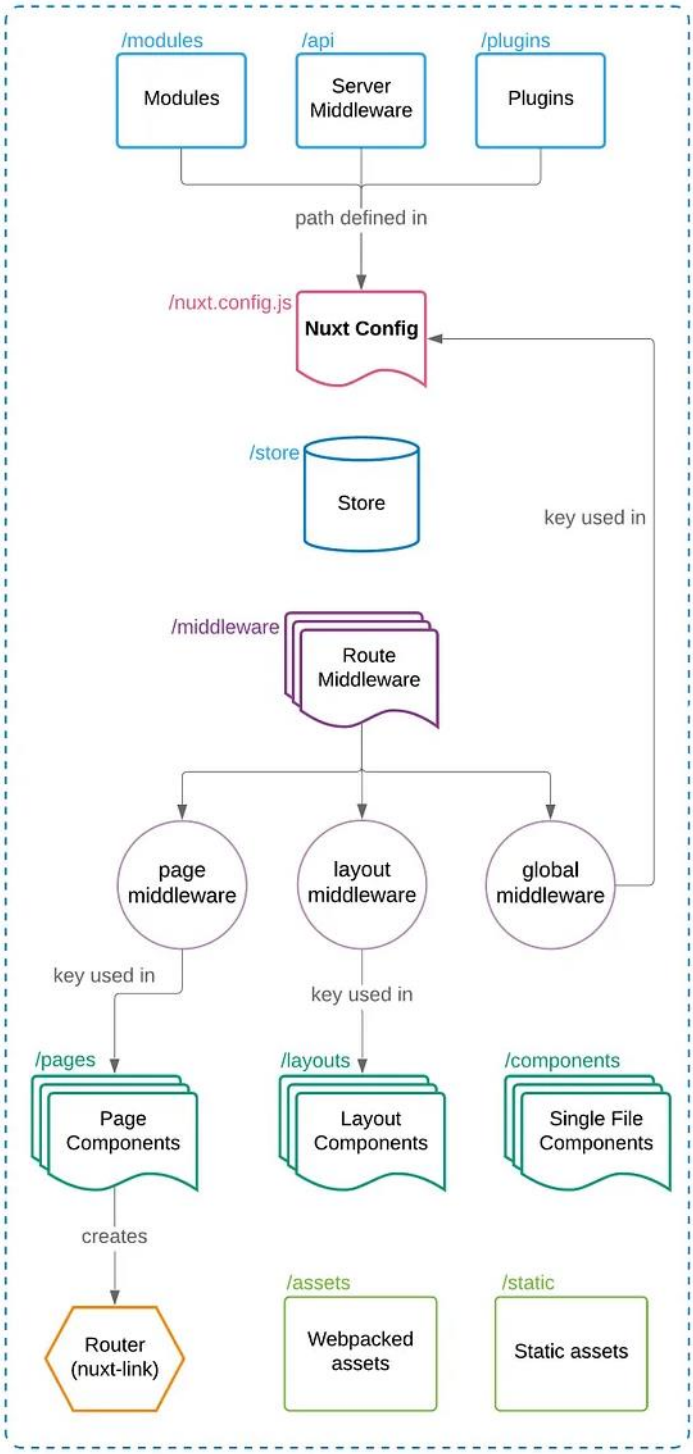

Рис. 3.1. Схема взаємодії елементів застосунку

Для збереження та передавання інформації між сторінками сторінки авторизованого сайту використовують сховище data, в якому за потреби створюються та знищуються об’єкти, що містять службову інформацію, яка впливає на роботу інших сторінок сайту. Для відображення інформації про поточного користувача та для використання електронного квитка

access_token для авторизованого доступу до серверної частини сайту благодійного фонду, всі сторінки авторизованої частини мають доступ до стану сховища auth. На кожній сторінці авторизованої частини розміщено посилання для завершення сеансу роботи поточного користувача. При натисканні на це посилання сховище auth очищується, знищуючи відповідну інформацію в Cookies браузера.

Неавторизована частина сайту складається з однієї сторінки входу. За умови успішного входу інформація про поточного користувача та електронний квиток доступу access_token для роботи з серверною частиною сайту благодійного фонду записується у сховище store та відповідні Cookies браузера.

3.2 HTTP маршрутизація в Nuxt-застосунку

HTTP маршрутизація у розробленому застосунку здебільшого має статичний характер. У випадку, коли потрібно передавати на сторінку відомості про об’єкт, з яким ця сторінка має працювати, використовується спосіб обміну інформації через локальне сховище Nuxt. Таким чином вдалося уникнути динамічної побудови URL. Маршрути Nuxt-застосунку сайту адміністрування роботи благодійного фонду наведені у таблиці 3.1.

Таблиця 3.1. – Маршрути адміністративної частини вебзастосунку «Каса взаємодопомоги»

HTTP маршрут	Призначення
/	Домашня сторінка Nuxt-застосунку
/login	Сторінка авторизації
/logout	Сторінка виходу
/user_list	Сторінка списку користувачів (адміністраторів системи)
/add_user	Сторінка додавання нового користувача (адміністратора)
/update_user	Сторінка редагування облікового запису

HTTP маршрут	**Призначення**
	користувача (адміністратора)
/program_list	Сторінка списку програм
/add_program	Додавання програми
/update_program	Редагування програми
/member_list	Сторінка списку учасників
/update_member	Сторінка редагування облікового запису учасника
/request_list	Сторінка списку запитів всіх учасників
/update_request	Сторінка редагування запису про запит учасника
/cash_flow	Сторінка книги руху коштів

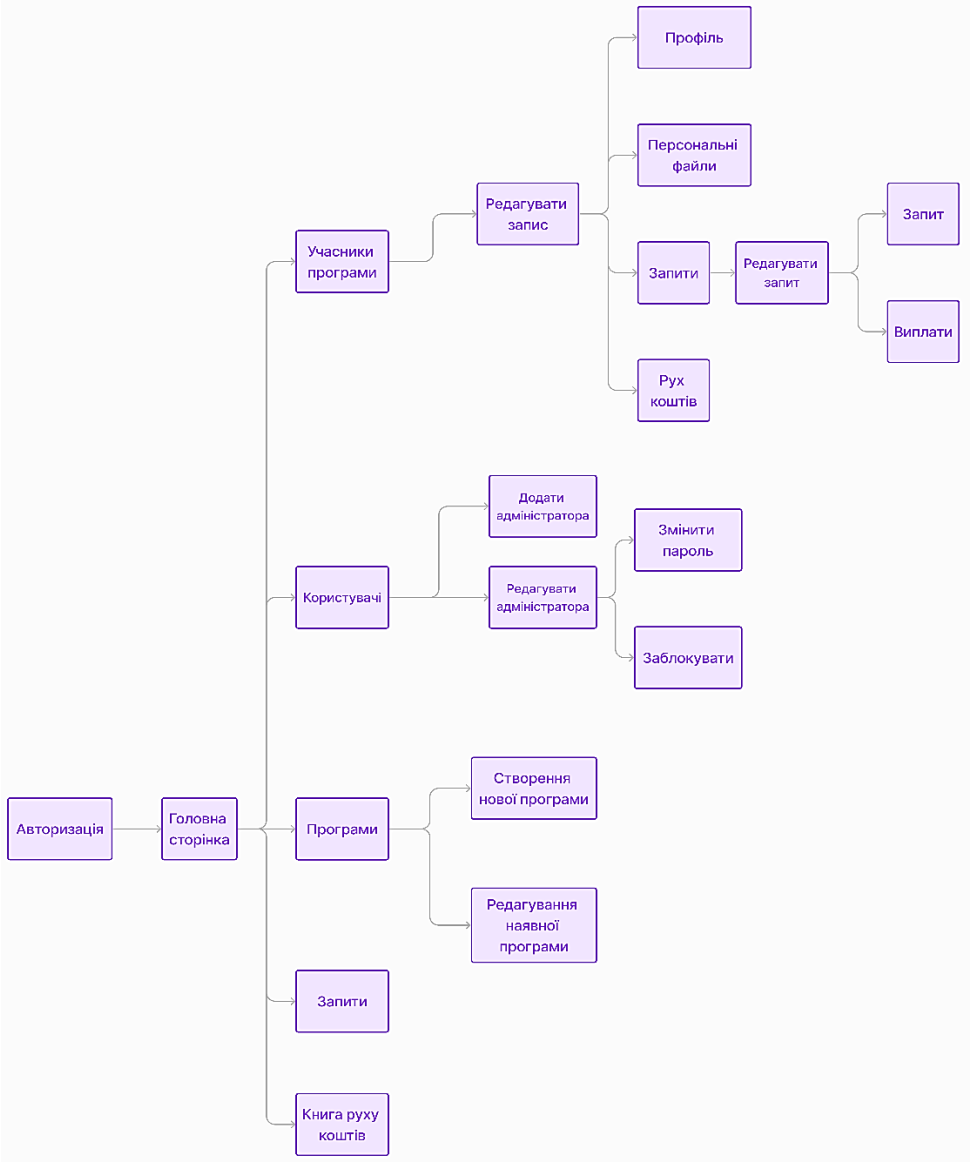

Рис. 3.2. Мапа переходів по адміністративному сайту

3.3 Опис особливостей розробленого програмного продукту

Для доступу до адміністративної частини сайту благодійного фонду необхідно пройти авторизацію; анонімний вхід не передбачений. Тому користувачеві пропонується авторизуватися при відвідуванні цієї секції.

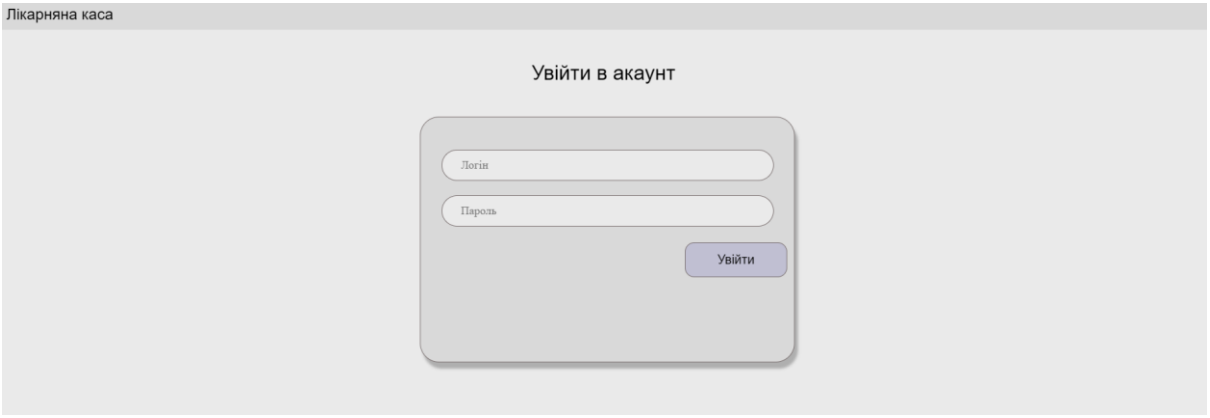

Рис. 3.3. Сторінка авторизації

На рівні застосунку авторизація вимагає використання сховища для ключа доступу, а також для інформації про користувача, яка має відображатися в заголовку. У Nuxt існує кілька модулів, які організовують власне сховище Store. Однак у цих модулів є істотний недолік: під час примусового перезавантаження сторінки їхній вміст очищається, що призводить до втрати даних про сесію користувача та його електронний квиток access_token для доступу до сервера сайту.

Для розв'язання цієї проблеми в розробленому застосунку було вирішено використовувати Cookies як основу для сховища. Впровадження цієї технології в додаток Nuxt було здійснено за допомогою модуля сховища store/auth.ts.

Лістинг 3.1 Вміст модуля store/auth.ts:

```
import { defineStore } from 'pinia';
import nuxtStorage from 'nuxt-storage';

const userCookie = useCookie("user");
```

```
const accessTokenCookie = useCookie("access_token");

const getUser = () => {
    try {
        return userCookie.value
    } catch(error) {
        return null;
    }
};

const getAccessToken = () => {
    try {
        return accessTokenCookie.value
    } catch(error) {
        return null;
    }
};

export const useAuthStore = defineStore({
  id: 'auth',
  state: () => ({
    user: getUser(),
    access_token: getAccessToken(),
  }),
  actions: {
    async login(loginForm) {
      const config = useRuntimeConfig()
      const baseUrl = config.public.baseURL;
      await $fetch(${baseUrl}signin, {
        method: "POST",
        body: loginForm
      })
        .then(response => {
          this.user = response.user
          userCookie.value = response.user
          this.access_token = response.access_token
          accessTokenCookie.value = response.access_token
        })
        .catch(error => { throw error })
    },
    logout() {
      this.user = null
```

```
      this.access_token = null
      userCookie.value = null
      accessTokenCookie.value = null
    },
    isAuthenticated() {
        return (this.user == null) ? false : true;
    }
  }
});
```

У разі успішної авторизації користувача буде перенаправлено на головну сторінку, з якої він зможе виконувати всі основні операції. До основних операцій належать:

- можливість роботи зі списком учасників благодійних програм;
- можливість обробки запитів від учасників;
- додавання інформації про благодійні програми;
- обробка платіжної інформації.

Якщо користувач є супер адміністратором, йому будуть доступні додаткові опції для роботи зі списком інших адміністраторів. Зокрема:

- додавання нових адміністраторів;
- редагування інформації про адміністраторів;
- зміна пароля;
- блокування облікових записів.

Всі зазначені вище опції будуть доступні тільки в тому випадку, якщо в поточний момент відвідувач сайту має електронний квиток access_token, термін дії якого ще не збіг. За перевірку цього відповідає модуль “середнього прошарку” middleware/auth.ts

Лістинг 3.2 Вміст модуля middleware/auth.ts:

```
import { useAuthStore } from "~/store/auth";

export default defineNuxtRouteMiddleware((to, from) => {
  const authStore = useAuthStore();
  if (!authStore.isAuthenticated()) {
    return navigateTo('/login');
```

```
    }
})
```

Для роботи з програмами благодійного фонду треба перейти до сторінки списку активних програм використовуючи головне меню.

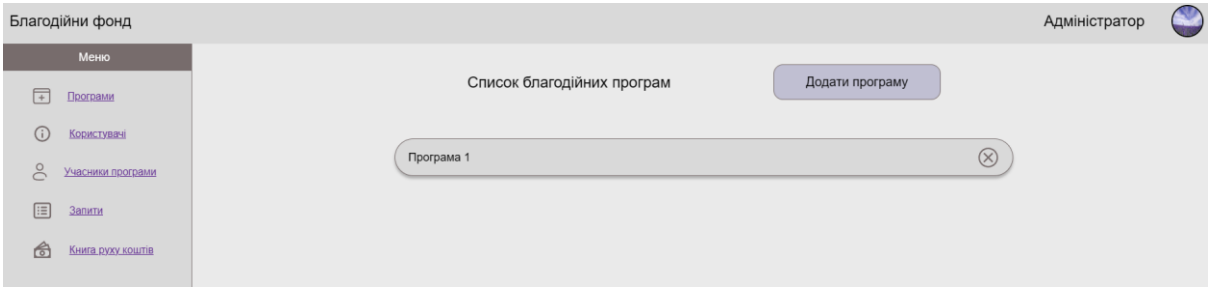

Рис. 3.4. Сторінка зі списком благодійних програм

На сторінці додавання програм можна ознайомитися зі списком активних програм або, за необхідності, додати нову, натиснувши на кнопку «додати програму», або редагувати вже наявну, натиснувши на неї.

Читання списку програм відбувається щоразу, як адміністратор відвідує відповідну сторінку. Кешування цього списку, а також решти списків у застосунку не відбувається, оскільки застосунок передбачає багатокористувацьку роботу, тому враховуючи невеликий обсяг даних для читання, немає сенсу їх кешувати. Читання списку програм відбувається наступним чином:

Лістинг 3.3 Typescript код читання списку програм:

```
await $fetch(`${baseUrl}programs`, {
        method: "GET",
        headers: {
            "Authorization": "Bearer "+access_token.value,
            "Content-Type": "application/json"
        }
    })
    .then(response => {
        programs.value = response.data;
    })
    .catch(error => {
        console.log('Помилка при отриманні даних:', error.message);
```

```
});
```

Операції додавання нової програми та редагування наявної мають схожий інтерфейс. На рис. 3.5 зображено сторінку з процесом редагування програми.

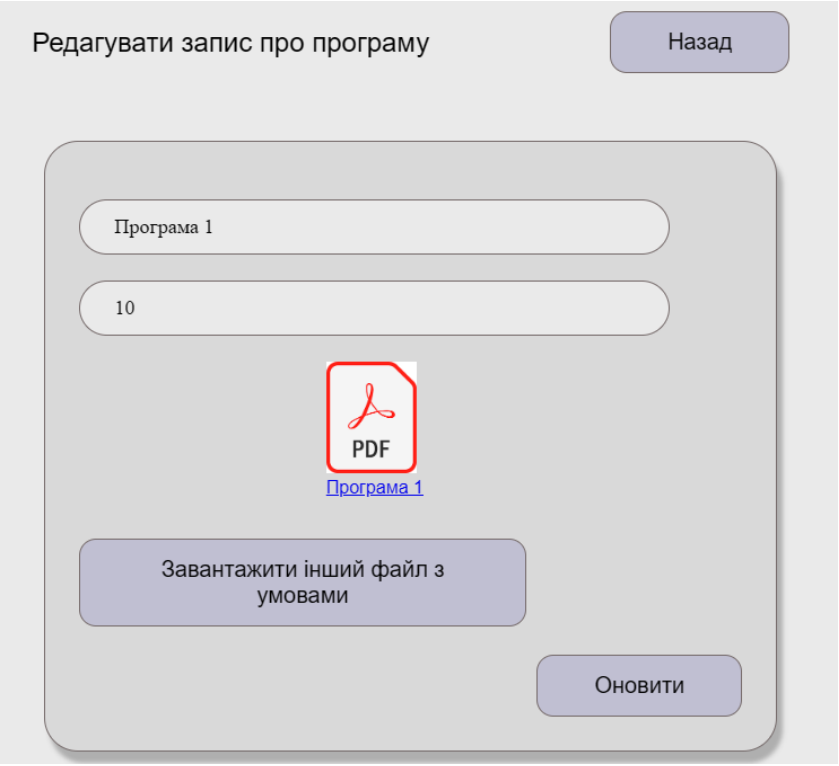

Рис 3.5. Сторінка редагування наявної благодійної програми

Наведена сторінка містить форму для редагування, яка відрізняється від форми додавання тим, що показується поточний файл програми, який можна замінити, завантаживши новий.

Завантаження файлів в застосунках Nuxt може виконуватися у різний спосіб, однак, найпопулярнішими є або встановлення додаткового модуля Nuxt, або написання власного запиту на завантаження з використанням заголовків multipart, або Content-Disposition. В цій роботі був використаний HTTP заголовок Content-Disposition оскільки серверна частина не дозволяє використовувати multipart

Лістинг 3.4 Typescript код виклику для завантаження файлів:

```
        await $fetch(`${baseUrl}program`, {
            method: "PUT",
            body: formData,
            headers: {
                "Authorization": "Bearer "+access_token.value,
                "Content-Disposition": `form-data; name="new_program_file";
filename=${newProgram.new_program_file.name}`
            }
        })
        .then (response => {
            navigateTo('/program_list');
        })
        .catch(err => { console.log(err); })
```

Найчастіше використовуваною функцією для адміністраторів благодійного фонду є робота зі списком учасників благодійних програм.

Фактично, адміністратор благодійного фонду має контактувати з учасниками програм, коли вони реєструються, подають запит, вносять зміни у свої облікові дані, та коли спливає термін участі у благодійній програмі. У всіх цих випадках, такі учасники потраплятимуть в окремий список та відображатимуться на вкладинці “Необроблені”, як показано на рисунку 3.6.

Рис 3.6. Сторінка зі списком учасників благодійної програми

Для того, аби передавати між різними формами інформації про об’єкт зі списку, щодо якого треба відкрити сторінку з деталізацією інформації, використовується модуль сховища Nuxt data.ts. В цьому модулі реалізовані

методи щодо тимчасового збереження всіх основних об’єктів, які можуть бути редаговані, або потрібні під час редагування.

Лістинг 3.5 Вміст модуля store/data.ts:

```
import { defineStore } from 'pinia';
import nuxtStorage from 'nuxt-storage';

const baseUrl = 'https://test4.iti.dp.ua/'

export const useDataStore = defineStore({
  id: 'data',
  state: () => ({
    pass_data: null,
  }),
  actions: {
    setUser(user) {
        if (this.pass_data == null) this.pass_data = {};
        this.pass_data.user = user;
    },
    setProgram(program) {
        if (this.pass_data == null) this.pass_data = {};
        this.pass_data.program = program;
    },
    setMember(member) {
        if (this.pass_data == null) this.pass_data = {};
        this.pass_data.member = member;
    },
    setMemberId(member) {
        if (this.pass_data == null) this.pass_data = {};
        this.pass_data.member_id = member.id;
    },
    setRequestId(request) {
        if (this.pass_data == null) this.pass_data = {};
        this.pass_data.request_id = request.id;
    },
    setProgramId(program) {
        if (this.pass_data == null) this.pass_data = {};
        this.pass_data.program_id = program.id;
    },
    setProgram(program) {
        if (this.pass_data == null) this.pass_data = {};
        this.pass_data.program = program;
```

```
        }
      }
    });
```

В інакшому випадку, якщо треба віднайти когось з учасників, необхідно перейти на вкладинку пошуку, як показано на рисунку 3.7

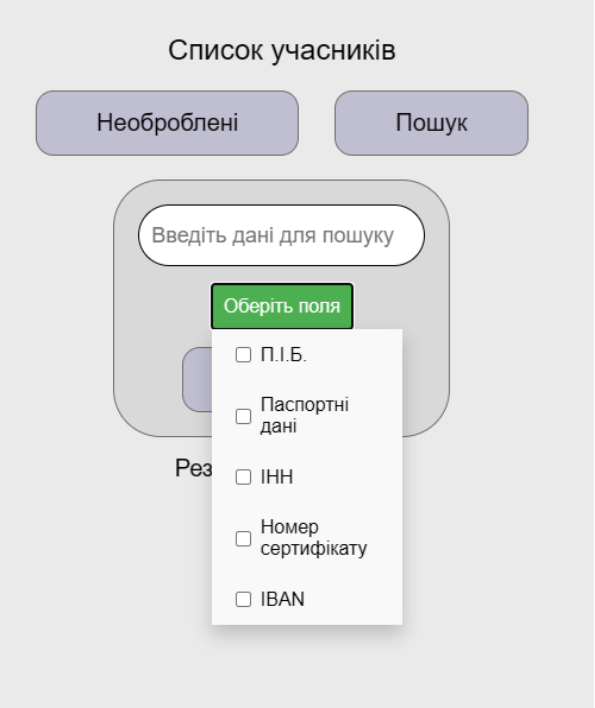

Рис 3.7. Вкладинка пошуку

Пошуковий рядок може містити шаблон пошуку, та за допомогою прапорців користувач можна позначити, де має відбуватися пошук:

- ✓ ПІБ;
- ✓ паспортні дані;
- ✓ ІНН;
- ✓ номер сертифіката;
- ✓ IBAN.

Якщо в результаті пошуку було знайдено записи про учасників програм, що відповідають шаблону, то вони з’являться під формою пошуку і переглянути інформацію про конкретний запис можна натиснувши на відповідний результат, як показано на рисунку 3.8.

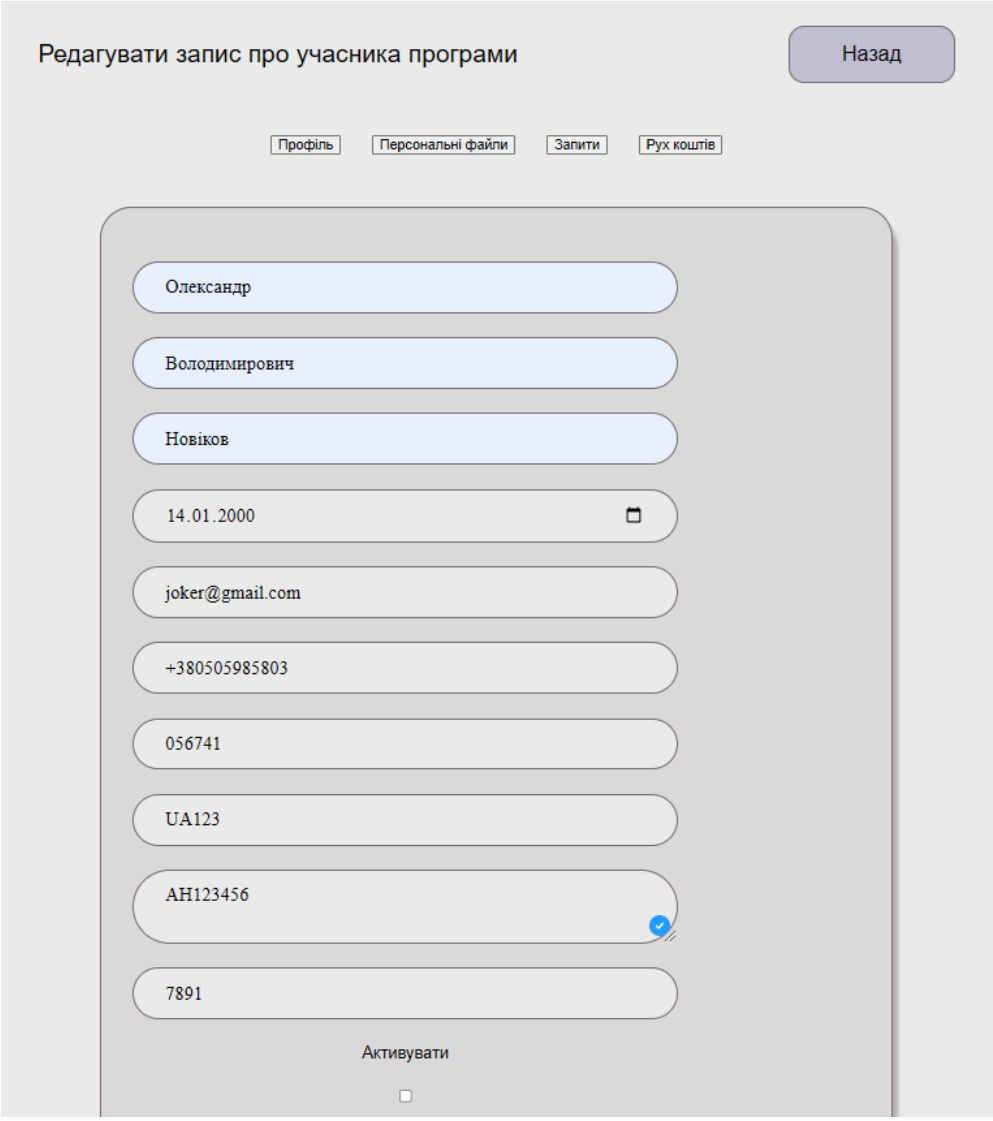

Рис 3.8. Вікно редагування інформації про учасника програми

Під час натискання на рядок з результатом відкривається вікно редагування інформації про учасника програм із кількома панелями:

Профіль, за замовчуванням, містить такі поля:

- ✓ ім'я;
- ✓ прізвище;
- ✓ по батькові;
- ✓ дата народження;
- ✓ електронна адреса;
- ✓ номер телефону;
- ✓ IBAN;

- ✓ ІНН;
- ✓ паспортні дані;
- ✓ номер сертифікату.

Також можна активувати або деактивувати користувача, встановивши прапорець «Активувати». Під полями профілю міститься список програм, у яких бере участь учасник, і його актуальні персональні файли.

В багатьох місцях на сайті треба виконувати перегляд завантажених на сервер сайту файлів. Однак, через політику безпеки, багато з цих файлів не мають прямого посилання й відповідно неможливо виконати їхній перегляд за допомогою метода GET. Тому, виникає проблема з тим, як переглянути файл завантажений методом POST. Для цього динамічно створюється окреме посилання для кожного файлу, як наведено нижче у лістингу 3.6.

Лістинг 3.6 Typescript код для динамічно створеного посилання:

```
.then (response => {
    let a = document.createElement("a");
    let fileURL = window.URL.createObjectURL(response);
    a.href = fileURL;
    a.download = file.file.filename,
    a.target = '_blank';
    a.click();
})
```

Персональні файли - тут можна переглянути всі файли користувача окремо від інших даних.

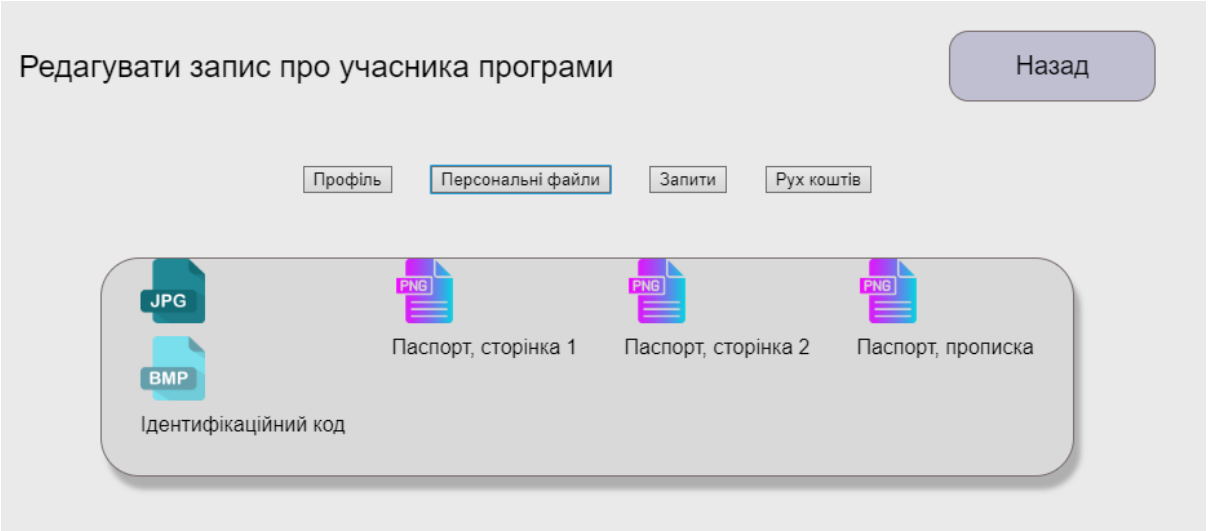

Рис 3.9. Панель персональних файлів

Запити - панель, де відображаються запити користувача з інформацією про них:

- дата та час зміни;
- дата та час створення;
- сума;
- статус.

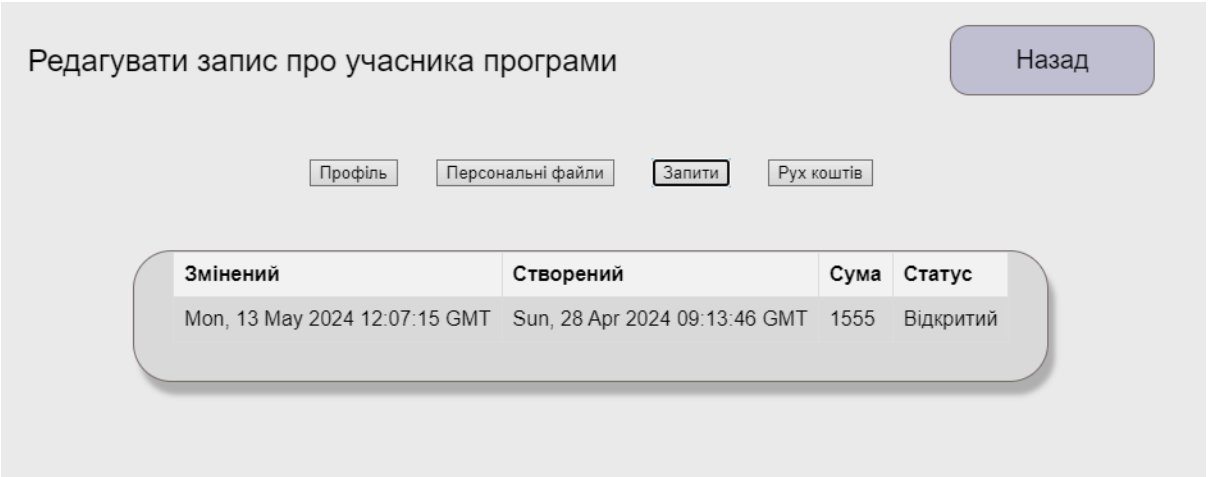

Рис 3.10. Панель запитів

При натисканні на рядок із запитом відкривається редагування інформації про нього, що містить:

- текст запиту;
- суму;
- діагноз;
- прикріплені до запиту файли.

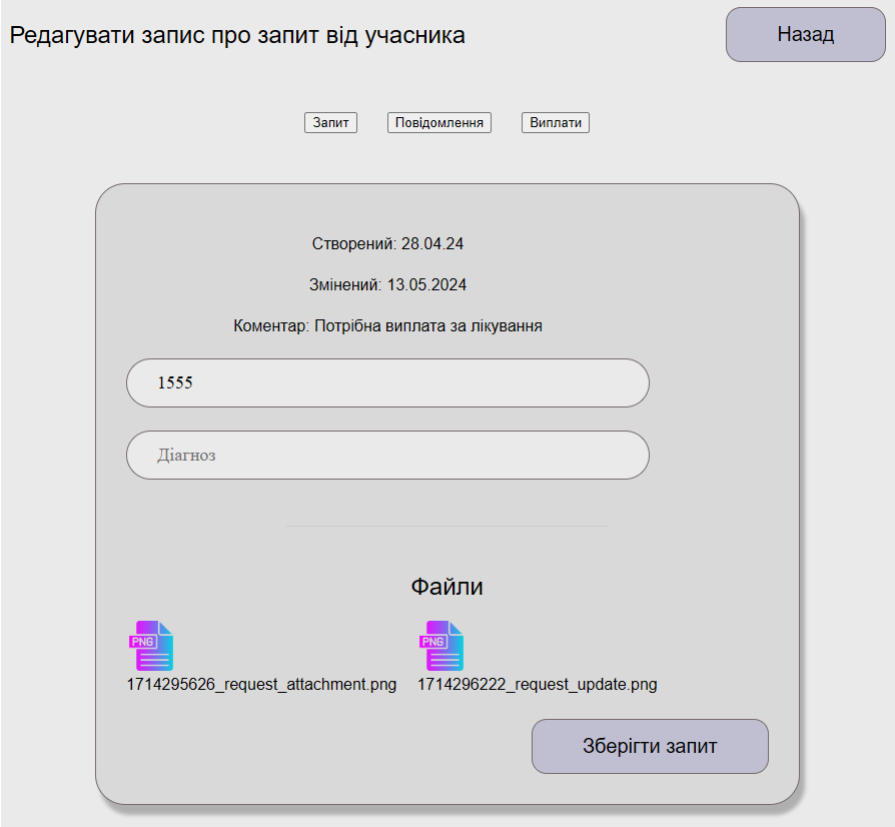

Рис 3.11. Панель редагування запиту

У цій же панелі є підрозділ «Виплати», де можна ознайомитися зі списком виплат за цим запитом:

- дата виплати;
- тип виплати;
- сума;
- запис про нову виплату.

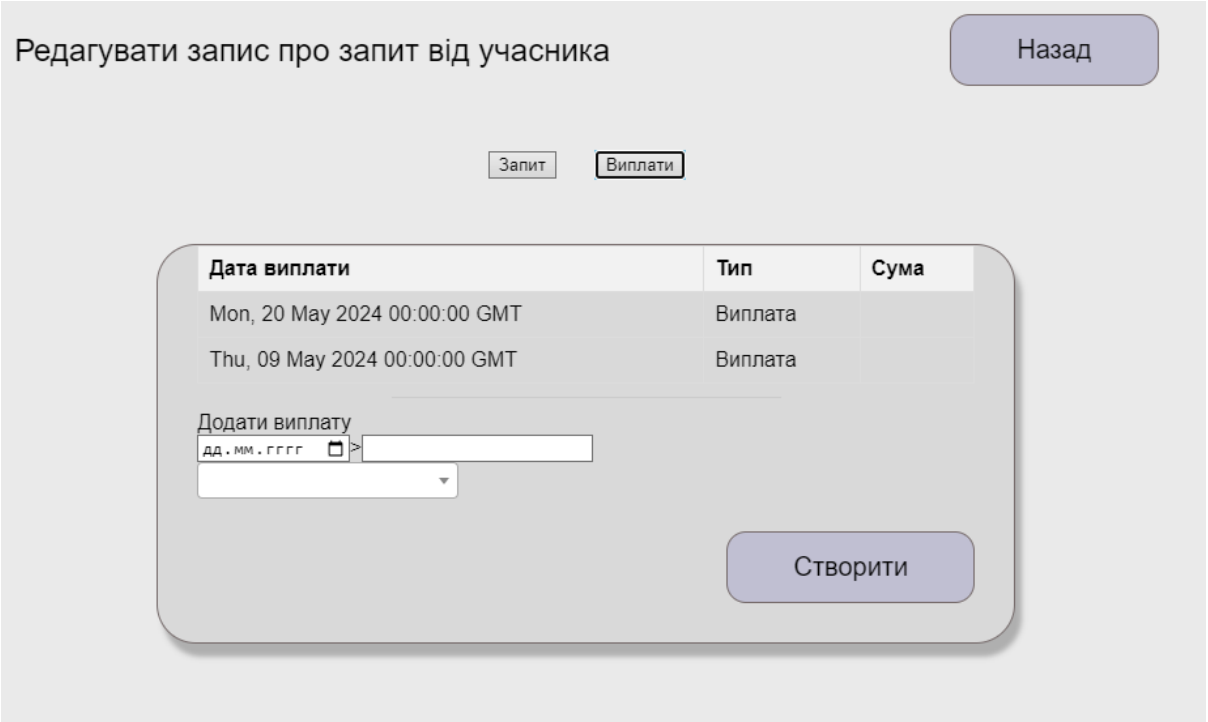

Рис 3.12. Підрозділ “Виплати”

Додати інформацію про нову виплату можна за допомогою коду наведеного у лістингу 3.7

Лістинг 3.7 Typescript код для додавання нової виплати:

```
await $fetch(`${baseUrl}payment`, {
    method: "PUT",
    body: charge.value,
    headers: {
        "Authorization": "Bearer "+access_token.value,
        "Content-Type": "application/json"
    }
})
.then (response => {
    pass_data.value.charge = charge.value;
    navigateTo('/update_request');
})
.catch(err => { console.log(err); })
```

Рух коштів – це панель, що відображає рух коштів користувача, і має наступні поля:

- дата;

- виплата;
- сума.

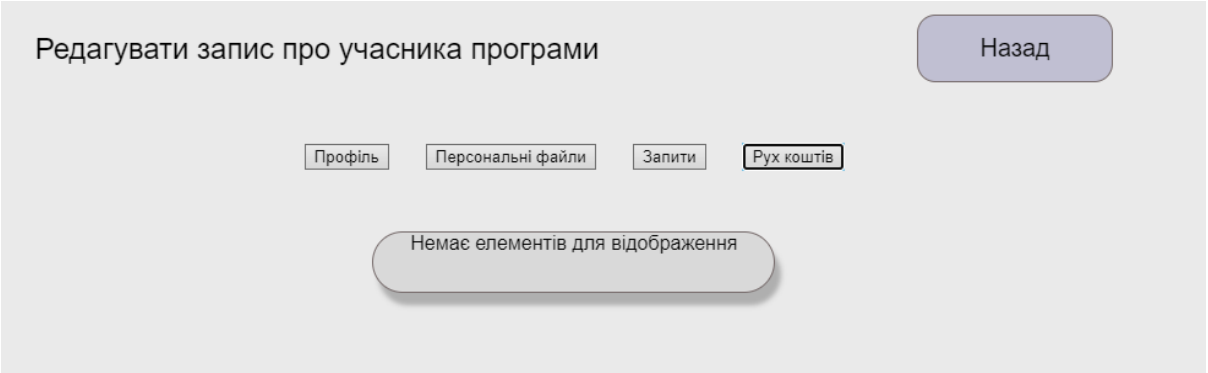

Рис 3.13. Панель з рухом коштів

Застосунок надає можливість переглянути всі запити від усіх користувачів отримані благодійним фондом. За необхідності, вони можуть бути відфільтровані за проміжком дат, статусом, або довільним ключем.

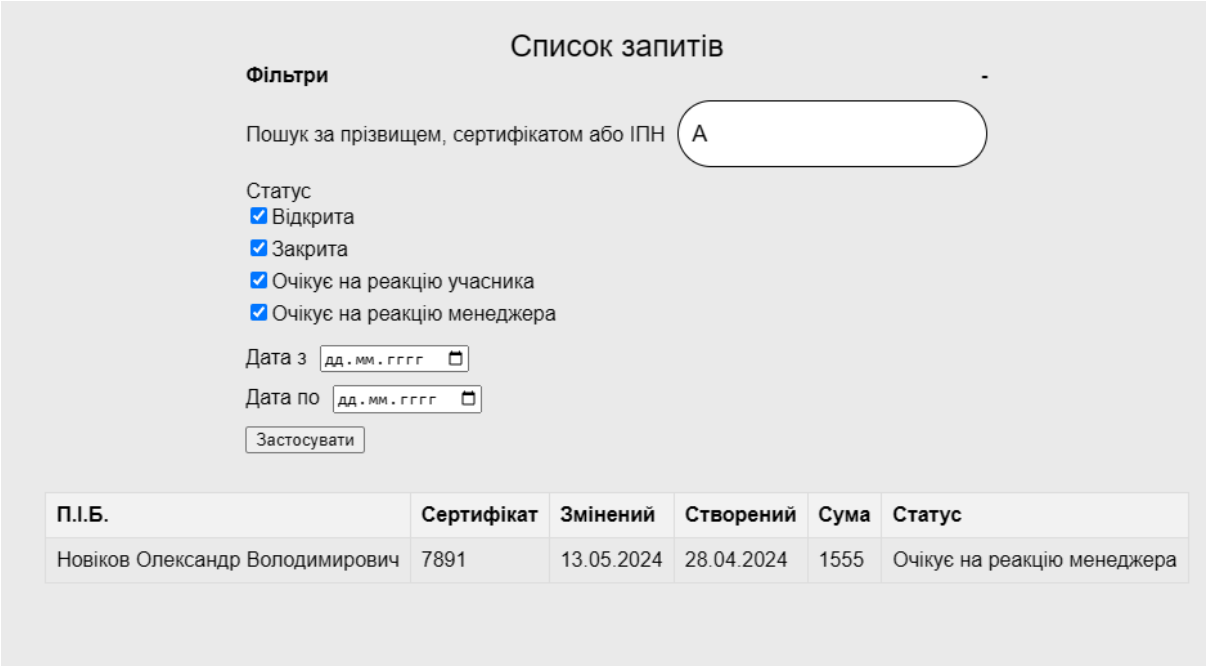

Рис 3.14. Сторінка списку запитів загалом по фонду

Список запитів можна отримати виконавши код наведений у лістингу 3.8.

Лістинг 3.8 Typescript код отримання списку запитів:

```
await $fetch(`${baseUrl}requests`, {
  method: "POST",
  body: {
    query: filter.value.query,
    status: selectedStatuses,
    date_start: filter.value.date_start,
    date_end: filter.value.date_end
  },
  headers: {
    "Authorization": "Bearer " + access_token.value,
    "Content-Type": "application/json"
  }
})
.then(response => {
  requests.value = response.data;
})
.catch(err => {
  console.log(err);
  if (err.status == 401) navigateTo('/login');
});
```

Також, застосунок дозволяє переглянути рух коштів загалом по розрахунковому рахунку фонду.

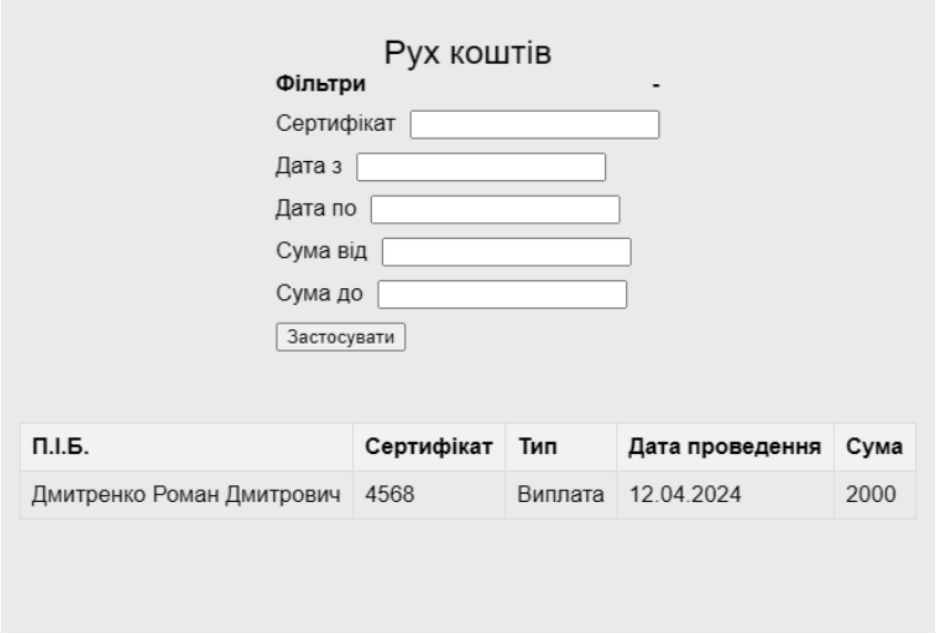

Рис 3.15. Книга руху коштів загалом по фонду

В тому разі, якщо поточний користувач є супер адміністратором, він може керувати обліковими записами інших користувачів, відвідавши відповідну частину сайту за посиланням на головному меню. Сторінка списку користувачів наведена на рисунку 3.16.

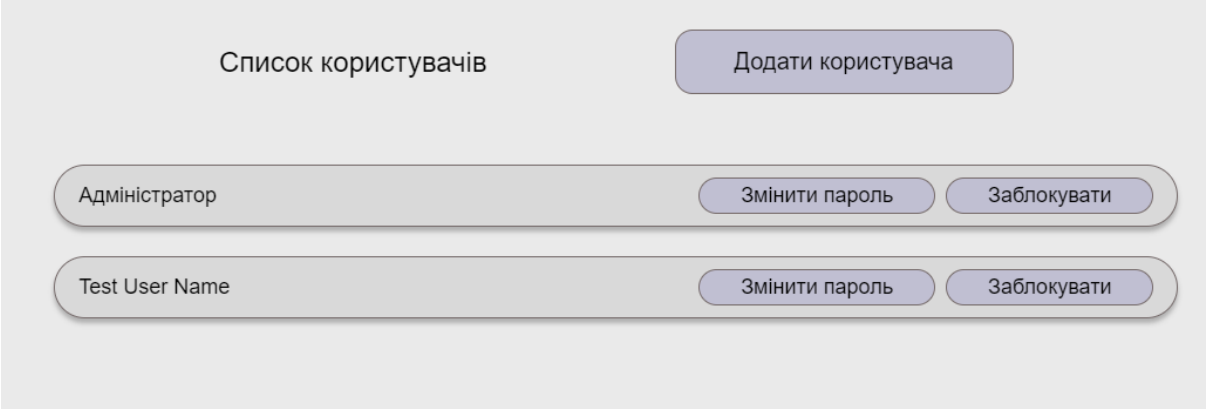

Рис 3.16. Сторінка зі списком облікових записів користувачів

На цій сторінці можна переглянути інформацію про наявних користувачів, а також виконати додавання чи редагування інформації про облікові записи натиснувши на ім’я облікового запису у списку.

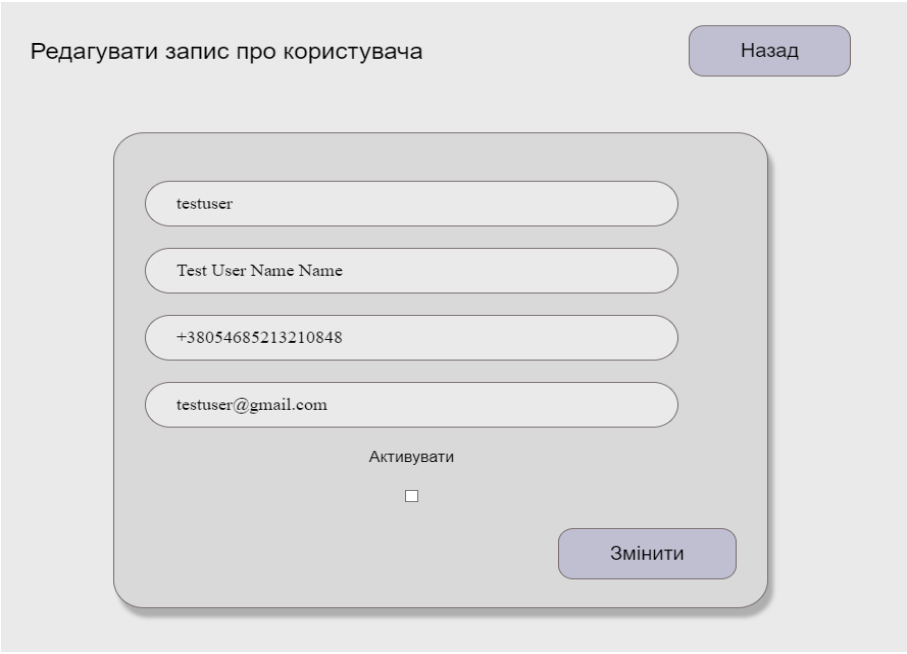

Рис 3.17. Сторінка редагування облікового запису наявного користувача

Можна також за необхідності змінити пароль у вже наявному обліковому записі.

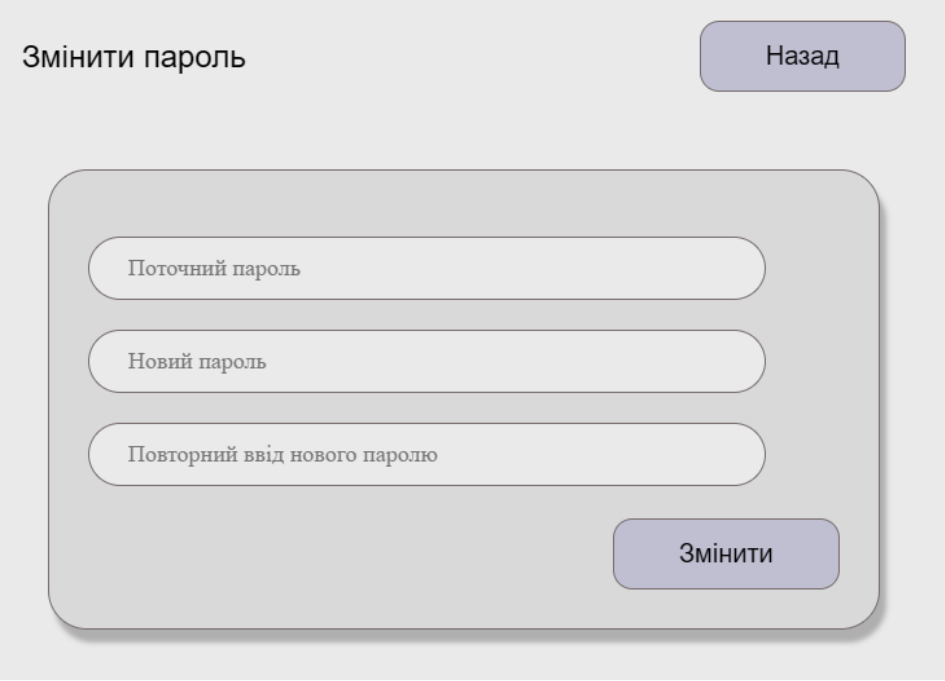

Рис 3.18. Сторінка зміни пароля

3.4 Висновки до розділу 3

Було досліджено архітектуру вебзастосунку, включаючи його складові, зміст і принцип роботи. Також була проаналізована схема HTTP маршрутизації, та алгоритми взаємодії структурних елементів програми. Описано структуру проєкту, включаючи всі каталоги і файли, з яких він складається, з детальним описом кожного з них.

Також було розроблено опис використання цього проекту, включаючи його компоненти та сторінки. Було створено керівництво користувача, яке дозволяє користувачам ознайомитися з функціональністю адміністративної частини інформаційної системи «Каса взаємодопомоги» та отримати більшу впевненість у його використанні та взаємодії з ним. Керівництво надає

детальні вказівки щодо використання можливостей застосунку та розуміння елементів інтерфейсу.

Крім того, було проведено тестування можливих сценаріїв взаємодії користувачів з застосунком. Результати тестування були успішними і відповідали очікуванням.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи "Розробка адміністративної частини інформаційної системи «Каса взаємодопомоги»" було успішно реалізовано ряд завдань та досягнуто мету роботи.

Розроблений Nuxt-застосунок адміністративного сайту благодійного фонду взаємодопомоги дозволяє виконувати операції з обробки інформації про поточну діяльність благодійного фонду, а саме:

- роботу зі списком програм та інформацією про програми;
- роботу зі списком учасників благодійного фонду та окремими записами про учасників;
- обробку запитів від учасників;
- роботу з поточною фінансовою інформацією;
- керування обліковими записами адміністраторів системи з боку фонду.

Завдяки використанню фреймворку Nuxt3 та сучасної мови програмування Typescript та мови розмітки стилів CSS3, сайт може бути використаний у будь-якому сучасному браузері, що робить його зручним інструментом підтримки поточної діяльності благодійного фонду.

Було проведено тестування функціоналу застосунку. Результати тестування були успішними, що підтверджує його стабільну та ефективну роботу.

Документація, включаючи опис архітектури застосунку, інструкції для користувачів та опис функціональності, була ретельно підготовлена. Це допоможе користувачам швидко ознайомитися з програмним продуктом та зручно використовувати його.

Загальним висновком є те, що розроблена адміністративна частина «Каси взаємодопомоги» відповідає поставленим вимогам та має потенціал для подальшого розвитку. У проєкті застосовані сучасні технології веброзробки і відповідні методи побудови архітектури вебзастосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rankins M. G. We Keep Us Safe, Venmo Doesn't: The Impact on Peer-to-Peer Payment Apps on Mutual Aid and Community Organizing. 2023. 27 N.C. BANKING INST. p. 240.

2. Mutual Aid 101: History, Politics, and Organizational Structures of Community Care. [Електронний ресурс] – Режим доступу: https://cunyurbanfoodpolicy.org/news/2023/08/22/mutual-aid-101-history-politics-and-organizational-structures-of-community-care/

3. Каса взаємодопомоги організації «Ліга майстрів». [Електронний ресурс] – Режим доступу: https://ligamaistriv.com.ua/kasa-vzayemodopomogy/

4. Допомога справою [Електронний ресурс] – Режим доступу: https://dopomoga.zp.ua/

5. Авраменко В.С., Авраменко А.С. Проектування інформаційних систем: навчальний посібник / В.С. Авраменко, А.С. Авраменко. – Черкаси: Черкаський національний університет ім. Б. Хмельницького, 2017. – 434 с.

6. Unhelkar B. Software Engineering with UML. – Auerbach Publications, 2018. – 427 p.

7. Vue.js [Електронний ресурс] – Режим доступу: https://vuejs.org/guide/introduction.html

8. Nuxt.js [Електронний ресурс] – Режим доступу: https://nuxtjs.ir/guide

9. TypeScript [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/

10. JavaScript [Електронний ресурс] – Режим доступу: https://learn.javascript.com.

11. Mancini D.R. Implementing a Clean Architecture Modular Application in Nuxt/Vue Typescript Part 1: Domain Layer. 2021. [Електронний ресурс] – Режим доступу: https://dirodriguezm.gitlab.io/nuxt-clean-architecture.html

12. Patel K. Universal application code structure in Nuxt.js. 2018. [Електронний ресурс] – Режим доступу: https://medium.com/free-code-camp/universal-application-code-structure-in-nuxt-js-4cd014cc0baa

13. Server-Side Rendering [Електронний ресурс] – Режим доступу: https://vuejs.org/guide/scaling-up/ssr.html

14. Трофименко О. Г., Козін О. Б., Задерейко О. В., Плачінда О. Є. Веб-технології та веб-дизайн : навч. посібник. Одеса: Фенікс, 2019. 284 с.

15. Robbins J.N. Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics. O'Reilly Media. 2018. p. 1-420.

16. Бородкіна І., Бородкін О. Web-технології та Web-дизайн. Ліра-К, 2019. 1090 с.

17. Crute A., Johnson F. Coding HTML CSS JavaScript Made Easy. Web, Apps and Desktop. Flame Tree: 2016. 256 p.

18. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення: ДСТУ 3008-95. – Чинний від 1996–01–01. – К.: Держстандарт України, 1996. – 39 с.

19. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги: ДСТУ 3582-97. – Чинний від 1998–07–01. – К.: Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

ДОДАТОК А. ЛІСТИНГ

Фрагмент add_program.vue

```
<template>
  <div class="screen">
    <div class="move_right">
      <div class="category">
        <h2>Додати програму</h2>
        <input type="submit" class="purple_btn" value="Назад" @click="$router.back()">
      </div>
    </div>
    <div class="new_program">
      <div class="prog_form">
        <div class="inputs">
          <input v-model="newProgram.program_name" type="text" name="name" placeholder="Назва програми">
          <input v-model="newProgram.month_price" type="number" name="cost" placeholder="Вартість на добу">
        </div>
        <div class="load_file">
          <label for="file">Завантажити файл з умовами</label>
          <input ref="FileInput" v-on:change="changeFile" type="file" class="purple_btn" id="file" style="display:none;">
        </div>
      </div>
      <input type="submit" class="purple_btn submit" value="Створити" @click="addProgram">
    </div>
  </div>
</template>

<script setup lang="ts">

import { useAuthStore } from "~/store/auth";
import { ref, onMounted } from '#imports';
import { storeToRefs } from 'pinia';
const authStore = useAuthStore();
const { access_token } =  storeToRefs(authStore);
const { params } = useRoute();

interface Program {
  program_name: string;
  month_price: number;
  new_program_file: File;
  is_active: bool,
};

let newProgram: Program = {
  program_name: '',
  month_price: 0.0,
  new_program_file: null,
  is_active: true
};

const FileInput = ref(null);

const changeFile = () => {
  newProgram.new_program_file = FileInput.value.files[0];
}

const router = useRouter();

const addProgram = async() => {
  const config = useRuntimeConfig();
  const baseUrl = config.public.baseURL;
  let formData = new FormData();
  for (let key of Object.keys(newProgram)) {
    formData.append(key, newProgram[key]);
  }
  await $fetch(`${baseUrl}program`, {
    method: "PUT",
    body: formData,
    headers: {
      "Authorization": "Bearer "+access_token.value,
      "Content-Disposition": `form-data; name="new_program_file"; filename=${newProgram.new_program_file.name}`
    }
  })
  .then (response => {
    navigateTo('/program_list');
  })
  .catch(err => {
```

```
      console.log(err)
      if (error.status == 401)
        navigateTo('/login')
    })
};

useHead({
  title: "Нова програма",
  meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
  ],
  link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
  ],
  googleFonts: {
    preconnect: true,
    Inter: {
      wght: [100, 300, 400, 600, 700, 900],
      display: 'swap',
    },

  },
});

definePageMeta({
 middleware: 'auth',
 layout: 'default'
});

</script>
```

Фрагмент add_user.vue

```
<template>
  <div class="screen">
    <div class="move_right">
      <div class="category" id="for_add_gap">
        <h2>Додати користувача</h2>
        <input type="submit" class="purple_btn" value="Назад" @click="$router.back()">
      </div>
    </div>
    <div class="new_program">
      <div class="prog_form">
        <div class="inputs">
          <input v-model="newUser.login" type="text" name="login" id="login" placeholder="Логін">
          <input v-model="newUser.name" type="text" name="name" placeholder="ПІБ">
          <input v-model="newUser.phone" type="text" name="phone" id="phone" placeholder="Контактний номер телефону">
          <input v-model="newUser.email" type="text" name="email" id="email" placeholder="E-mail">
          <label for="activate">Активувати</label>
          <input v-model="newUser.is_active" type="checkbox" name="activate" id="activate" checked>
        </div>
      </div>
      <input type="submit" class="purple_btn submit" value="Додати" @click="addUser">
    </div>
  </div>
</template>

<script setup lang="ts">

import { useAuthStore } from "~/store/auth";
import { ref, onMounted } from '#imports';
import { storeToRefs } from 'pinia';
const authStore = useAuthStore();
const { access_token } =  storeToRefs(authStore);
const { params } = useRoute();

interface User {
 login: string,
 name: string,
 phone: string,
 email: string,
 is_active: bool
}

let newUser: User = {
 login: '',
 name: '',
 phone: '',
 email: '',
 is_active: true
};

const router = useRouter();
```

```
const addUser = async() => {
   const config = useRuntimeConfig();
   const baseUrl = config.public.baseURL;
   await $fetch(`${baseUrl}user`, {
      method: "PUT",
      body: newUser,
      headers: {
         "Authorization": "Bearer "+access_token.value,
         "Content-Type": "application/json"
      }
   })
   .then (response => {
      router.push('/user_list');
   })
   .catch(err => {
      console.log(err)
      if (error.status == 401)
         navigateTo('/login')
   })
};

useHead({
   title: "Додавання користувача",
   meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
   ],
   link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
   ],
   googleFonts: {
      preconnect: true,
      Inter: {
         wght: [100, 300, 400, 600, 700, 900],
         display: 'swap',
      },
   },
});

definePageMeta({
 middleware: 'auth',
 layout: 'default'
});

</script>
```

Фрагмент appeal_history.vue

```
<template>
   <div class="screen">
      <div class="move_right">
         <div class="category" id="for_info_gap">
            <h2>Учасник №12345</h2>
            <input type="submit" class="purple_btn"
value="Назад" @click="$router.back()">
         </div>
      </div>
      <div class="program_list">
         <div class="member">
            <p>Звернення 1</p>
            <p>Дата: 00.00.0000</p>
         </div>
         <div class="member">
            <p>Звернення 2</p>
            <p>Дата: 00.00.0000</p>
         </div>
         <div class="member">
            <p>Звернення 3</p>
            <p>Дата: 00.00.0000</p>
         </div>
         <div class="member">
            <p>Звернення 4</p>
            <p>Дата: 00.00.0000</p>
         </div>
      </div>
   </div>
</template>

<script setup lang="ts">

useHead({
   title: "Історія звернень",
   meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
   ],
   link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
   ],
   googleFonts: {
      preconnect: true,
      Inter: {
```

```
        wght: [100, 300, 400, 600, 700, 900],
        display: 'swap',
      },

  },
});

definePageMeta({
 layout: 'default'
});

</script>
```

Фрагмент appeal.vue

```
<template>
  <div class="menu">
    <p>Меню</p>
    <div class="menu_list">
      <div class="menu_btn">
        <img src="/images/program.svg">
        <label for="btn">Програми</label>
        <input type="button" name="btn" id="btn" style="display:none;">
      </div>
      <div class="menu_btn">
        <img src="/images/user.svg">
        <label for="btn">Користувачі</label>
        <input type="button" name="btn" id="btn" style="display:none;">
      </div>
      <div class="menu_btn">
        <img src="/images/people.svg">
        <label for="btn">Учасники програми</label>
        <input type="button" name="btn" id="btn" style="display:none;">
      </div>
      <div class="menu_btn">
        <img src="/images/ask.svg">
        <label for="btn">Звернення</label>
        <input type="button" name="btn" id="btn" style="display:none;">
      </div>
      <div class="menu_btn">
        <img src="/images/money.svg">
        <label for="btn">Книга руху коштів</label>
        <input type="button" name="btn" id="btn" style="display:none;">
      </div>
    </div>
  </div>
  <div class="screen">
    <div class="move_right">
      <div class="category" id="for_info_gap">
        <h2>Звернення №3456</h2>
        <input type="submit" class="purple_btn" value="Назад" @click="$router.back()">
      </div>
    </div>
    <div class="new_program">
      <div class="user_info">
        <p>ПІБ:</p>
        <p class="with_backgr">прізвище ім’я по-батькові</p>
      </div>
      <div class="user_info">
        <p>Дата подачі:</p>
        <p class="with_backgr">00.00.0000</p>
      </div>
      <div class="user_info">
        <p>Сума компенсації:</p>
        <p class="with_backgr">10 000 грн</p>
      </div>
      <div class="user_info">
        <p>Коментар:</p>
        <p class="with_backgr">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin dictum sapien nec felis vestibulum viverra. Aliquam ut enim justo. Sed sed ligula est.</p>
      </div>
    </div>
  </div>
</template>

<script setup lang="ts">

useHead({
  title: "Звернення",
  meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
```

```
  ],
  link: [
   { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
  ],
  googleFonts: {
    preconnect: true,
    Inter: {
      wght: [100, 300, 400, 600, 700, 900],
      display: 'swap',
    },

  },
});

definePageMeta({
 layout: 'default'
});

</script>
```

Фрагмент block_user.vue

```
<template>
  <div class="screen">
    <div class="block_user">
      <h5>Підтвердження блокування</h5>
      <p>Ви дійсно хочете заблокувати Користувача 1?</p>
        <div class="user_btn" id="block_btn">
          <input type="button" class="purple_btn_user" value="Назад">
          <input type="button" class="purple_btn_user" value="Заблокувати">
        </div>
    </div>
    <div class="move_right">
      <div class="category">
        <h2>Список користувачів</h2>
        <input type="button" class="purple_btn" value="Додати користувача">
      </div>
    </div>
    <div class="program_list">
      <div class="user">
        <p>Користувач 1</p>
        <div class="user_btn">
          <input type="button" class="purple_btn_user" value="Змінити пароль">
          <input type="button" class="purple_btn_user" value="Заблокувати">
        </div>
      </div>
      <div class="user">
        <p>Користувач 2</p>
        <div class="user_btn">
          <input type="button" class="purple_btn_user" value="Змінити пароль">
          <input type="button" class="purple_btn_user" value="Заблокувати">
        </div>
      </div>
      <div class="user">
        <p>Користувач 3</p>
        <div class="user_btn">
          <input type="button" class="purple_btn_user" value="Змінити пароль">
          <input type="button" class="purple_btn_user" value="Заблокувати">
        </div>
      </div>
      <div class="user">
        <p>Користувач 4</p>
        <div class="user_btn">
          <input type="button" class="purple_btn_user" value="Змінити пароль">
          <input type="button" class="purple_btn_user" value="Заблокувати">
        </div>
      </div>
    </div>
  </div>
</template>

<script setup lang="ts">

useHead({
  title: "Про нас",
  meta: [
   { charset: 'utf-8' },
   { viewport: 'width=device-width, initial-scale=1' },
  ],
  link: [
```

```
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
  ],
  googleFonts: {
    preconnect: true,
    Inter: {
      wght: [100, 300, 400, 600, 700, 900],
      display: 'swap',
    },

  },
});
</script>
```

Фрагмент cash_flow.vue

```
<template>
 <div class="screen">
  <h2 id="member_title">Рух коштів</h2>
  <div class="filter-container">
   <div class="filter-header" @click="toggleFilter">
    Фільтри <span>{{ filterVisible ? '-' : '+' }}</span>
   </div>
   <div v-show="filterVisible" class="filter-content">
    <div>
     <label for="cert_input">Сертифікат</label>
     <input v-model="filter.certificate" type="text" id="cert_input" name="cert_input">
    </div>
    <div>
     <label for="date_start">Дата з</label>
     <input v-model="filter.date_start" id="date_start" name="date_start">
    </div>
    <div>
     <label for="date_end">Дата по</label>
     <input v-model="filter.date_end" id="date_end" name="date_end">
    </div>
    <div>
     <label for="amount_start">Сума від</label>
     <input v-model="filter.amount_start" id="amount_start" name="amount_start">
    </div>
    <div>
     <label for="amount_end">Сума до</label>
     <input v-model="filter.amount_end" id="amount_end" name="amount_end">
    </div>
    <button @click="setFilter">Застосувати</button>
   </div>
  </div>
  <div v-if="payments && payments.length > 0">
   <table v-if="requests && requests.length > 0" class="prog_table">
    <thead>
     <tr>
      <th>Дата/час</th>
      <th>П.І.Б.</th>
      <th>Сертифікат</th>
      <th>Тип операції</th>
      <th>Сума</th>
     </tr>
    </thead>
    <tbody>
     <tr v-for="payment in payments" :key="request.id" @click="updateRequest(request)">
      <td>{{ payment.updated_at }}</td>
      <td>{{ payment.member.last_name + ' ' + payment.member.first_name + ' ' + payment.member.middle_name }}</td>
      <td>{{ payment.member.certificate }}</td>
      <td>{{ payment.payment_type.name }}</td>
      <td>{{ payment.amount }}</td>
     </tr>
    </tbody>
   </table>
  </div>
  <div v-else>
   Немає елементів для відображення
  </div>
 </div>
</template>


<script setup lang="ts">

import { ref, onMounted } from '#imports';
import { useAuthStore } from "~/store/auth";
import { storeToRefs } from 'pinia';
import { useRouter } from 'vue-router';
import { useDataStore } from "~/store/data";
```

```
const authStore = useAuthStore();
const { access_token } =  storeToRefs(authStore);
const dataStore = useDataStore();
let { pass_data } = storeToRefs(dataStore);
const payments = ref([])
const filter = ref({
    certificate: '',
    date_start: '',
    date_end: '',
    amount_start: '',
    amount_end: ''
})

const getPayments = async() => {
    const config = useRuntimeConfig();
    const baseUrl = config.public.baseURL;
    await $fetch(`${baseUrl}payments`, {
        method: "POST",
        body: {
            'filter': filter
        },
        headers: {
            "Authorization": "Bearer "+access_token.value,
            "Content-Type": "application/json"
        }
    })
    .then (response => {
        payments.value = response.data;
    })
    .catch(err => {
        console.log(err)
        if (error.status == 401)
            navigateTo('/login')
    })
}

onMounted(async () => {
    getPayments()
})

useHead({
    title: "Рух коштів",
    meta: [
      { charset: 'utf-8' },
      { viewport: 'width=device-width, initial-scale=1' },
    ],
    link: [
      { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
    ],
    googleFonts: {
        preconnect: true,
        Inter: {
            wght: [100, 300, 400, 600, 700, 900],
            display: 'swap',
        },

    },
});

definePageMeta({
  layout: 'default'
});

const filterVisible = ref(false);

const toggleFilter = () => {
  filterVisible.value = !filterVisible.value;
};

</script>

<style scoped>
.filter-container {
  margin-bottom: 20px;
}

.filter-header {
  cursor: pointer;
  font-weight: bold;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.filter-content {
  padding: 10px 0;
}

.filter-content div {
  margin-bottom: 10px;
}
```

```
.filter-content label {
  margin-right: 10px;
}

</style>
```

Фрагмент change_password.vue

```
<template>
   <div class="screen">
      <div class="move_right">
         <div class="category" id="for_change_gap">
            <h2>Змінити пароль</h2>
            <input type="submit" class="purple_btn" value="Назад" @click"$router.back()">
         </div>
      </div>
      <div class="new_program">
         <div class="prog_form">
            <div class="inputs">
               <input type="text" name="old_password" placeholder="Поточний пароль" v-model="changeForm.old_password">
               <input type="text" name="new_password" id="new_password" placeholder="Новий пароль" v-model="changeForm.new_password">
               <input type="text" name="confirm_password" id="confirm_password" placeholder="Повторний ввід нового паролю">
            </div>
         </div>
         <input type="submit" class="purple_btn submit" value="Змінити" @click="changePassword">
      </div>
   </div>
</template>

<script setup lang="ts">

import { useAuthStore } from "~/store/auth";
import { useDataStore } from "~/store/data";
import { ref, onMounted } from '#imports';
import { storeToRefs } from 'pinia';
const authStore = useAuthStore();
const { access_token } =  storeToRefs(authStore);
const { params } = useRoute();

const dataStore = useDataStore();
let{ pass_data } = storeToRefs(dataStore);

interface ChangePasswordForm {
   id: number,
   old_password: string,
   new_password: string,
};

let changeForm: ChangePasswordForm = {
   id: 0,
   old_password: "",
   new_password: ""
};

onMounted(async () => {
//    console.log(pass_data.value.id);
//    changeForm.id = pass_data.value.id;
});

useHead({
   title: "Зміна пароля",
   meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
   ],
   link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
   ],
   googleFonts: {
      preconnect: true,
      Inter: {
         wght: [100, 300, 400, 600, 700, 900],
         display: 'swap',
      },

   },
});

definePageMeta({
  middleware: [ 'auth', 'data'],
  layout: 'default'
});
```

```
const changePassword = async () => {
   const config = useRuntimeConfig();
   const baseUrl = config.public.baseURL;
   changeForm.id = pass_data.value.id;
   await $fetch(`${baseUrl}user/change_password`, {
      method: "POST",
      body: changeForm,
      headers: {
         "Authorization": "Bearer "+access_token.value,
         "Content-Type": "application/json"
      }
   })
   .then (response => {
      navigateTo("/user_list");
   })
   .catch(err => {
      console.log(err)
      if (error.status == 401)
         navigateTo('/login')
   })
};
</script>
```

Фрагмент documents.vue

```
<template>
   <div class="screen">
      <div class="move_right">
         <div class="category" id="for_info_gap">
            <h2>Учасник №12345</h2>
            <input type="submit" class="purple_btn"
value="Назад" @click="$router.back()">
         </div>
      </div>
      <div class="new_program">
         <div class="docs_col">
            <h4>Документи</h4>
            <div class="docs_row">
               <img src="/images/for_docs.png">
               <img src="/images/for_docs.png">
               <img src="/images/for_docs.png">
            </div>
            <div class="docs_row">
               <img src="/images/for_docs.png">
               <img src="/images/for_docs.png">
               <img src="/images/for_docs.png">
            </div>
         </div>
      </div>
   </div>
</template>

<script setup lang="ts">

useHead({
   title: "Про нас",
   meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
   ],
   link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
   ],
   googleFonts: {
      preconnect: true,
      Inter: {
         wght: [100, 300, 400, 600, 700, 900],
         display: 'swap',
      },
   },
});

definePageMeta({
 layout: 'default'
});

</script>
```

Фрагмент index.vue

```
<template>
   <div>
      <for_dash :texts="textArray"></for_dash>
   </div>
</template>

<script setup lang="ts">
import for_dash from '~/components/for_dash.vue';

definePageMeta({
 middleware: [ 'auth', 'data'],
 name: 'dashboard',
 components: { for_dash },
```

```
 data() {
  return {
   textArray: ['Надходження за добу: 10000 грн', 'Необроблені звернення: 5', 'Нові запити на приєднання до програм: 15'],
  };
 }
});
</script>
```

Фрагмент login.vue

```
<template>
 <div class="form_center">
  <div class="content">
   <h1 class="central_txt">Увійти в акаунт</h1>
   <form class="login" @submit.prevent="login">
    <div class="inputs">
     <input type="text" name="login" id="login" placeholder="Логін" v-model="loginForm.login">
     <input type="password" name="password" id="password" placeholder="Пароль" v-model="loginForm.password">
    </div>
    <input type="submit" class="purple_btn submit" value="Увійти">
   </form>
  </div>
 </div>

</template>

<script setup lang="ts">
import { useAuthStore } from "~/store/auth";
const authStore = useAuthStore();
const router = useRouter();

interface loginForm {
 login: string;
 password: string;
};

let loginForm: loginForm = {
 login: "",
 password: "",
};

function login() {
 authStore
  .login(loginForm)
  .then((_response) => navigateTo("/"))
  .catch((error) => console.log("API error", error));
}

definePageMeta({
 middleware: [],
 layout: 'anonymous'
});

useHead({
  title: "Вхід",
  meta: [
   { charset: 'utf-8' },
   { viewport: 'width=device-width, initial-scale=1' },
  ],
  link: [
   { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
  ],
  googleFonts: {
    preconnect: true,
    Inter: {
      wght: [100, 300, 400, 600, 700, 900],
      display: 'swap',
    },

  },
});

</script>
```

Фрагмент member_list.vue

```
<template>
 <div class="screen">
  <h2 id="member_title">Список учасників</h2>
  <div class="buttons">
   <input type="button" class="purple_btn" value="Необроблені" @click="showUnprocessed">
   <input type="button" class="purple_btn" value="Пошук" @click="showSearch">
```

```
  </div>
  <div v-if="showSearchInput" class="search">
   <input type="text" v-model="filter.query" placeholder="Введіть дані для пошуку" id="search_input">
   <!-- Випадаюче меню з чекбоксами -->
   <div class="dropdown">
    <button class="dropbtn" @click="toggleFilterDropdown">Оберіть поля</button>
    <div class="dropdown-content" v-show="showDropdown">
     <div>
      <input type="checkbox" id="full_name" v-model="filter.fields.name" value="full_name">
      <label for="full_name">П.І.Б.</label>
     </div>
     <div>
      <input type="checkbox" id="passport_data" v-model="filter.fields.passport" value="passport_data">
      <label for="passport_data">Паспортні дані</label>
     </div>
     <div>
      <input type="checkbox" id="inn" v-model="filter.fields.inn" value="inn">
      <label for="inn">ІНН</label>
     </div>
     <div>
      <input type="checkbox" id="certificate" v-model="filter.fields.certificate" value="certificate">
      <label for="certificate">Номер сертифікату</label>
     </div>
     <div>
      <input type="checkbox" id="iban" v-model="filter.fields.iban" value="iban">
      <label for="iban">IBAN</label>
     </div>
    </div>
   </div>
   <!-- Кінець випадаючого меню -->
   <input type="button" class="purple_btn" value="Знайти" @click="searchMember">
  </div>
  <div class="program_list">
   <p v-if="showSearchInput" id="result">Результати пошуку:</p>
   <div v-for="member in members" :key="member.id" class="member" @click="updateMember(member)">
    <p><img v-if="member.created_at==member.updated_at && !member.is_submited" src="/images/new.svg">
     <img v-if="member.created_at!=member.updated_at && !member.is-submited" src="/images/changes.svg">
     <img v-if="member.has_new_request > 0" src="/images/request.svg"></p>
    <p>{{ member.last_name }} {{ member.first_name }}</p>
    <p v-if="member.certificate"><nuxt-link @click="updateMember(member)">№{{ member.certificate }}</nuxt-link></p>
    <p v-else>?</p>
    <p v-if="member.inn">ІПН: <nuxt-link @click="updateMember(member)">{{ member.inn }}</nuxt-link></p>
    <p v-else>?</p>
   </div>
  </div>
 </div>
</template>

<script setup lang="ts">
 import { useDataStore } from "~/store/data";
 import { ref, onMounted } from '#imports';
 import { storeToRefs } from 'pinia'
 import { useAuthStore } from "~/store/auth";
 const authStore = useAuthStore();
 const { access_token }=  storeToRefs(authStore);
 const dataStore = useDataStore();
 let { pass_data, pass_filter  } = storeToRefs(dataStore);

 interface Member {
   id: number;
   first_name: string;
   middle_name: string;
   last_name: string;
   passport_data: string;
   has_new_request: number;
   inn: string;
   certificate: string;
   expiration_date: string;
```

```
    birth_date: string;
    actual_iban: string;
    email: string;
    is_active: bool;
    updated_at: string;
    is_active: bool;
    is_submited: bool;
  };
  let members = ref([]);
  let filter = {
    query: '',
    fields: {
      passport: false,
      name: false,
      inn: false,
      certificate: false,
      iban: false
    }
  };

  const searchMember = async () => {
    const config = useRuntimeConfig();
    const baseUrl = config.public.baseURL;
    members.value = [];
    await $fetch(`${baseUrl}members`, {
        method: "POST",
        body: {
          "filter": filter
        },
        headers: {
          "Authorization": "Bearer "+access_token.value,
          "Content-Type": "application/json"
        }
      })
      .then(response => {
        members.value = response.data;
      })
      .catch(error => {
        console.log('Помилка при отриманні даних:',
error.message);
        if (response.status == 401)
          navigateTo('/login')
      });
  };

  const updateMember = async (member) => {
    dataStore.setMemberId(member);
    navigateTo("/update_member");
  };

  onMounted(async () => {
    showUnprocessed();
  });

  useHead({
    title: "Список учасників",
    meta: [
      { charset: 'utf-8' },
      { viewport: 'width=device-width, initial-scale=1' },
    ],
    link: [
      { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
    ],
    googleFonts: {
      preconnect: true,
      Inter: {
        wght: [100, 300, 400, 600, 700, 900],
        display: 'swap',
      },

    },
  });

  definePageMeta({
    middleware: [ 'auth', 'data' ],
    layout: 'default'
  });

  const showSearchInput = ref(false);
  const showDropdown = ref(false); // змінна для
відображення/приховання випадаючого меню

  const showUnprocessed = () => {
    showSearchInput.value = false;
    searchMember();
  };

  const showSearch = () => {
    showSearchInput.value = true;
  };

  // розгортає або згортає список чекбоксів
```

```
  const toggleFilterDropdown = () => {
    showDropdown.value = !showDropdown.value;
  };
</script>

<style scoped>
.dropdown {
  position: relative;
  display: inline-block;
}

.dropbtn {
  background-color: #4CAF50;
  color: white;
  padding: 10px;
  font-size: 16px;
  border: none;
  cursor: pointer;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0 8px 16px 0 rgba(0,0,0,0.2);
  z-index: 1;
}

.dropdown-content div {
  padding: 12px 16px;
  display: flex;
  align-items: center;
}

.dropdown-content input[type="checkbox"] {
  margin-right: 8px;
}

.dropdown-content label {
  margin: 0;
}

.dropdown:hover .dropdown-content {
  display: block;
}
</style>
```

Фрагмент payment_history.vue

```
<template>
  <div class="screen">
    <div class="move_right">
      <div class="category" id="for_info_gap">
        <h2>Учасник №12345</h2>
        <input type="submit" class="purple_btn" value="Назад" @click="$router.back()">
      </div>
    </div>
    <div class="program_list">
      <div class="member">
        <p>Платіж 1</p>
        <p>Дата: 00.00.0000</p>
      </div>
      <div class="member">
        <p>Платіж 2</p>
        <p>Дата: 00.00.0000</p>
      </div>
      <div class="member">
        <p>Платіж 3</p>
        <p>Дата: 00.00.0000</p>
      </div>
      <div class="member">
        <p>Платіж 4</p>
        <p>Дата: 00.00.0000</p>
      </div>
    </div>
  </div>
</template>

<script setup lang="ts">

useHead({
  title: "Історія платежів",
  meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
  ],
  link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
  ],
  googleFonts: {
```

```
    preconnect: true,
    Inter: {
      wght: [100, 300, 400, 600, 700, 900],
      display: 'swap',
    },

  },
});

definePageMeta({
 layout: 'default'
});

</script>
```

Фрагмент program_list.vue

```
<template>
  <div class="screen">
    <div class="move_right">
      <div class="category">
        <h2>Список благодійних програм</h2>
        <input type="button" class="purple_btn" value="Додати програму" @click="navigateTo('/add_program')">
      </div>
    </div>
    <div class="program_list">
      <div v-for="program in programs" class="program">
        <p><nuxt-link @click="updateProgram(program)">{{ program.program_name }}</nuxt-link></p>
        <img src="/images/close.svg">
      </div>
    </div>
  </div>
</template>

<script setup lang="ts">

import { ref, onMounted } from '#imports';
import { storeToRefs } from 'pinia'
import { useAuthStore } from "~/store/auth";
import { useDataStore } from "~/store/data";

const dataStore = useDataStore();
let { pass_data } = storeToRefs(dataStore);
const authStore = useAuthStore();
const { access_token }=  storeToRefs(authStore);

interface Program {
  id: number;
  program_name: string;
  month_price: number;
  program_file: string;
  updated_at: string;
  updated_by: object;
};

const programs = ref([]);

onMounted(async () => {
  const config = useRuntimeConfig();
  const baseUrl = config.public.baseURL;
  console.log("!!!!!!!");
  pass_data = null;
  await $fetch(`${baseUrl}programs`, {
      method: "GET",
      headers: {
        "Authorization": "Bearer "+access_token.value,
        "Content-Type": "application/json"
      }
    })
    .then(response => {
      programs.value = response.data;
    })
    .catch(error => {
      console.log('Помилка при отриманні даних:', error.message);
      console.log(error.status)
      if (error.status == 401)
        navigateTo('/login')
    });
});

useHead({
  title: "Програми",
  meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
  ],
```

```
  link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
  ],
  googleFonts: {
    preconnect: true,
    Inter: {
      wght: [100, 300, 400, 600, 700, 900],
      display: 'swap',
    },

  },
});

definePageMeta({
  middleware: [ 'auth', 'data' ],
  layout: 'default',
});

const updateProgram = async(program) => {
  dataStore.setProgramId(program);
  navigateTo("/update_program");
};

</script>
```

Фрагмент request_list.vue

```
<template>
 <div class="screen">
  <h2 id="member_title">Список запитів</h2>
  <div class="filter-container">
   <div class="filter-header" @click="toggleFilter">
    Фільтри <span>{{ filterVisible ? '-' : '+' }}</span>
   </div>
   <div v-show="filterVisible" class="filter-content">
    <div>
     <label for="search_input">Пошук за прізвищем, сертифікатом або ІПН</label>
     <input v-model="filter.query" type="text" id="search_input" name="search_input">
    </div>
    <div>
     <label>Статус</label>
     <div class="checkbox-group">
      <div>
       <input type="checkbox" id="opened" v-model="filter.status.opened">
       <label for="opened">Відкрита</label>
      </div>
      <div>
       <input type="checkbox" id="closed" v-model="filter.status.closed">
       <label for="closed">Закрита</label>
      </div>
      <div>
       <input type="checkbox" id="awaiting_client" v-model="filter.status.awaiting_client">
       <label for="awaiting_client">Очікує на реакцію учасника</label>
      </div>
      <div>
       <input type="checkbox" id="awaiting_agent" v-model="filter.status.awaiting_agent">
       <label for="awaiting_agent">Очікує на реакцію менеджера</label>
      </div>
     </div>
    </div>
    <div>
     <label for="date_start">Дата з</label>
     <input v-model="filter.date_start" id="date_start" name="date_start" type="date">
    </div>
    <div>
     <label for="date_end">Дата по</label>
     <input v-model="filter.date_end" id="date_end" name="date_end" type="date">
    </div>
    <button @click="applyFilter">Застосувати</button>
   </div>
  </div>
  <div>
   <table v-if="requests && requests.length > 0" class="prog_table">
    <thead>
     <tr>
      <th>П.І.Б.</th>
      <th>Сертифікат</th>
      <th>Змінений</th>
      <th>Створений</th>
      <th>Сума</th>
```

```
          <th>Статус</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="request in requests" :key="request.id" @click="updateRequest(request)">
          <td>{{ request.member.last_name + ' ' + request.member.first_name + ' ' + request.member.middle_name }}</td>
          <td>{{ request.member.certificate }}</td>
          <td>{{ request.updated_at }}</td>
          <td>{{ request.created_at }}</td>
          <td>{{ request.suma }}</td>
          <td>{{ request.status.name }}</td>
        </tr>
      </tbody>
    </table>
  </div>
 </div>
</template>



<script setup lang="ts">
import { ref, onMounted } from 'vue';
import { useAuthStore } from "~/store/auth";
import { storeToRefs } from 'pinia';
import { useRouter } from 'vue-router';
import { useDataStore } from "~/store/data";

const authStore = useAuthStore();
const { access_token } = storeToRefs(authStore);
const dataStore = useDataStore();
let { pass_data } = storeToRefs(dataStore);
const requests = ref([]);

const filterVisible = ref(false);
const filter = ref({
 query: '',
 status: {
  opened: false,
  closed: false,
  awaiting_client: false,
  awaiting_agent: false
 },
 date_start: '',
 date_end: ''
});

const getRequests = async () => {
 const config = useRuntimeConfig();
 const baseUrl = config.public.baseURL;
 const selectedStatuses = Object.entries(filter.value.status)
  .filter(([_, value]) => value)
  .map(([key]) => key);
 console.log(selectedStatuses)
 await $fetch(`${baseUrl}requests`, {
  method: "POST",
  body: {
   query: filter.value.query,
   status: selectedStatuses,
   date_start: filter.value.date_start,
   date_end: filter.value.date_end
  },
  headers: {
   "Authorization": "Bearer " + access_token.value,
   "Content-Type": "application/json"
  }
 })
 .then(response => {
  requests.value = response.data;
 })
 .catch(err => {
  console.log(err);
  if (err.status == 401) navigateTo('/login');
 });
};

onMounted(async () => {
 getRequests();
});

const updateRequest = async (request) => {
 dataStore.setRequestId(request);
 navigateTo("/update_request");
};

const toggleFilter = () => {
 filterVisible.value = !filterVisible.value;
};

const applyFilter = () => {
 getRequests();
```

```
};

useHead({
  title: "Список запитів",
  meta: [
    { charset: 'utf-8' },
    { viewport: 'width=device-width, initial-scale=1' },
  ],
  link: [
    { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
  ],
  googleFonts: {
    preconnect: true,
    Inter: {
      wght: [100, 300, 400, 600, 700, 900],
      display: 'swap',
    },
  },
});

definePageMeta({
  layout: 'default'
});
</script>

<style scoped>
.filter-container {
  margin-bottom: 20px;
}

.filter-header {
  cursor: pointer;
  font-weight: bold;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.filter-content {
  padding: 10px 0;
}

.filter-content div {
  margin-bottom: 10px;
}

.filter-content label {
  margin-right: 10px;
}

.checkbox-group {
  display: flex;
  flex-direction: column;
}

.checkbox-group div {
  margin-bottom: 5px;
}


</style>
```

Фрагмент update_user.vue

```
<template>
    <div class="screen">
        <div class="move_right">
            <div class="category" id="for_add_gap">
                <h2>Редагувати запис про користувача</h2>
                <input type="submit" class="purple_btn" value="Назад" @click="$router.back()">
            </div>
        </div>
        <div class="new_program">
            <div class="prog_form">
                <div class="inputs">
                    <input v-model="user.login" type="text" name="login" id="login" readonly>
                    <input v-model="user.name" type="text" name="name" placeholder="ПІБ">
                    <input v-model="user.phone" type="text" name="phone" id="phone" placeholder="Контактний номер телефону">
                    <input v-model="user.email" type="text" name="email" id="email" placeholder="E-mail">
                    <label for="activate">Активувати</label>
                    <input v-model="user.is_active" type="checkbox" name="activate" id="activate" checked>
                </div>
            </div>
            <input type="submit" class="purple_btn submit" value="Змінити" @click="updateUser">
```

```
      </div>
    </div>
</template>

<script setup lang="ts">

import { ref, onMounted } from '#imports';
import { useAuthStore } from "~/store/auth";
import { storeToRefs } from 'pinia';
const authStore = useAuthStore();
const { access_token } =  storeToRefs(authStore);
const { params } = useRoute();
 import { useDataStore } from "~/store/data";

const dataStore = useDataStore();
let { pass_data } = storeToRefs(dataStore);


interface User {
 id: number,
 login: string,
 name: string,
 phone: string,
 email: string,
 is_active: bool
}

let user: User = {
 id: 0,
 login: '',
 name: '',
 phone: '',
 email: '',
 is_active: true
};

onMounted(async () => {
    user.id = pass_data.value.user.id;
    user.login = pass_data.value.user.login;
    user.name = pass_data.value.user.name;
    user.phone = pass_data.value.user.phone;
    user.email = pass_data.value.user.email;
    user.is_active = pass_data.value.user.is_active;
});

const router = useRouter();

const updateUser = async() => {
    const config = useRuntimeConfig();
    const baseUrl = config.public.baseURL;
    await $fetch(`${baseUrl}user`, {
        method: "POST",
        body: user,
        headers: {
            "Authorization": "Bearer "+access_token.value,
            "Content-Type": "application/json"
        }
    })
    .then (response => {
        navigateTo('/user_list');
    })
    .catch(err => {
        console.log(err)
        if (error.status == 401)
            navigateTo('/login')
    })
};

useHead({
    title: "Додавання користувача",
    meta: [
      { charset: 'utf-8' },
      { viewport: 'width=device-width, initial-scale=1' },
    ],
    link: [
      { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
    ],
    googleFonts: {
        preconnect: true,
        Inter: {
            wght: [100, 300, 400, 600, 700, 900],
            display: 'swap',
        },
    },
});

definePageMeta({
  middleware: [ 'auth', 'data' ],
  layout: 'default'
});

</script>
```

Фрагмент user_list.vue

```
<template>
  <div class="screen">
    <div class="move_right">
      <div class="category">
        <h2>Список користувачів</h2>
        <input type="button" class="purple_btn" value="Додати користувача" @click="navigateTo('/add_user')">
      </div>
    </div>
    <div class="program_list">
      <div v-for="user in users" :key="user.id" class="user">
        <p><nuxt-link @click="updateUser(user)">{{ user.name }}</nuxt-link></p>
        <div class="user_btn">
          <input type="button" class="purple_btn_user" value="Змінити пароль" @click="changePassword(user)">
          <input v-if="user.is_active == true" type="button" class="purple_btn_user" value="Заблокувати" @click="block(user)">
          <input v-else type="button" class="purple_btn_user" value="Розблокувати" @click="unblock(user)">
        </div>
      </div>
    </div>
  </div>
</template>

<script setup lang="ts">

import { ref, onMounted } from '#imports';
import { storeToRefs } from 'pinia'
import { useAuthStore } from "~/store/auth";
import { useDataStore } from "~/store/data";

const dataStore = useDataStore();
let{ pass_data } = storeToRefs(dataStore);
const authStore = useAuthStore();
const { access_token }=  storeToRefs(authStore);

interface User {
  id: number;
  name: string;
  email: string;
  is_active: bool;
  updated_at: string;
};

interface ChangeIsActiveUser {
  id: number,
  is_active: bool
};

const users = ref([]);

onMounted(async () => {
  const config = useRuntimeConfig();
  const baseUrl = config.public.baseURL;
  pass_data = null;
  await $fetch(`${baseUrl}user`, {
      method: "GET",
      headers: {
        "Authorization": "Bearer "+access_token.value,
        "Content-Type": "application/json"
      }
    })
    .then(response => {
      users.value = response.data
    })
    .catch(error => {
      console.error('Помилка при отриманні даних:', error.message);
      if (error.status == 401)
        navigateTo('/login')
    });
 });

const changePassword = async (user) => {
  dataStore.setUser(user);
  navigateTo("/change_password");
};

const updateUser = async (user) => {
  dataStore.setUser(user);
   navigateTo("/update_user");
};

const block = async (user) => {
```

```
    let form: ChangeIsActiveUser = {
        id: user.id,
        is_active: false
    };

    await $fetch(`${baseUrl}user/change-password`, {
            method: "POST",
            body: form,
            headers: {
                "Authorization": "Bearer "+access_token.value,
                "Content-Type": "application/json"
            }
        })
        .then(response => {
            user.is_active = false;
        })
        .catch(error => {
            console.error('Помилка при отриманні даних:',
error.message);
            if (error.status == 401)
                navigateTo('/login')
        });
};

const unblock = async (user) => {
    let form: ChangeIsActiveUser = {
        id: user.id,
        is_active: true
    };
    await $fetch(`${baseUrl}user/change-password`, {
            method: "POST",
            body: form,
            headers: {
                "Authorization": "Bearer "+access_token.value,
                "Content-Type": "application/json"
            }
        })
        .then(response => {
            user.is_active = true;
        })
        .catch(error => {
            console.error('Помилка при отриманні даних:',
error.message);
            if (error.status == 401)
                navigateTo('/login')
        });
};

useHead({
    title: 'Новий користувач',
    meta: [
      { charset: 'utf-8' },
      { viewport: 'width=device-width, initial-scale=1' },
    ],
    link: [
      { rel: 'icon', type: 'image/x-icon', href: '/favicon.ico' },
    ],
    googleFonts: {
      preconnect: true,
      Inter: {
        wght: [100, 300, 400, 600, 700, 900],
        display: 'swap',
      },
    },
});



definePageMeta({
    middleware: [ 'auth', 'data' ],
    layout: 'default',
});
</script>
```

Фрагмент for_dash.vue

```
<template>
    <div class="info">
      <p v-for="(text, index) in texts" :key="index">{{ text
}}</p>
    </div>
</template>

<script>
export default {
    name: 'for_dash',
    props: {
      texts: Array,
    },
  };
</script>

<style scoped>
.info{
```

```
  display: flex;
  flex-direction: column;
  justify-content: center;
  gap: 20px;
}

.info p{
  font-size: 20px;
        font-style: normal;
        font-weight: 400;
        line-height: normal;
        margin: 0;
}

</style>
```

Фрагмент Menu.vue

```
<template>
  <div class="menu">
   <p>Меню</p>
   <div class="menu_list">
    <div class="menu_btn">
     <nuxt-link to="/program_list" class="menu_btn">
      <img src="/images/program.svg">
      <label for="btn_program">Програми</label>
     </nuxt-link>
     <input type="button" name="btn_program" id="btn_program" style="display:none;">
    </div>
    <div v-if="user && user.login=='admin'" class="menu_btn">
     <nuxt-link to="/user_list" class="menu_btn">
      <img src="/images/user.svg">
      <label for="btn_users">Користувачі</label>
     </nuxt-link>
     <input type="button" name="btn_users" id="btn_users" style="display:none;">
    </div>
    <div class="menu_btn">
     <nuxt-link to="/member_list" class="menu_btn">
      <img src="/images/people.svg">
      <label for="btn_participants">Учасники програми</label>
     </nuxt-link>
     <input type="button" name="btn_participants" id="btn_participants" style="display:none;">
    </div>
    <div class="menu_btn">
     <nuxt-link to="/request_list" class="menu_btn">
      <img src="/images/ask.svg">
      <label for="btn_appeals">Запити</label>
     </nuxt-link>
     <input type="button" name="btn_appeals" id="btn_appeals" style="display:none;">
    </div>
    <div class="menu_btn">
     <nuxt-link to="/cash_flow" class="menu_btn">
      <img src="/images/money.svg">
      <label for="btn_cash_flow">Книга руху коштів</label>
     </nuxt-link>
     <input type="button" name="btn_cash_flow" id="btn_cash_flow" style="display:none;">
    </div>
   </div>
  </div>
</template>

<script setup lang="ts">

import { useAuthStore } from "~/store/auth";
const authStore = useAuthStore();
const { user, access_token }=  storeToRefs(authStore);
console.log(user)

</script>
```