Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ЗАСТОСУНКУ ДЛЯ РОЗМІЩЕННЯ ТА ПЕРЕГЛЯДУ БЛАГОДІЙНИХ ЗБОРІВ З ІНДИВІДУАЛЬНИМИ ЦІЛЯМИ КОРИСТУВАЧІВ

Текстова частина до курсової роботи
за спеціальністю „ Комп'ютерні науки” 122

Керівник курсової роботи
асистент Калітовський Б. В.

_____
(підпис)
“____” _____ 2024 р.

Виконала студентка 3 курсу
Вальковець М. Ф.
(прізвище та ініціали)
“____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних
систем, доцент, кандидат наук
_____ Жежерун О.П.
(підпис)
„____” _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студентці Вальковець М. Ф. факультету інформатики 3 курсу
ТЕМА Розробка клієнт-серверного застосунку
Вихідні дані:

- Веб-застосунок для розміщення та перегляду благодійних зборів з індивідуальними цілями користувачів

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Анотація
Вступ
1. Аналіз предметної області
2. Теоретичні відомості
3. Опис реалізації веб-застосунку
Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2024 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	29.10.2023	
2.	Огляд технічної літератури за темою роботи.	30.11.2023	
3.	Аналіз актуальності та дослідження схожих рішень.	30.12.2023	
3.	Вибір стеку технологій для розробки	15.01.2024	
4.	Розробка застосунку	30.03.2024	
5.	Попередня демонстрація роботи керівнику	10.04.2024	
6.	Написання теоретичної частини	01.04.2024	
7.	Створення презентації	05.05.2024	
8.	Захист курсової роботи		

Студент _____

Керівник _____

“_____”_____

Зміст

Перелік використаних скорочень та термінів

Бан — один з прийнятих в Інтернеті способів контролю за діями користувачів. Як правило, бан полягає в обмеженні певних прав користувача.

RBAC(Role Based Access Control) - керування доступом на основі ролей.

HTTP(HyperText Transfer Protocol) - протокол, який є основою передачі даних для всесвітньої павутини , де гіпертекстові документи містять гіперпосилання на інші ресурси, до яких користувач може легко отримати доступ.

JSON(JavaScript Object Notation) - текстовий формат обміну даними.

JWT(JSON Web Token) - це спосіб передавати інформацію між сторонами у безпечному форматі, використовуючи структурований об'єкт JSON. Він складається з трьох частин: заголовка, даних та підпису.

DOM (Document Object Model) - це стандарт, що визначає структуру документу веб-сторінки, яка представлена у вигляді дерева об'єктів.

ER-модель(Entity-Relationship) - модель даних сутність-зв’язок, яка дозволяє описати схеми баз даних за допомогою блоків та зв’язків між ними.

Middleware - шар проміжного програмного забезпечення, який виконує певні визначені функції.

SMTP (Simple Mail Transfer Protocol) – протокол, який використовується для надсилання та отримання повідомлень електронної пошти через Інтернет.

TLS (Transport Layer Security) - це протокол шифрування, який забезпечує безпеку комунікації в мережі Інтернет.

Анотація

Курсова робота присвячена розробці клієнт-серверного застосунку для розміщення та перегляду благодійних зборів з індивідуальними цілями користувачів. Такий застосунок може бути актуальним для користувача з різних причин: бажання досягнути регулярності донатів, потреба розмістити десь власний благодійний збір коштів, можливість підтримувати збори інших людей, потреба мати в одному місці різні благодійні збори, які не потрібно шукати в соціальних мережах.

В цій роботі досліджено схожі рішення, проаналізовано актуальність питання, розглянуто принципи клієнт-серверної розробки. Проведено обгрунтований вибір стеку технологій : Node.js, React, MongoDB. Описано реалізацію та продемонстровано графічний інтерфейс розробленого застосунку.

Вступ

Через складні історичні події в Україні виникли волонтерські рухи. Головною причиною розвитку першого етапу волонтерства слугували події Революції Гідності грудня 2013 року. Після повномасштабного вторгнення РФ в Україну в лютому 2022 року розпочався другий етап, який відзначився новою силою та потужністю. [2]

Тож зрозуміло, для того щоб досягти мети благодійного збору організація чи волонтери повинні розповісти на певній платформі свої ініціативи, а зацікавлені люди повинні їх підтримати. З повномасштабним вторгненням одною з основних платформ для розміщення зборів стали соціальні мережі.

Разом з цим виник ряд проблем. Соціальні мережі мають свої правила, обмеження та можуть забанити дописи або ж сторінки, що унеможливлює поширення збору. Також виникає проблема, що користувача, який створив допис про збір, показує меншій кількості людей, ніж зазвичай, адже велика частина аудиторії незацікавлена та ніяк не реагує на такі дописи. Тож алгоритми соціальних мереж зменшують ймовірність того, що люди, які могли б підтримати певний збір, можуть його просто не побачити. [3]

З іншої сторони зараз збори закриваються важче і довше, ніж наприклад на початку повномасштабного вторгнення. В цьому є багато причин : виникнення недовіри до волонтерів через прикрі випадки, погіршення фінансового становища людей, політичні та корупційні скандали, погані настрої в суспільстві тощо. Також можна побачити, що швидше збори закриваються, коли стається якась емоційна подія, хоча насправді важливо, щоб донати були регулярні, а не лише через певні обставини.

Можливим вирішенням таких проблем є застосунок, де можна додавати свої збори, які будуть переглядати люди, що зацікавлені підтримати їх. Для користувачів соціальних мереж, котрі не мають великої аудиторії, такий

застосунок може збільшити кількість благодійних внесків. Для досягнення регулярності донатів можна ставити собі цілі на певні періоди часу.

Метою є створення веб-застосунку, для розміщення та перегляду благодійних зборів з індивідуальними цілями користувачів.

Завданням є аналіз питання, дослідження схожих рішень, розробка веб-застосунку, що матиме інтуїтивно зрозумілий інтерфейс та вирішує проблеми користувача.

Робота складається з трьох розділів.

Перший розділ присвячено аналізу предметної області, схожих рішень, дослідженню актуальності теми.
В другому розділі проаналізовано основні теоретичні відомості, які потрібні в подальшому для реалізації застосунку.

В третьому розділі описано реалізацію веб-застосунку та продемонстровано користувацький інтерфейс.

Розділ 1. Аналіз предметної області

1.1 Аналіз сучасного стану питання та обґрунтування теми

Кожен свідомий громадянин України розуміє, що треба допомагати нашому війську, постраждалим від війни. На сьогоднішній день в нас є багато волонтерів, благодійних організацій, які постійно створюють нові збори на різні потреби. Також багато хто має знайомих, друзів, родичів у війську, які мають різні потреби для власної безпеки, ефективного виконання бойових завдань тощо.

Отже, постійно створюються нові благодійні збори. Коли хтось відкриває приватний збір, то додає до нього реквізити для пожертв. Це або картки банків, або наразі популярний інструмент – банка в Монобанку.

За статистикою, яку надав Монобанк 30 грудня 2023 року, на банки фондів та фізичних осіб за 2022 рік було задоначено - 8,5 мільярда гривень, за 2023 рік - 27,4 мільярда гривень (Додаток А). Також на 2,5 мільйони зросла кількість донорів порівняно з 2023 роком. [4]

Очевидно, щоб зібрати необхідну суму потрібно залучити якнайбільше людей. Тож кожен організатор збору ділиться ним в соціальних мережах, залишає QR-коди в кафе чи магазинах. Не кожен має велику аудиторію в соціальних мережах, а ті хто її має теж стикаються з рядом проблем. Блогери, інфлуенсери можуть отримати бан за порушення правил спільноти, видалення їхнього допису про збір, зниження охоплень, що зменшує кількість людей, які побачать ініціативу та задонатять.

Зборів стає постійно все більше, потреб все більше, тому вони дуже тяжко і довго закриваються, деякі навіть – ні. Однією з можливих причин цього є нерегулярність донатів. Важливо робити донати постійно аби це ставало звичкою, навіть якщо це невеликі суми. Адже якщо велика кількість людей закине по 20 грн, то певна частина збору може закритися. Таким чином і

збори зможуть закриватися швидше, бо часто це нагальні потреби від яких залежить життя.

1.2 Опитування щодо використання застосунку

З метою розуміння чи буде застосунок користуватися попитом та чи має він потенційних користувачів було нами проведено опитування серед студентів. Воно було створене у вигляді Google форми та складалося з чотирьох питань:

1) Чи регулярно ви донатите?
2) Чи виділяєте певну суму для донатів щомісяця(або інший проміжок часу)?
3) Чи користувалися б ви платформою, де можна було б ставити собі за ціль певну суму донатів та моніторити її?
4) Чи користувалися б ви платформою, де можна додавати свої власні збори, дивитися збори інших?

Питання були спрямовані на визначення регулярності донатів у студентів, можливого попиту на застосунок. Опитування показало, що майже 60 відсотків користувалося б застосунком, де можна було б ставити собі за ціль певну суму донатів та моніторити її, а майже 65 відсотків - користувалися б ви платформою, де можна додавати свої власні збори, дивитися збори інших. Результати опитувань зображено на рисунку 1.2 та 1.3.

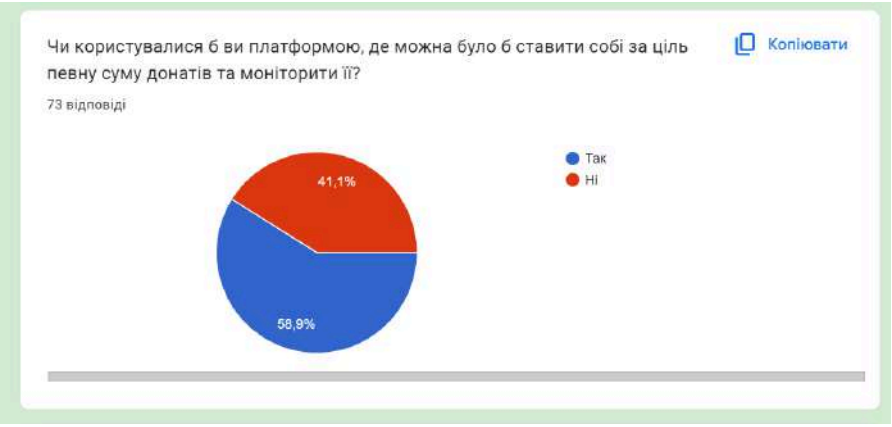

Рисунок 1.2 – Результати опитування

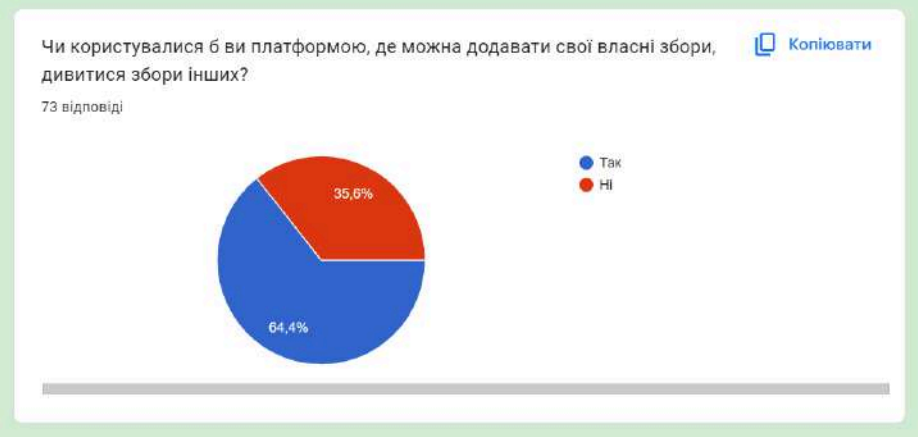

Рисунок 1.3 – Результати опитування

Дослідження проблеми показало, що веб-застосунок, де можна розміщувати власні збори, переглядати збори інших, ставити особисту ціль по донатах на певний проміжок часу або ж періодичну ціль є актуальним.

1.3 Аналіз подібних рішень

Щоб дослідити подібні рішення було обрано сайти благодійного фонду “Повернись живим”, всеукраїнського гуманістичного руху UAnimals. Вони мають такі спільні риси, як оформлення підписки на фонд, перегляд актуальних та архівних проєктів. Також було обрано застосунок Donate24, в якому можна дивитися актуальні збори, зібрані з соціальних мереж.

1.3.1 Сайт благодійного фонду “Повернись живим”

На рисунку 1.4 зображено частину з головної сторінки сайту “Повернись живим”, де показані актуальні проєкти та є можливість їх підтримати.

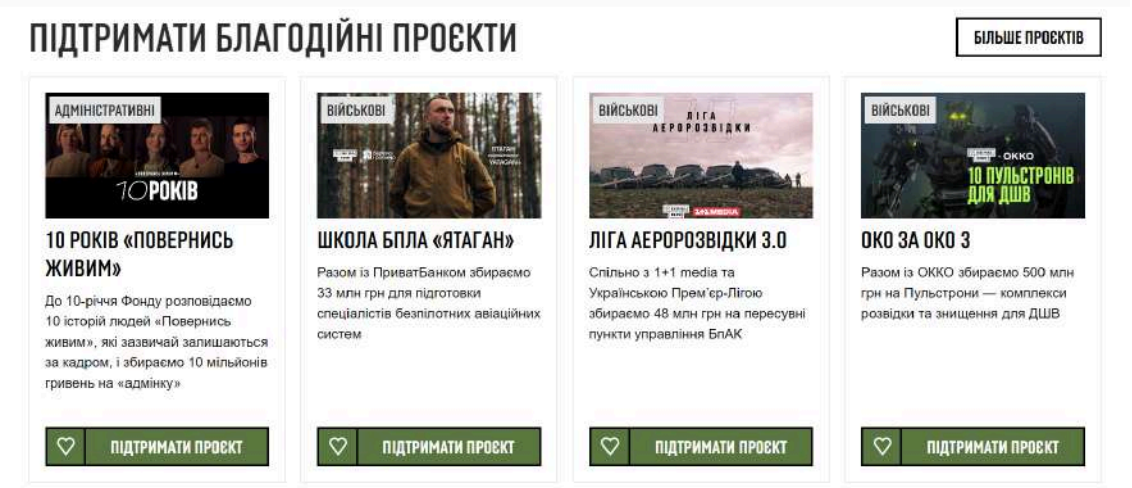

Рисунок 1.4 – Частина сайту благодійного фонду «Повернись живим»

На рисунку 1.5 можна побачити інтерфейс оформлення підписки на фонд з вибором способу переказу, валютою та сумою платежу.

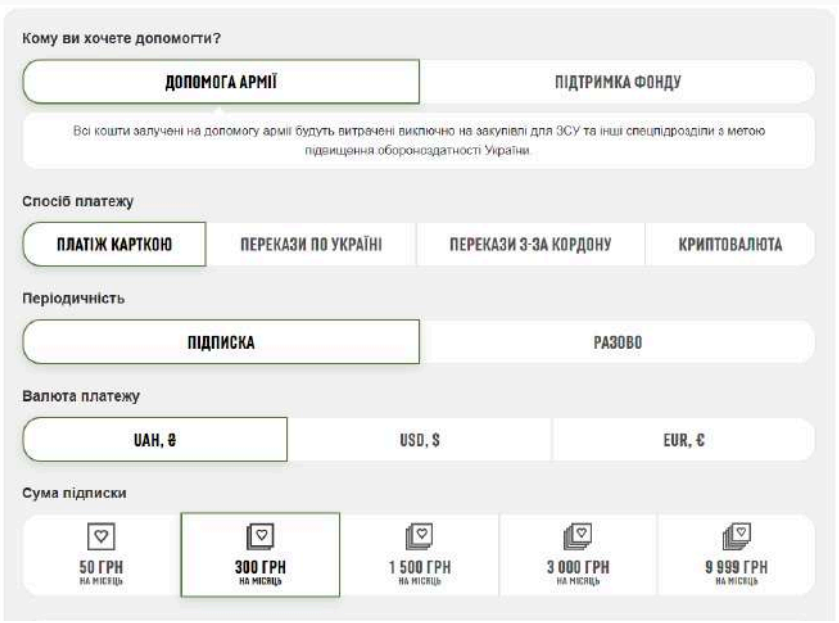

Рисунок 1.5 – Оформлення підписки на фонд

На сайті “Повернись живим” користувач має можливість :

- подивитися всі проєкти фонду та детальну інформацію про них;
- подивитися звітність фонду;
- подивитися скільки коштів зібрано;
- подивитися мету збору;

- зробити запит на допомогу.

Власних зборів розміщувати не можна, адже це власний сайт фонду.

1.3.2 Сайт всеукраїнського гуманістичного руху UAnimals

На цьому сайті, можна запустити свій збір, а тобто подію, яка слугуватиме як привід для збору коштів для постраждалих тварин. Це означає, що користувач має поширювати посилання на свою згенеровану веб-сторінку з описом події, метою, де інші люди зможуть зробити пожертву.

На рисунку 1.7 зображено головна сторінка, на якій можна ознайомитися детальніше з ініціативою “Святкуйте по-особливому разом із UAnimals”. На цій сторінці можна перейти за кнопкою “святкуємо разом” та заповнити форму для додавання своєї події.

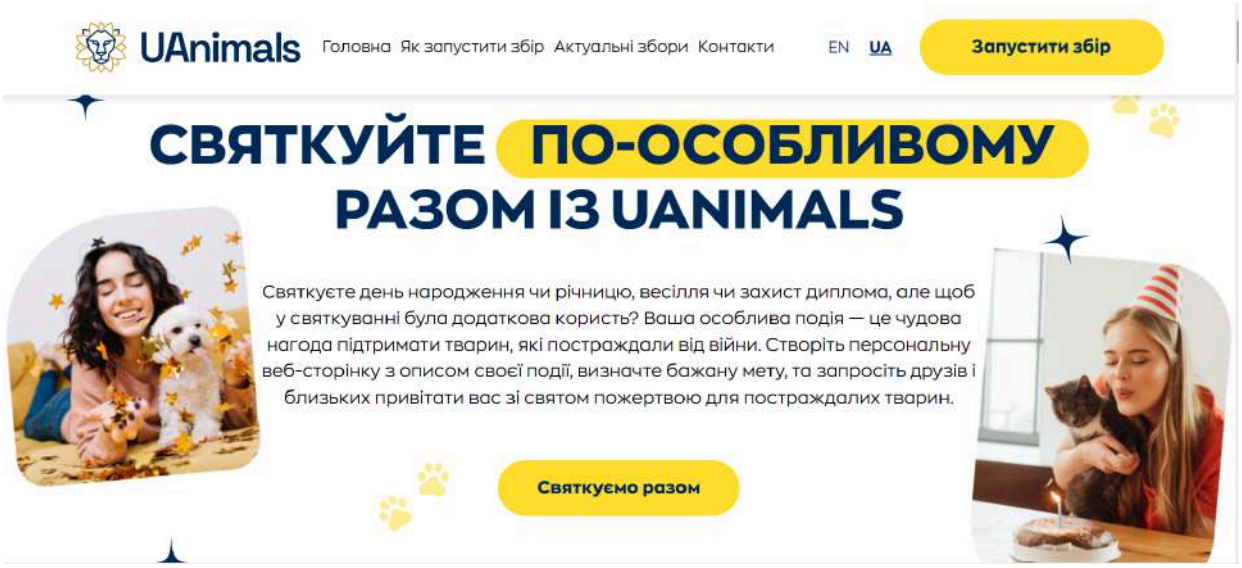

Рисунок 1.7 – Головна сторінка, де можна запустити свій збір

Форма містить різні персональні питання та опис збору, після заповнення яких користувач надсилає її на затвердження. На рисунку 1.8 можна побачити частину цієї форми.

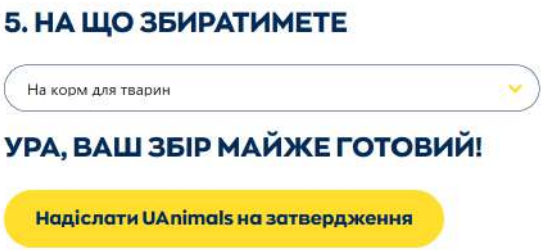

Рисунок 1.8 – Частина форми

Тобто, на сайті UAnimals користувач може :

- подивитися активні збори;
- подивитися звітність;
- створити подію, святкуючи яку збирати кошти для постраждалих тварин;
- заповнити форму(обрати подію чи свято, розказати про себе, завантажити фото, коротко описати свій збір та обрати на який саме напрямок UAnimals підуть зібрані кошти);
- надіслати на затвердження заповнені дані.

Отож свій власний збір додати ми не можемо. Можемо лише створити певну подію, з якою ділитися та збирати таким чином кошти до UAnimals.

1.3.3 Застосунок для збору донатів Donate24

Застосунок Donate24 створений українськими розробниками має на меті показувати збори зібрані в соціальних мережах.

В цьому мобільному веб-застосунку користувач має можливість :

- переглядати збори;
- копіювати реквізити;
- ділитися посиланням на збір;
- дивитися скільки набрано та ціль збору.

На рисунку 1.9 можна познайомится з інтерфейсом цього застосунку.

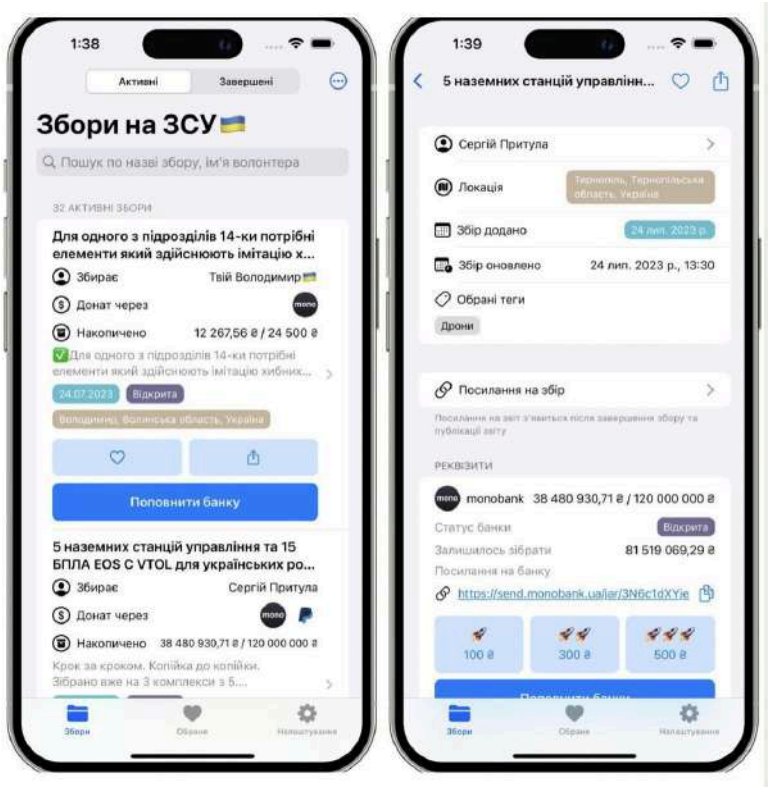

Рисунок 1.9 – Інтерфейс застосунку Donate24

Недолік застосунку в тому, щоб додати власний збір, треба писати напряму розробнику. Також цей застосунок доступний лише для iOS, що значно обмежує кількість користувачів, які зможуть його завантажити та користуватися.

1.4 Постановка завдання

Після аналізу предметної області, дослідження схожих рішень, а також вивчення актуальності теми було сформовано наступне завдання, та вимоги які необхідно виконати.

Потрібно розробити застосунок, який відповідатиме UI-UX стандартам, тобто матиме інтерфейс, що буде зручним у використанні. Користувачем такого застосунку буде людина, яка має потребу розмістити свій збір або має зацікавленість підтримати інші збори.

Тобто користувач повинен мати змогу зареєструватися, увійти, та використовувати свій профіль для перегляду інформації про себе, додавати збори та отримувати підтвердження або відмову в його розміщенні, дивитися свої збори та їх прогрес, задавати цілі донатів, змінювати їх значення, додавати донати та позначати збори як закриті. Також може переглядати всі збори, їхні прогреси, переходити на профілі організатора та робити донати через монобанк.

Для того, щоб забезпечити графічний функціонал для адміністрування веб-застосунку окремо виділено роль адміністратора. Адміністратор має доступ до всіх функцій користувача, а також може переглядати непідтверджені збори та підтверджувати їх, переглядати список потенційних верифікованих користувачів та позначати їх як верифікованих. Також може скасувати підтвердження збору.

1.5 Висновки до розділу 1

Отже, було проведено аналіз сучасного стану питання та обгрунтовано тему курсової роботи, наведено мету та результати проведеного опитування. Був проведений аналіз таких подібних рішень, як сайт благодійного фонду “Повернись живим”, сайт всеукраїнського гуманістичного руху UAnimals та застосунок Donate24. Описано можливості користувача на вищезгаданих ресурсах та згадано деякі недоліки. Враховуючи результати досліджень було сформовано завдання.

Розділ 2. Теоретичні відомості

2.1 Функціональні вимоги до застосунку

Планується розробити веб-застосунок для розміщення благодійних зборів, перегляду інформації про інші, можливістю перейти за посиланням та зробити донат, створити особисту ціль. Цільовою аудиторією є люди, які відкривають та поширюють власні збори з метою їхнього закриття. Також люди, які хочуть мати додаткову мотивацію для регулярних донатів, матимуть змогу ставити собі періодичні цілі.

Ролі користувачів розділено на користувача та адміністратора. Користувачі розділяються на верифіковані та не верифіковані. Верифіковані - це ті, які мали більше 5 підтверджених зборів та були верифіковані адміністратором. Інтерфейс веб-застосунку складається зі сталого верхнього меню, у якому є такі розділи: збори, додати збір, мій профіль, вийти.

- Збори

На сторінці, де розміщені всі збори та така інформація для кожного, як категорія збору, його мета, для кого цей збір, організатор збору, скільки вже набрано коштів і скільки потрібно та прогрес-бар який показує відсоток зібраної суми до всієї. Після натискання на організатора збору користувач може побачити його активні та завершені збори. Користувач може ввести суму донату, з якою після натискання кнопки “зробити донат” він буде перенаправлений за посиланням на банку збору. Після повернення до сайту користувачеві буде запропоновано додати суму донату, яку він вводив, до своєї поточної суми.

- Додати збір

На цій сторінці користувач може заповнити форму для додавання свого збору. Для цього він повинен вказати інформацію у всі обов’язкові поля та надіслати збір для перевірки адміністратором.

- Мій профіль

На сторінці свого профілю користувач може дивитися всі свої збори, дивитися статус їхнього підтвердження, позначати збір, як завершений. Також в своєму профілі користувач може дивитися, яка в його ціль на поточний момент та змінювати її. Можна змінювати суму цілі та тип цілі: на тиждень, на місяць, до певної дати. На цій сторінці є можливість додати свій донат до поточної суми цілі для її відстеження. Користувачу також показується історія його цілей та інформація чи були вони виконані.

- Вийти

Після натискання цієї кнопки користувач завершує свою сесію.

В адміністратора верхнє меню має додатковий розділ:

- панель адміністратора

На цій сторінці адміністратор може переглянути всі непідтверджені збори, натиснути кнопку підтвердити - зробити видимим збір для всіх користувачів, а також кнопку видалити. Цим організатор збору отримає лист на свою email-адресу та буде проінформований про статус розгляду його збору. Адміністратору показуються потенційні верифіковані користувачі, верифікованість яких він може підтвердити.

Також адміністратор на сторінці збори може скасувати підтвердження збору, якщо для цього є потреба.

Перед тим як розпочати реалізацію програмного продукту, треба визначити та описати його усі можливі функції. Отож розділимо функціональні вимоги залежно від ролі.

2.2 Забезпечення безпеки доступу до інформації та операцій

Під час розробки застосунку важливо створити систему керування доступом до інформації.

Є такий підхід як, Attribute-Based Access Control(ABAC) – модель керування доступом на основі атрибутів користувачів, ресурсів, дії та середовища. Тобто рішення про надання доступу базується на визначених характеристиках усіх об’єктів, які беруть участь в процесі. Наприклад, користувач може мати доступ до певного файлу, якщо файл повинен має визначену характеристику і сам користувач теж має певну характеристику. Тобто, за такого керування можна достатньо гнучко визначити доступ.

Також одним із можливих є керування доступом на основі ролей - Role-based access control (RBAC). Для того щоб використовувати такий контроль, треба проаналізувати функціональні потреби користувачів і згрупувати їх у ролі відповідно до потреб. Тобто роль міститиме набір дозволів. Потім призначити одну або декілька ролей користувачеві.

Для розробки клієнт-серверного застосунку було обрано керування доступом на основі ролей(RBAC), адже було важливо відділити функціонал адміністратора та користувача та не потрібно визначати додаткові атрибути для різних об’єктів.

2.3 Авторизація та автентифікація

Щоб створити застосунок, де будуть різні ролі, а також контроль доступу на основі ролей потрібно розуміти такі поняття як авторизація та автентифікація.

Автентифікація – це процес підтвердження того, що користувачі є тими, ким вони себе видають. Це перший основний крок у процесі безпеки. Наприклад, перевірка чи збігаються введений пароль з тим, що в базі даних.

Авторизація – це процес надання користувачеві дозволу на доступ до певних ресурсів або функцій. Що забезпечує контроль доступу.

2.4 Вибір технологій розробки

Для того щоб, розробити веб-застосунок з зазначеними вище вимогами потрібно правильно підібрати технології для розробки.

2.4.1 Вибір технологій для серверної частини

Для реалізації серверної частини було обрано Node.js, адже його можливості забезпечують швидку та ефективну розробку за допомогою JavaScript, що спрощує розробку та підтримку коду. Також плюсом є його потужний пакетний менеджер npm за допомогою якого можна швидко встановлювати та оновлювати різні пакети та модулі. Розробляючи на Node.js можна користуватися інструментом nodemon, який автоматично перезапускає сервер, якщо є зміни в коді, що значно збільшує продуктивність розробки.

Express.js – це зручний фреймворк для веб-застосунків, побудованих на Node.js. Він був обраний, тому що надає інструменти для створення маршрутів, обробки HTTP-запитів та відправки відповідей, тобто ідеально підходить для реалізації API.

Для забезпечення безпеки паролів одним з варіантів є їх хешування, тож було вибрано інструмент bcrypt, який дозволяє нам зберігати паролі в базі даних у формі хешу. Плюсом також можна зазначити можливість задавати параметри, які додатково зміцнюють хешування.

JWT(JSON Web Tokens) використовується в застосунку для забезпечення безпеки та автентифікації користувачів, а також для реалізації керування доступу за ролями(RBAC). JWT дозволяє стисло представити інформацію про конкретного користувача за допомогою токена, які можна перевіряти та розшифровувати. Таким чином ми можемо безпечно передавати дані між клієнтом та сервером.

2.4.2 Вибір технологій для клієнтської частини

Для розробки клієнтської частини застосунку було обрано React. Це бібліотека JavaScript для створення інтерфейсів користувача, яка дозволяє створювати компоненти, які можна використовувати повторно на різних сторінках. Його перевагою є те, що він використовує віртуальний DOM(Document Object Model). Коли до користувацького інтерфейсу додаються нові елементи, то створюється нове віртуальне дерево віртуального DOM, потім воно порівнюється з попереднім. Після цього обчислюється найоптимальніший метод внесення змін у реальний DOM, що гарантує, що з реальним DOM буде виконано мінімум операцій. Такий алгоритм забезпечує, що складні сторінки будуть мати невеликий час рендерингу, адже все реальне DOM-дерево не мусить оновлюватися щоразу після змін. Це є перевагою React над Angular, який використовує реальний DOM.

Для зручного та сучасного дизайну користувацького інтерфейсу було прийнято рішення обрати Material-UI. Це безкоштовна бібліотека компонентів React з відкритим кодом, яка реалізує Material Design від Google(це принципи дизайну сайтів, програмного забезпечення і застосунків). Material-UI пропонує велику кількість вже готових компонентів та інструментів, це в свою чергу, дозволяє ефективно і швидко створити стильний інтерфейс. У порівнянні з іншими бібліотеками компонентів, такими як Bootstrap чи Ant Design, Material-UI дозволяє створити сучасний та актуальний дизайн сайту.

2.4.3 Зв’язок між серверною та клієнтською частиною

Взаємодія між клієнтом та сервером буде реалізована за допомогою REST API. Уся комунікація в такому випадку відбувається через HTTP-запити. Запит від клієнта до сервера надсилається у форматі веб-URL, як запит HTTP:

- GET – отримання даних з сервера;
- POST – надсилання даних на обробку до сервера;

- PUT – надсилання даних на сервер для створення або оновлення ресурсу;
- DELETE – видалення вказаного об’єкта;

Після цього відповідь буде повертатися у форматі JSON. Цю логіку можна побачити на рисунку 2.1.

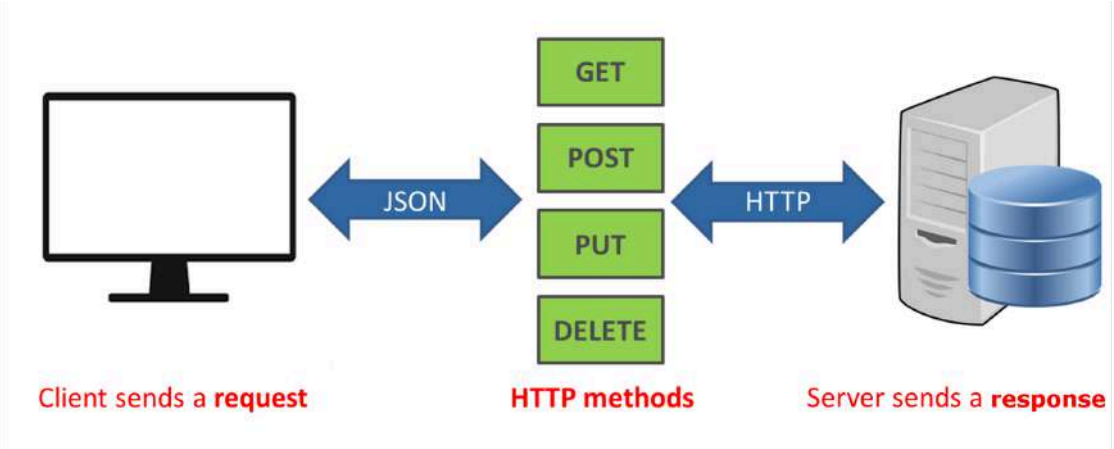

Рисунок 2.1 - взаємодія між клієнтом та сервером

З метою забезпечення зручного та ефективного способу взаємодії клієнта з сервером через HTTP-запити потрібно обрати спосіб за допомогою якого ці запити будуть надсилатися. Є багато таких способів в JavaScript, але два з найпопулярніших — це Axios і fetch().

Fetch() — це вбудований API JavaScript. Це асинхронний веб-API, який повертає дані у формі обіцянок. Axios - це широко поширена бібліотека JavaScript, також підтримує обіцянки, а також має ряд переваг. Вона автоматично перетворює відповіді у формат JSON, має більш зручний та простий синтаксис, має більшу спільноту розробників.

Вибір Axios для надсилання HTTP-запитів є оптимальним рішенням, тому саме вона буде використовуватися при реалізації застосунку.

2.4.4 Вибір бази даних

MongoDB може бути зручною для розробки клієнт-серверних веб-застосунків, тому саме її було вибрано. Одною з основних причин

зручності MongoDB є її гнучкість, тобто за потреби під час розробки можна швидко змінити структуру даних. Також варто додати, що обрана база даних спрощує взаємодію з сервером, дозволяє робити швидкі запити, бо використовує документи у форматі JSON.

Для взаємодії сервера з базою даних використовується Mongoose – бібліотека для Node.js, яка надає зручний об’єктно орієнтований підхід. За допомогою Mongoose можна визначати моделі даних у форматі JavaScript класів або схем.

2.5 Односторінкова програма (SPA) з використанням стеку MERN

Обрані технології розробки для веб-застосунку - це стек MERN, який означає MongoDB для бази даних, Express.js та Node.js для створення серверної частини, а також React для клієнтської частини. Застосунок буде односторінковим, що означає, що він буде перезавантажувати лише певні частини сторінок за допомогою JavaScript, без перезавантаження сторінки повністю.

Односторінкові програми мають основні переваги : швидкість та безперебійність в роботі, зручний роутинг та менше навантаження на сервер, адже під час взаємодії користувача із сервера завантажуються лише необхідні ресурси.

2.6 Висновки до розділу 2

Отож, підсумовуючи вищенаведену інформацію у другому розділі, було обґрунтовано вибір контролю доступу на основі ролей, пояснено поняття авторизації та автентифікації. Було визначено, що застосунок повинен мати дві ролі: користувач та адміністратор. Також було обрано технології для розробки такі, як MongoDB, React, Node.js, Axios та пояснено такий вибір. Було зроблено вибір реалізувати взаємодію між клієнтом та сервером за допомогою REST API через HTTP-запити .Було вирішено розробляти веб-застосунок як односторінковий сайт.

Розділ 3. Опис реалізації веб-застосунку

3.1 ER-модель

Першим етапом розробки стала побудова ER-моделі, для того щоб зрозуміти структуру даних, які будуть використовуватися та мати загальне уявлення про базу даних перед реалізацією.

На рисунку 3.1 зображена побудована ER-модель.

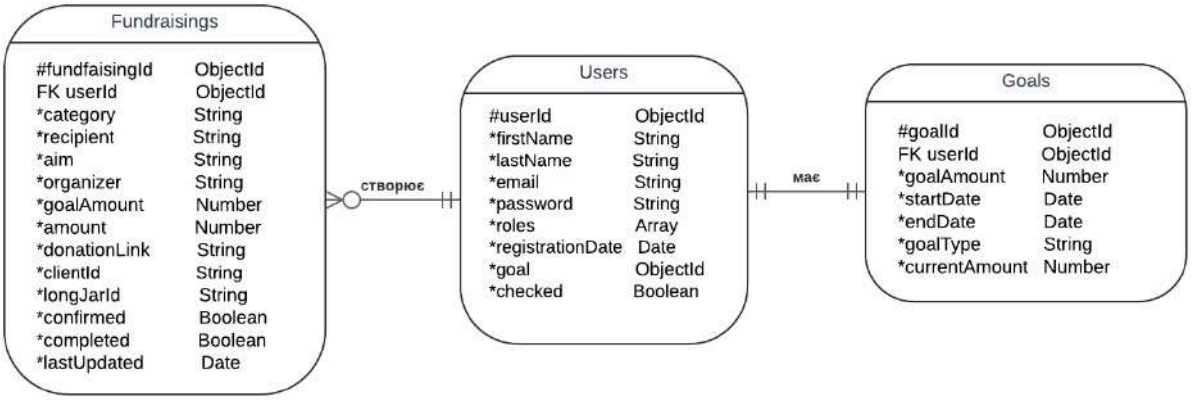

Рисунок 3.1 - ER-модель

Створено три сутності fundraisings(збори), users(користувачі), goals(цілі). Між зборами та користувачами зв’язок один до багатьох, адже кожен збір має одного користувача-організатора, а користувач може створювати багато зборів. Зв’язок між цілями та користувачами один до одного, адже один користувач може мати одну ціль і одна ціль може належати одному користувачеві.

Сутність fundraising має такі атрибути :

- fundraisingId: Це унікальний ідентифікатор (objectId), що ідентифікує кожен конткретний збір коштів.
- userId: Ідентифікатор користувача (FK), який посилається на користувача, який створив збір.
- category: Категорія, до якої відноситься акція збору коштів. Може бути гуманітарна або для армії.
- recipient: Для кого проводиться збір, наприклад назва бригади.
- aim: Мета або призначення збору коштів, тобто на що саме збирається.
- organizer: Ім’я організатора, який додав цей збір.
- goalAmount: Сума коштів, яку необхідно зібрати.
- amount: Поточна сума коштів, яку вдалося зібрати до моменту оновлення запису.
- donationLink: Посилання на сторінку, де можна здійснити пожертвування.
- clientId: Ідентифікатор банки Монобанку.
- longJarId: Ідентифікатор банки Монобанку, який використовується для оновленння інформації .
- confirmed: Позначає чи підтверджено збір коштів.
- completed: Позначає, чи завершений збір коштів.
- lastUpdated: Дата та час останнього оновлення запису.

Сутність users має такі атрибути:

- userId: Унікальний ідентифікатор користувача.
- firstName: Ім'я користувача.
- lastName: Прізвище користувача.
- email: Адреса електронної пошти користувача.
- password: Пароль користувача (зашифрований).
- roles: Масив ролей, які має користувач у системі (адміністратор, звичайний користувач).

- registrationDate: Дата реєстрації користувача в системі.
- goal: Ідентифікатор (FK) його цілі, пов'язаної з користувачем.
- checked: Вказує, чи верифікований профіль користувача.

Сутність goals має такі атрибути:

- goalId: Унікальний ідентифікатор цілі.
- userId: Ідентифікатор користувача (FK), до якого відноситься ця ціль.
- goalAmount: Сума коштів, яку необхідно зібрати для досягнення цієї цілі.
- startDate: Дата початку періоду, протягом якого користувач планує досягнути цілі.
- endDate: Дата закінчення періоду, протягом якого користувач планує досягнути цілі.
- goalType: Тип цілі (на тиждень, на рік, до певної дати).
- currentAmount: Поточна сума коштів, яку вдалося зібрати для цієї цілі.

3.2 Реалізація серверної частини застосунку

3.2.1 Структура проєкту

Перш за все треба сформувати правильну структуру папок проєкту, щоб всі різні функції були розділені між собою. Це необхідно для полегшеного розуміння коду та організації проєкту. Правильна структура надає можливість додавати новий функціонал без змін, перейменувань створених частин. Відповідно до того, що різні частини програми відокремлені одна від одної, зменшується залежність між ними.

Нижче на рисунку 3.1 зображено структуру мого проєкту

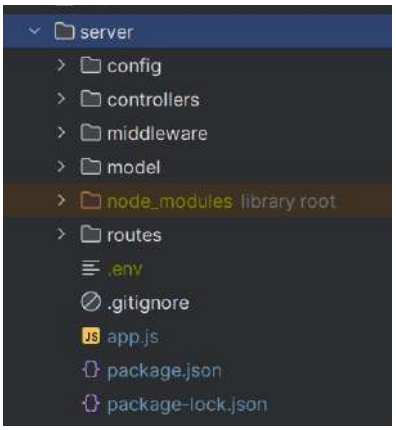

Рисунок 3.1 – структура проєкту

Для зручності було розділено функціонал за папками. В папці контролери розміщені файли, в яких зосереджена вся основна логіка. Для кожної сутності : admin, goal, user, fundraising є свої контролери. В теці config визначено файл, де відбувається підключення до бази даних, в middleware – проміжне програмне забезпечення, в model – визначені схеми для сутностей, в route - всі маршрути.

3.2.2 Визначення схем сутностей та підключення до бази даних

Наступним кроком є визначення схем для кожної сутності бази даних. Для цього використовується бібліотека Mongoose, про яку вище було сказано. Щоб визначити схему потрібно описати всі атрибути та їхні типи даних, потім треба створити модель сутності, яка базується на цій схемі. Модель – це клас, який в подальшому буде використовуватися в контролерах для взаємодії безпосередньо з базою даних. Нижче на рисунку 3.2 можна побачити приклад визначення схеми для користувача.

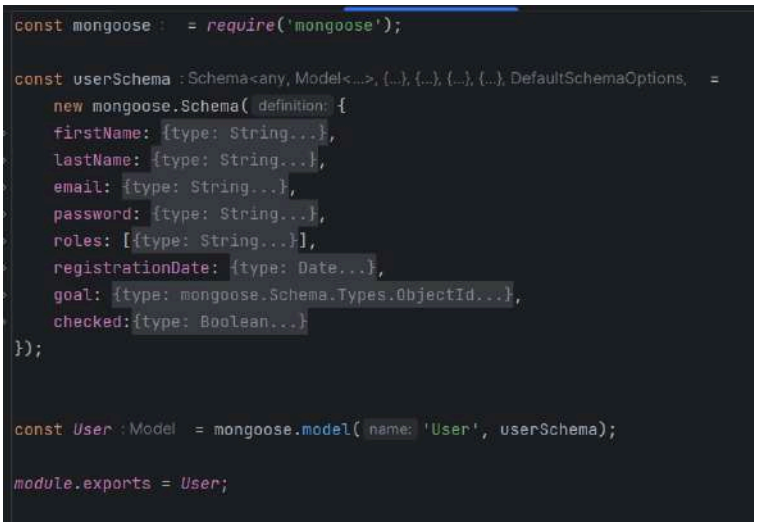

Рисунок 3.2 – Приклад визначення моделі користувача

Підключення до бази даних відбувається також за допомогою бібліотеки Mongoose використовуючи метод connect().

3.2.3 Контролери та зв’язок з моделлю

Для кожної сутності контролери реалізовані в окремих файлах.

Нижче на прикладі можна побачити реалізацію одної з функцій-контролерів для адміністратора. Ця функція приймає об'єкти req (запит) та res (відповідь). Всередині виконується виклик імпортованої моделі ‘Fundraising’ з методом ‘find’, який здійснює запит до бази даних MongoDB, щоб знайти всі збори, які ще не підтверджені (тобто ті, у яких поле confirmed має значення false). Після отримання результатів запиту відправляється відповідь у JSON форматі, в цьому випадку повертаються всі непідтверджені збори а також статус 200(запит успішно виконаний). Якщо ж виникає помилка, то відправляється статус відповіді 500 та JSON з повідомленням про помилку.

Рисунок 3.3 – Приклад контролера для отримання всіх непідтверджених зборів

В подальшому ці контролери використовуються у визначенні маршрутів. На рисунку 3.4 можна побачити приклад визначення для адміністратора.

```
router.get( path: '/admin', auth.protect, auth.checkAdmin, adminController.getAdminPage);

router.get( path: '/allUsers', auth.protect, auth.checkAdmin, adminController.getAllUsers);

router.get( path: '/unconfirmedFundraising',auth.protect, auth.checkAdmin, adminController.getUnconfirmedFundraising);

router.post( path: '/confirmFundraising/:fundraisingId', auth.protect, auth.checkAdmin, adminController.confirmFundraising);

router.delete( path: '/deleteFundraising/:fundraisingId',auth.protect, auth.checkAdmin, adminController.deleteFundraising);

router.post( path: '/cancelConfirmFundraising/:fundraisingId', auth.protect, auth.checkAdmin, adminController.cancelConfirmFundraising);

router.get( path: '/potentialVerifiedUsers',auth.protect, auth.checkAdmin, adminController.getPotentialVerifiedUsers);

router.put( path: '/verifyUser/:userId', auth.protect, auth.checkAdmin, adminController.verifyUser);

router.post( path: '/sendEmail', auth.protect, auth.checkAdmin, adminController.sendEmail);

module.exports = router;
```

Рисунок 3. 4 – визначення маршрутів для адміністратора

В головному файлі app.js сервер Express та підключаються всі ці маршрути, які визначені у файлах. На рисунку 3.5 показано як це реалізовано.

```
const userRoutes : Router | {...} = require('./routes/userRoutes');
const adminRoutes : Router | {...} = require('./routes/adminRoutes');
const fundraisingRoutes : Router | {...} = require('./routes/fundraisingRoutes');
const goalRoutes : Router | {...} = require('./routes/goalRoutes');

app.use('/api/users', userRoutes);
app.use('/api/admin', adminRoutes);
app.use('/api/fundraising',fundraisingRoutes );
app.use('/api/goal', goalRoutes);
```

Рисунок 3.5 – підключення та визначення маршрутів

3.2.4 Реалізація механізму реєстрації, авторизації та автентифікації

Для того щоб реалізувати механізм доступу до інформації за допомогою ролей потрібно, щоб користувачі мали можливість авторизуватися на сайті.

Перш за все має бути логіка реєстрації користувача, коли визначаться його всі його обов’язкові атрибути та збережуться до бази даних разом із захешованим паролем. Основні кроки реєстрації користувача :

1) Отримання даних з запиту
2) Хешування паролю
3) Створення користувача в базі даних
4) Створення та призначення цілі користувача
5) Генерація JWT-токена (Нижче на рисунку 3.6 зображено цей процес)
6) Надсилання відповіді клієнту

Рисунок 3.6 – Генерація токена для користувача

Наступним кроком є реалізація автентифікації. Основні етапи цього процесу:

1) Отримання даних з запиту
2) Пошук користувача в базі даних
3) Перевірка чи наявний користувач в базі даних
4) Перевірка пароля за допомогою функції bcrypt.compare(). Нижче на рисунку 3.7 це показано.
5) Генерація JWT-токена, який в подальшому буде використовуватися для авторизації користувача під час запитів
6) Відправка відповіді клієнту

Рисунок 3.7 – перевірка введеного пароля

Наступний крок – реалізація авторизації. Для цього було створено middleware функція protect, яка перевіряє чи JWT-токен, який передався від клієнта коректний та чи дійсний він; checkAdmin, яка перевіряє чи користувач, який передав запит має роль адміністратора.

Основні кроки функції protect:

1) Перевіряється чи наявний токен у заголовку запиту, розбивається на дві частини : префікс та токен.
2) Якщо токен не існує або він неправильний, то повертається статус 401.
3) Після виявлення коректного токену його вміст розшифровується та береться ідентифікатор користувача.
4) Перевіряється чи такий користувач існує в базі даних.
5) Перевіряється чи токен не застарів, якщо застарів, то повертається статус 401.
6) Якщо всі перевірки пройшли успішно, то користувач додається до об'єкта запиту (req) як req.user, а потім за допомогою функції next() викликається або наступний middleware, або обробляється запит.

Основні кроки функції checkAdmin:

1) Перевіряється чи користувач є у полі req.user.
2) Перевіряється чи користувач має роль адміністратора (значення в базі даних 1), якщо так то викликається обробник запиту.
3) Якщо користувач не має ролі адміністратора, то повертається статус 403.

Нижче на рисунку 3.8 наведено реалізацію функції checkAdmin.

```
exports.checkAdmin = async (req, res, next) : Promise<void>  => {
    if (req.user && req.user.roles && req.user.roles.includes('1')) {
        console.log("Зайшов адмін")
        next();
    } else {
        res.status(403).json({ message: 'Access denied' });
    }
}
```

Рисунок 3.8 – реалізація функції checkAdmin

Описані вище функції використовуються в подальшому використовуються у визначенні маршрутів.

3.2.5 Реалізація автоматичного збору інформації з банки Монобанку

Важливим кроком є реалізація автоматичного витягування інформації про поточний стан банки. Для цього треба мати два атрибути clientId – ідентифікатор банки та longJarId – ідентифікатор банки, який використовується в запиті до API Монобанку. Коли адміністратор підтверджує збір, то надсилається запит на сервер, який вилучає clientId з посилання на банку. Далі відбувається POST-запит до API монобанку для отримання longJarId, в якому є параметри clientId, pc-будь-яке значення, c-“hello”. Нижче на рисунку 3.9 наведено запит, який отримує longJarId.

```
const response : AxiosResponse<any>  = await axios.post( url: 'https://send.monobank.ua/api/handler',
    data: {
    c: "hello",
    clientId: clientId,
    Pc: "pcpcpcpcpcpcpcpcpcpc"
});
```

Рисунок 3.9 – Отримання longJarId

Значення знайдених двох ідентифікаторів зберігаються в базі даних.

Було визначено функцію getCurrentInfo, яка отримує поточну інформацію про заданий збір та зберігає зміни до бази даних. Для того щоб отримати поточну інформацію про банку треба виконати GET-запит до

‘https://api.monobank.ua/bank/jar/:longJarId’. Цей запит показано на рисунку 3.10.

```
const response :AxiosResponse<any>  =
    await axios.get( url: `https://api.monobank.ua/bank/jar/${longJarId}`,  config: {
    headers: {
        'Accept': 'application/json'
    }
});
```

Рисунок 3.10 – запит для отримання інформації про банку

Відповідь отримується у форматі JSON, що і вказано в запиті. Приклад відповіді зображено на рисунку 3 n.


```
{
  "amount": 2771656,
  "goal": 10000000,
  "ownerIcon": "https://ava-img.monobank.com.ua/NqJAEHONJZ4lEaerJztWu7fTz9droQQM29vtyp0jFVM=.jpg",
  "title": "Старлінки та екофлоу",
  "ownerName": "Мирослава В.",
  "currency": 980,
  "description": "",
  "jarId": "3HG6D2vW9N",
  "blago": false,
  "closed": false
}
```


Рисунок 3.11 – відповідь від API Монобанку

Важливо уникнути встановленого обмеження на кількість запитів до API. Тож було створено middleware updateCurrentInfo. Він використовує бібліотеку node-cron для планування і автоматизації виконання певних функцій у заданий часовий інтервал. Тут за встановленим розкладом кожну хвилину отримується перші 10 зборів впорядкованих за зростанням значення lastUpdated. Потім виконується ітерація по кожному збору коштів і для кожного викликається getCurrentInfo, якій передається ідентифікатор збору. На рисунку 3.12 можна побачити повну реалізацію планувальника.

```
cron.schedule( expression: '*/1 * * * *', func: async () : Promise<void> => {
    try {
        const fundraisings = await Fundraising.find().sort( arg: { lastUpdated: 1 }).limit( val: 10);

        console.log(fundraisings);

        for (const fundraising : Promise<Unpacked<Array<Hydrate... of fundraisings) {
            const fundraisingId = fundraising._id;
            await getCurrentInfo( req: { params: { fundraisingId } });
        }

        console.log('All fundraising information updated successfully');
    } catch (error) {
        console.error('Error updating fundraising information:', error);
    }
});
```

Рисунок 3.12 – реалізація планувальника

Таким чином було досягнуто збереження в базі даних актуальної інформації про збір.

3.2.6 Реалізація надсилання повідомлень користувачу на електронну пошту

Коли адміністратор підтверджує або ні збір користувачеві, то відповідно надсилається повідомлення про це на електронну пошту. Для цього було реалізовано функцію sendEmail, яка на вхід приймає ідентифікатор користувача та статус підтвердження збору. За допомогою SMTP-сервера Gmail можна надіслати повідомлення, що виконує nodemailer модуль. Для цього потрібно задати параметри host(адреса SMPT-сервера), port(порт SMPT-сервера, зазвичай 587), secure(чи використовувати захищене з’єднання), tls(Transport Layer Security), auth(об’єкт, який містить дані автентифікації на SMPT-сервері, тобто адреса електронної пошти та пароль, який надавався під час налаштувань сервера).

Отож реалізовано сервер, який виконує усі задані до його вимоги.

3.3 Реалізація клієнтської частини застосунку

3.3.1 Структура проєкту

Перш за все було сформовано структуру проєкту. Нижче на рисунку 3.13 зображено, яким чином було поділено теки.

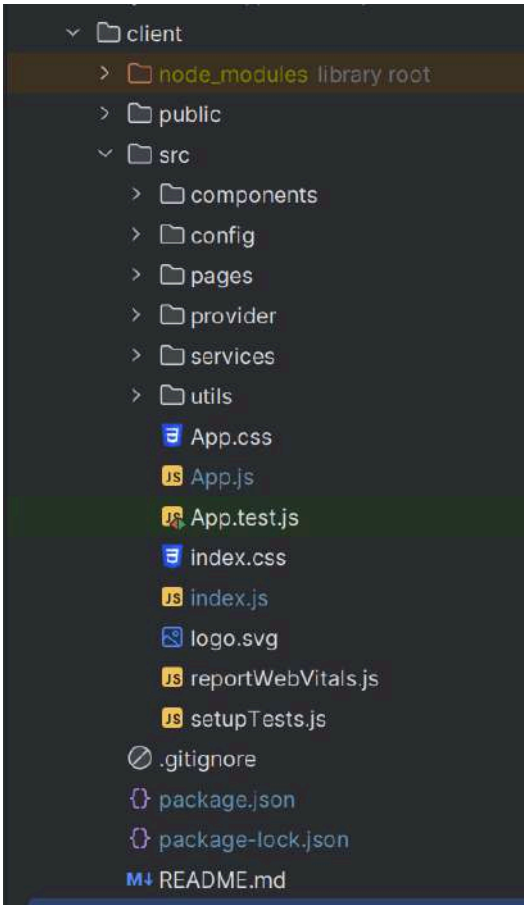

Рисунок 3.13 – Структура проєкту

Основний файл – це App.js, тобто головний компонент React, який відповідає за відображення користувацького інтерфейсу та маршрутизацію між різними сторінками.

3.3.2 Підключення стилів

Після встановлення Material-UI його треба підключити до проєкту. Також треба підключити визначену тему, тобто набір параметрів стилів, які визначаються зовнішній вигляд.

Для того, щоб визначені стилі діяли в усіх компонентах, тему треба підключити до головного компоненту. Перший крок – імпорт теми та

ThemeProvider, який дозволяє передавати стилі теми через контекст до всіх компонентів, розташованих у відповідному дереві компонентів. Це зображено на рисунку 3.15.

```
import theme from '../src/utils/theme';
import { ThemeProvider } from '@mui/material/styles';
```

Рисунок 3.15 – імпорт теми та ThemeProvider

Наступним крок – обгортання App.js в ThemeProvider, що можна побачити на рисунку 3.16.

```
return (
    <ThemeProvider theme={theme}>
        <CssBaseline />
        <Router>
            <Header />
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/auth" element={<AuthPage />} />
                <Route path="/admin" element={<AdminPage />} />
                <Route path="/profile" element={<UserProfilePage />} />
                <Route path="/fundraising" element={<GeneralPage />} />
                <Route path="/fundraising/createFundraising" element={<AddFundraising />} />
                <Route path="/fundraising/organizer/:userId" element={<OrganizerProfilePage />} />
            </Routes>
            <Footer />
        </Router>
    </ThemeProvider>
);
```

Рисунок 3. 16 – Підключення теми стилів

В подальшому всі компоненти, визначені в app.js будуть мати підключену тему.

3.3.3 Контроль доступу на основі ролей

Щоб забезпечити контроль доступу на основі ролей на стороні клієнта було визначено у файлі authProvider компонент AuthProvider, який буде обгортати всі компоненти застосунку та реалізувати систему автентифікації. Він зберігає та оновлює стан токену автентифікації, використовуючи локальне

сховище та бібліотеку Axios для HTTP-запитів. Також у файлі визначено hook useAuth, який дозволяє компонентам зручно отримувати доступ до токену та функції встановлення токену.

Щоб в подальшому в компонентах можна було користуватися визначеним хуком, його треба імпортувати. Щоб отримати значення токену та користуватися ним всередині компоненту потрібно витягнути його з useAuth.

3.3.4 Зв’язок з сервером. Сервіси.

Для відокремлення логіки взаємодії з сервером від компонентів було винесено окремо сервіси, які відправляють запити від клієнта до сервера. Кожен з сервісів відповідає за свої завдання.

Розглянемо взаємодію на прикладі сервісу, який повертає всі збори, що є підтвердженими. Перш за все треба сформувати адресу запиту до сервера, в цьому випадку це : `http://localhost:${SERVER_PORT}/api/fundraising/confirmedFundraising`. За допомогою бібліотеки Axios буде надсилатися GET-запит на вказану URL-адресу. Також буде передаватися токен автентифікації у заголовку запиту для автентифікації на сервері. Після отримання відповіді від сервера повертаються дані, які були отримані. В цьому випадку, це дані про підтверджені збори. У разі виникнення помилки під час виконання запиту, вона обробляється, а повідомлення про помилку виводиться у консоль. Описану вище логіку можна побачити на рисунку 3.20.

```
const BASE_URL : string  = `http://localhost:${SERVER_PORT}/api/fundraising`;

3 usages
const getConfirmedFundraisings = async (token, selectedCategory) : Promise<...>  => {
    try {
        let url : string  = `${BASE_URL}/confirmedFundraising`;
        if (selectedCategory) {
            url += `/${selectedCategory}`;
        }
        const response : AxiosResponse<any>  = await axios.get(url,  config: {
            headers: {
                Authorization: `Bearer ${token}`
            }
        });
        return response.data;
    } catch (error) {
        console.error('Error fetching confirmed fundraisings:', error);
        throw error;
    }
};
```

Рисунок 3.20 – сервіс, який робить запит до серверу

3.3.5 Компоненти

Всі компоненти знаходяться в теці components. Це основні блоки інтерфейсу, які можна використовувати повторно в різних місцях застосунку.

Наприклад, було створено компонент FundraisingCard, який відповідає за відображення кожного окремого збору на сторінці. На вхід він приймає властивості, основною з яких є fundraising, тобто об’єкт, який містить в собі всю інформацію про збір. На виході генерується візуальне представлення для збору, воно було оформлено як компонент Card – компонент із Material-UI.

В компоненті FundraisingCard використовується застилізований JarProgressBar. Цей компонент приймає на вхід значення поточної суми на банці та загальну суму збору. На виході візуальне зображення прогресу збору коштів у вигляді прогрес-бару, для цього також використовується компонент із Material-UI – LinearProgress.

В подальшому компонент FundraisingCard використовується в декількох місцях, наразі буде розглянуто його використання в ConfirmedFundraisingList.

Цей компонент відображає список підтверджених зборів із можливістю фільтрації за категорією, сумою та статусом організатора.

Важливою частиною компонента ConfirmedFundraisingList є виклик функції getConfirmedFundraisings, з раніше розглянутого сервісу. Після успішного отримання даних вони встановлюються за допомогою setFundraisings та надалі передаються.

Компонент ConfirmedFundraisingList взаємодіє з компонентом FundraisingCard через передачу пропсів при його відображенні у списку зборів коштів. Головний пропс це fundraising, який передає дані про конкретний збір і вже у FundraisingCard відображається.

Важливим моментом є натискання кнопки “зробити донат”, яка встановлює onShowModal=true, що забезпечує показ модального вікна додавання донату після повернення на головну сторінку застосунку.

3.3.6 Сторінки

Всі визначені компоненти використовуються у конкретних сторінках. Для цього вони імпортуються. Нижче на рисунку 3.21 можна побачити використання компонентів UnconfirmedFundraising та PotentialVerifiedUserList на сторінці AdminPage, яка вже безпосередньо показується користувачеві.

```
3 usages
function AdminPage() {
    return (
        <div>
            <UnconfirmedFundraising/>
            <PotentialVerifiedUsersList/>
        </div>
    );
}

2 usages
export default AdminPage;
```

Рисунок 3.21 – Використання компонентів на сторінках

3.4 Демонстрація інтерфейсу

3.4.1 Аутентифікація, реєстрація

На рисунку 3.22 можна побачити інтерфейс аутентифікації користувача.

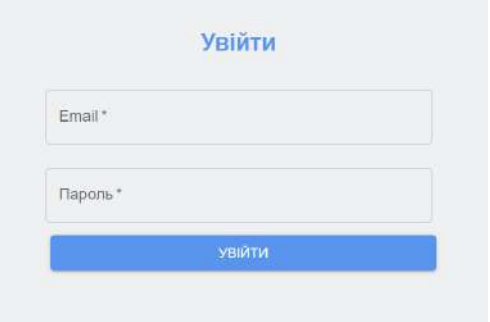

Рисунок 3.22 – Сторінка автентифікації

На рисунку 3.23 можна побачити інтерфейс реєстрації користустувача.

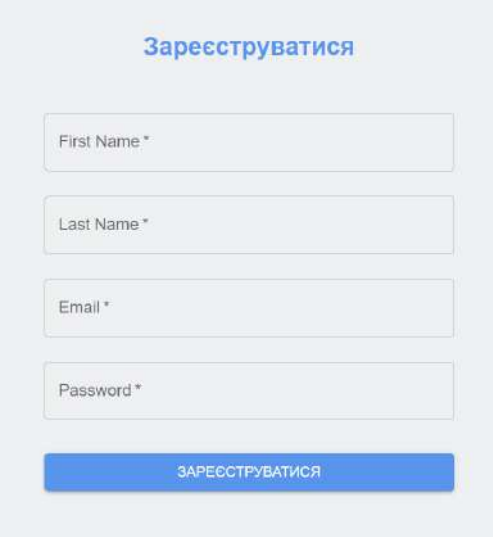

Рисунок 3.22 – Сторінка реєстрації

3.4.2 Головний екран

На головному екрані користувач бачить всі активні збори, які пройшли підтвердження, а також прогрес своєї цілі. Головний екран зображено на рисунку 3.23.

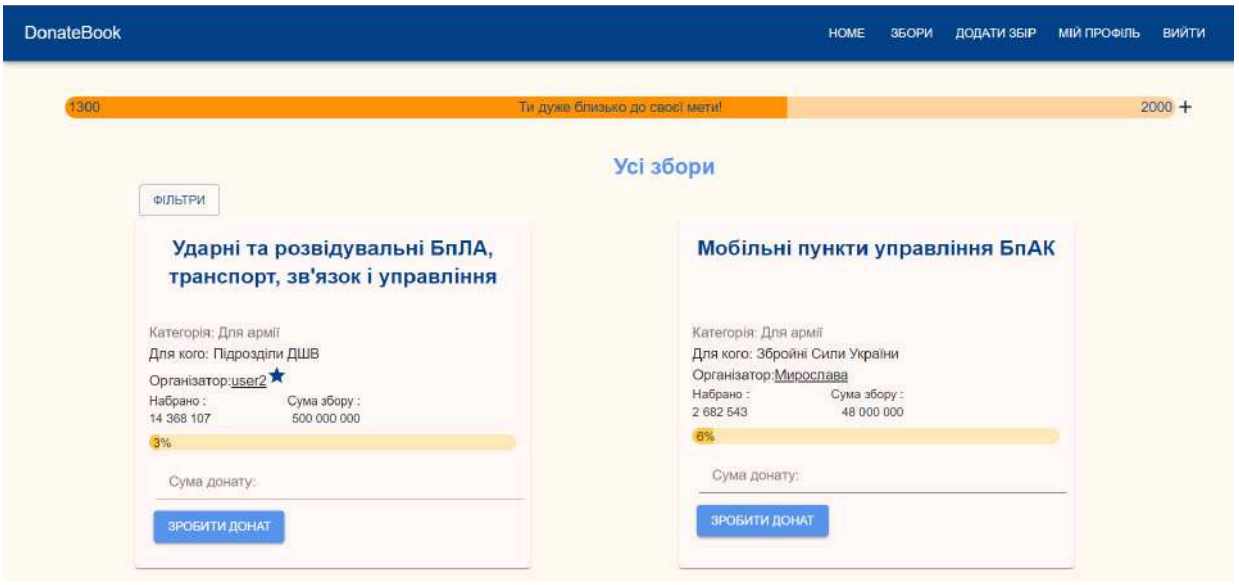

Рисунок 3.23 – Головний екран застосунку

Звідси можна переходити на свій профіль, фільтрувати збори. На рисунку 3.24 зображено можливі фільтри.

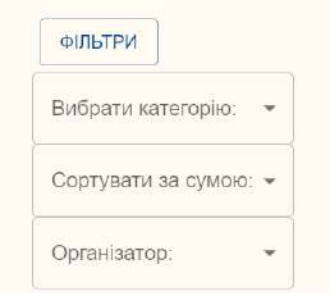

Рисунок 3.24 – Фільтри

На головній сторінці біля прогрес-бару особистої цілі можна натиснути кнопку “+” та додати свою ціль. Це зображено на рисунку 3.25.

Рисунок 3.25 - додавання донату до поточної суми

3.4.3 Профіль користувача

В профілі представлено інтерфейс редагування цілі, перегляд наявної та історія попередніх виконаних/невиконаних цілей.На рисунку 3.26 зображено редагування цілі.

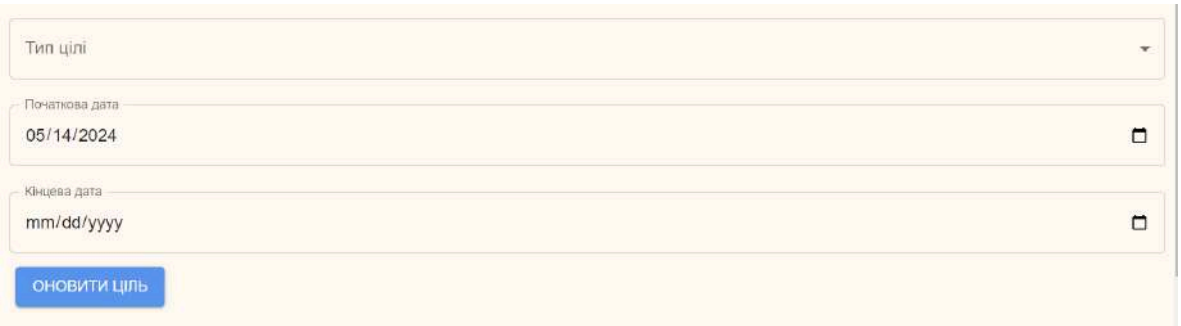

Рисунок 3.26 – редагування цілі

На рисунку 3.27 зображено представлення історії цілей користувача.

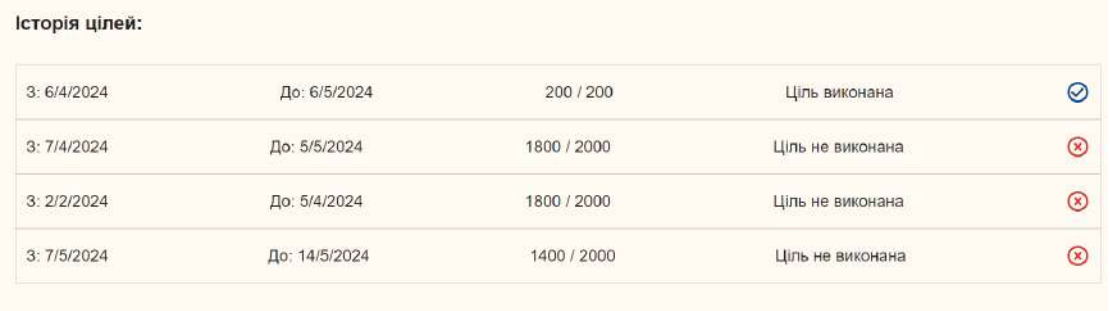

Рисунок 3.27 – історія цілей

3.6.4 Додавання збору

На рисунку 3.28 зображена форма додавання збору

Рисунок 3.28 – Додавання збору

3.5 Висновки до розділу 3

Отже, у цьому розділі було описано структуру та зв’язки між сутностями у моделі даних, визначено атрибути кожної сутності та описано їхнє значення. Розглянуто структуру серверної частини, виділено основні етапи її розробки. Реалізовано на Node.js із сервером Express серверну частину, підключено її до бази даних MongoDB. Було описано структуру клієнської частини, пояснено взаємодію з серверною за допомогою бібліотеки Axios. Розглядалися етапи розробки на React, пояснено зв’язок між різними компонентами. Розроблено графічний інтерфейс з використанням бібліотеки Material-UI.

Висновки

Отже, у процесі роботи було реалізовано клієнт-серверний застосунок, де користувачі можуть розміщувати свої збори та переглядати збори інших, задавати собі цілі, слідкувати за своїм прогресом. Перед реалізацією було ретельно вивчено та досліджено актуальність теми, обгрунтовано її. Було проведено аналіз актуальних рішень, поставлено завдання.

У цій роботі було проаналізовано вибір засобів розробки такі, як Node.js, React, MongoDB. Основні етапи реалізації було описано та пояснено. Тож, отримано веб- застосунок з визначеними вимогами.

Список використаних джерел

1. Бан (Інтернет). *Вікіпедія*. URL:https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%BD_(%D0%86%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82)
2. О. В. Панькова. О. Ю. Касперович.Українське волонтерство в умовах збройної російської агресії. *Економічний вісник Донбасу. Цифрова економіка та інформаційні технології*. 2022. № 2(68). С. 113-121, URL:http://dspace.nbuv.gov.ua/bitstream/handle/123456789/188015/13-Pankova.pdf?sequence=1
3. Instagram буде видаляти акаунти за мову ворожнечі. *Українські національні новини*. URL:https://unn.ua/news/instagram-bude-vidalyati-akaunti-za-movu-vorozhnechi
4. Допис у соціальній мережі X (Twitter) від monobank. URL:https://twitter.com/monobankua/status/1741048857927057468?ref_src=twsrc%5Etfw%7Ctwcamp%5Etweetembed%7Ctwterm%5E1741048857927057468%7Ctwgr%5Ecdd765ca4972a6b5e3e29b36d779bcb117fe3bf5%7Ctwcon%5Es1_&ref_url=https%3A%2F%2Fwww.epravda.com.ua%2Fnews%2F2023%2F12%2F30%2F708257%2F
5. Сайт благодійного фонду “Повернись живим”. URL:https://savelife.in.ua/
6. Сайт всеукраїнського гуманістичного руху “UAnimals”. URL:https://uanimals.org/
7. What is HTTP? *Cloudflare.* URL:https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/
8. Pankaj Pandey. Understanding of Client-Server Architecture and Communication Protocols. *Medium.*

URL:https://medium.com/@pankaj_pandey/understanding-of-client-server-architecture-and-communication-protocols-486884ed5f5e

9. What is the Simple Mail Transfer Protocol (SMTP)? *Cloudflare.* URL:https://www.cloudflare.com/learning/email-security/what-is-smtp/
10. HTTP Methods. *http. dev*. URL: https://http.dev/methods
11. Client Server Architecture – Detailed Explanation. *Interviewbit.* URL:https://www.interviewbit.com/blog/client-server-architecture/
12. Role-Based Access Control. *Auth0 by Okta*. URL:https://auth0.com/docs/manage-users/access-control/rbac
13. Анжані Тараїл. React JS: DOM і процес узгодження. *anywhereclub*. URL:https://aw.club/global/uk/blog/react-js-dom-and-how-the-reconciliation-process-works
14. Oladeji Tosin.Implementing Authentication and Authorization Using JWT In a Node.js Application. *Medium.* URL:https://blog.stackademic.com/implementing-authentication-and-authorization-using-jwt-in-a-node-js-application-7e68a49d456d
15. Authentication vs. Authorization. *Okta*. URL:https://www.okta.com/identity-101/authentication-vs-authorization/
16. REST API Introduction. *geeks for geeks.* URL:https://www.geeksforgeeks.org/rest-api-introduction/
17. Joseph Benharosh. What is REST API? in plain English. *PHPenthusiast.* URL: https://phpenthusiast.com/blog/what-is-rest-api
18. Katie Lawson. What Are Single Page Applications and Why Do People Like Them So Much? *Bloomreach*. URL:https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application
19. Vaishak. The Pros and Cons of Single-Page Applications (SPAs). *Medium.*URL:https://medium.com/@VAISHAK_CP/the-pros-and-cons-of-single-page-applications-spas-06d8a662a149

Додаток А

Статистика від Монобанку за 30 грудня 2023 року