Міністерство освіти і науки України
Львівський національний університет ім. Івана Франка
Факультет електроніки та комп’ютерних технологій
Кафедра системного проектування

КУРСОВА РОБОТА

на тему: “Реалізація системи контролю витрат благодійного фонду з Гугл автентифікацією”

Виконав:
студент групи ФеІм-12с
Мазуркевич Т. А.
Науковий керівник:
доцент Шувар Р. Я.

Львів - 2023р.

АНОТАЦІЯ

Ця курсова робота присвячена реалізації системи контролю витрат для благодійного фонду. Розглядається актуальна проблема прозорості фінансових потоків в благодійних організаціях, що сприяє зміцненню довіри між жертводавцями та фондом. Розроблена система забезпечує внесення, редагування та перегляд витрат благодійного фонду в режимі реального часу. Спеціальні інструменти пошуку, фільтрації та сортування дозволяють волонтерам та співробітникам фонду ефективно керувати витратами.

Проект використовує передові практики IT-індустрії, такі як неперервна інтеграція та постійна доставка (CI/CD), і вже активно застосовується в роботі фонду. Перспективи проекту включають введення додаткових модулів аналітики для глибшого аналізу витрат. Ця робота підкреслює значимість сучасних технологій для оптимізації роботи благодійних організацій, демонструючи нові можливості автоматизації обліку витрат і покращення взаємодії між фондом та його жертводавцями.

ЗМІСТ

АНОТАЦІЯ.....2
ЗМІСТ.....3
ВСТУП.....4
1. Опис теоретичної частини.....5
2. Використані технології.....7
3. Опис практичної частини.....13
Висновок.....25
Джерела..... 26

ВСТУП

Темою роботи є реалізації системи контролю витрат для благодійного фонду. Тему було обрано через її актуальність і можливість зробити корисний проект для благодійної організації. Благодійні фонди мають побудувати довіру з своїми жертводавцями. Якщо цей аспект не буде виконаний, то імовірно, що пожертви зменшуватимуться або припиняться. Одним з способів вибудовувати довіру є публічна демонстрація доходів і витрат фонду.

Було вирішено розробити систему контролю витрат яка підключена в основну базу даних проекту. Таким чином витрати, що вноситимуться співробітниками та волонтерами фонду зберігатимуться в базу даних, а звідти зможуть автоматично відображатися на веб сайті фонду. Таким чином буде реалізовано прозорість витрат і очікується, що це підсилить довіру жертводавців до благодійного фонду.

1. Опис теоретичної частини

Варто почати з формулювання цілі роботи. Ціль роботи це розробити зручний для користувача веб додаток, який має функції внесення редагування і переглядання витрат. Також важливими функціональними вимогами є можливість пошуку по витратах, фільтрування за параметрами, сортування. Додатковою вимогою є пропонування додатком варіантів автозаповнення деяких полей вводу базуючись на існуючих витратах в базі даних.

Першою дієї було виконання аналізу low-code рішень. Low-code це вид програмування, коли більшість проекту збирається у візуальному конструкторі з готових функціональних блоків, в яких є підтримка додавання власного коду для зміни поведінки. У випадку знаходження low-code рішення, яке б задовольняло вимогам, то ціль роботи була б частково змінена до функціоналу, який не можна замінити low-code рішенням і було б реалізовано більше функціоналу як фінальний результат роботи. Провівши аналіз придатності кількох low-code рішень для вирішення задачі цієї роботи було зроблено висновок, що жодна з них не відповідає вимогам. Основною причиною не обирати low-code рішення були обмеженість розширення проекту в довгостроковій перспективі і неможливість реалізувати елемент автозаповнення, який брав би інформацію з бази даних.

Було прийнято рішення реалізовувати проект як веб додаток. Альтернативним рішенням була реалізація мобільних додатків, але це б збільшило час розробки, а також через етап перевірки магазинами застосунків (App Store і Google Play) оновлення до наступної версії додатку зайняло б додатковий час.

Однією з вимога була безпека даних і доступу до веб додатку. Вибір був між написанням власної системи авторизації, використанням сторонніх систем

авторизації таких як Auth0 і використанням існуючого Google workspace як системи авторизації. Було прийнято рішення використовувати Google workspace для того щоб ідентифікувати користувача, а після цього видавати користувачу JWT токен, який надає йому доступ до веб додатку. Таким чином імплементація рішення буде найшвидшим, але завдяки генеруванні власного токена веб додаток залишить за собою весь контроль доступу і його можливі характеристики. Це потрібно було для того, щоб могти змінювати тривалість дії токена доступу до 2 годин.

Ціллю роботи є реалізувати і запровадити проект системи контролю витрат у досліджуваному благодійному фонді. Також додатковою ціллю є реалізувати проект використовуючи сучасні технології і передові практики IT індустрії, як наприклад Continuous integration and continuous delivery (CI CD).

2. Використані технології

Для фронтенду використовуватимуться класичні для веб додатків технології JavaScript, HTML (Hyper Text Markup Language), CSS (Cascading Style Sheets). Фронтенд бібліотекою для проекту був обраний ReactJS. ReactJS - це безкоштовна бібліотека на JavaScript з відкритим кодом для створення інтерфейсів користувача на основі компонентів.

Для бекенду проекту було вирішено використовувати NodeJS. NodeJS є відкритою, кросплатформовою середовищем виконання JavaScript, яке побудоване на платформі JavaScript V8 компанії Google. Ця платформа також використовується в браузері Google Chrome. NodeJS дозволяє виконувати JavaScript на сервері, поза контекстом веб-браузера, що робить його ідеальним для розробки серверних застосунків.

Для хостингу проекту було вирішено використовувати Google Cloud Platform, а саме сервіс Google App Engine. Це serverless рішення, яке дозволяє розробникам створювати та розгортати веб-додатки та сервіси в інфраструктурі Google Cloud без необхідності самостійно керувати основною інфраструктурою. За допомогою Google App Engine розробники можуть зосередитися на написанні коду та створенні своїх програм, в той час як Google Cloud займається масштабуванням, розгортанням і керуванням інфраструктурою. Він абстрагує базову інфраструктуру, забезпечуючи середовище, де додатки можуть працювати надійно та автоматично збільшувати чи зменшувати масштаб відповідно до потреб.

Як базу даних було вирішено використовувати NoSQL базу даних MongoDB, яка вже використовувалася в інших проектах фонду. Її гнучка модель даних, масштабованість і ефективність роблять її популярним вибором для розробників і

організацій, які працюють з великими наборами даних, що можуть змінюватися. Одиниця даних в MongoDB називається документом. Те, що в традиційних SQL базах даних називають таблицями, в MongoDB називається колекціями. Щоб не розгортати вручну MongoDB екземпляр і не займатися масштабуванням і керуванням інфраструктурою, було вирішено використовувати хмарний сервіс Mongo Atlas.

Засобом автентифікації користувача було обрано Google автентифікація.

Google автентифікація використовує протокол OAuth 2.0. OAuth 2.0 - це відкритий протокол авторизації, який дозволяє додаткам отримувати обмежений доступ до облікових записів користувачів на сторонніх сервісах. Він створений для роботи з HTTP і може бути використаний в веб-додатках, мобільних додатках та десктопних додатках.

Щоб зрозуміти, як працює OAuth 2.0, спочатку потрібно ознайомитися з деякими термінами:

1. Клієнт - це додаток, який хоче отримати доступ до облікового запису користувача на сторонньому сервісі.
2. Ресурсний власник (Resource Owner) - це користувач, який володіє обліковим записом, до якого додаток хоче отримати доступ.
3. Сервер ресурсів (Resource Server) - це сервер, на якому зберігаються дані користувача. Додаток хоче отримати доступ до цих даних.
4. Сервер авторизації (Authorization Server) - це сервер, який знає користувача і може дати додатку токен доступу для доступу до даних користувача на сервері ресурсів.

Детальніше процес виглядає наступним чином:

1. Авторизація: Коли користувач використовує додаток (наприклад, веб-додаток) і додаток вимагає доступ до облікового запису користувача на сторонньому сервісі (наприклад, Google), він перенаправляє користувача на сторінку авторизації на сервері авторизації (в цьому випадку Google).
2. Надання дозволу: На цій сторінці користувачу запропонують дозволити або відмовити додатку в доступі до своїх даних. Якщо користувач погоджується, він буде перенаправлений назад до додатка.
3. Отримання токену доступу: Коли користувач повертається до додатка, сервер авторизації включає в URL спеціальний код. Додаток використовує цей код для отримання токену доступу до даних користувача на сервері авторизації.
4. Використання токену доступу: За допомогою цього токену доступу додаток може звернутися до сервера ресурсів і отримати доступ до даних користувача. Токен доступу діє протягом обмеженого часу і дозволяє отримати доступ лише до певних даних, які визначені в "області дії" (scope).

Веб додаток буде використовувати формат JWT (JSON Web Token). JWT (JSON Web Token) - це стандарт токенів між сторонами у вигляді JSON об'єкта. JWT складається з трьох частин: заголовка (header), тіла (payload) та підпису (signature). Цей формат використовується для передачі інформації між клієнтом і сервером або між послідовними сервісами як спосіб автентифікації та авторизації.

Заголовок (header) JWT містить тип токена (typ) і алгоритм шифрування (alg), який використовується для генерації підпису. Заголовок закодований в Base64.

Тіло (payload) JWT містить корисну інформацію, таку як ідентифікатор користувача (subject), термін дії токена (expiration time) та додаткові дані. Тіло також закодоване в Base64.

Підпис (signature) JWT використовується для перевірки цілісності токена і підтвердження, що він не був змінений під час передачі. Підпис генерується шляхом об'єднання закодованого заголовка, закодованого тіла та приватного ключа за допомогою алгоритму шифрування, зазначеного у заголовку.

JWT може бути використаний для реалізації механізму автентифікації та авторизації. Коли користувач успішно аутентифікується на сервері, йому виділяється JWT, який він може включити до запиту до сервера. Сервер може перевірити цей токен, перевіривши підпис і його цілісність. Після перевірки сервер може використовувати інформацію з тіла токена для авторизації та прийняття рішень про доступ користувача до ресурсів.

JWT є популярним механізмом для забезпечення безпеки та ідентифікації в розподілених системах, таких як веб-додатки і мікросервіси. Він має переваги у порівнянні з традиційними методами сеансового керування, такими як використання кукі, оскільки JWT не потребує зберігання стану на сервері і може бути переданий із запитом, зменшуючи навантаження на сервер та роблячи систему більш масштабованою.

Для того, що в індустрії називають CI/CD (CI/CD означає "постійну інтеграцію та постійну доставку" і є практикою розробки програмного забезпечення, яка дозволяє автоматизувати процеси збирання, тестування та впровадження коду в програмні системи. Таким чином нові зміни в код проекту

мають змогу швидше інтегруватися в основну вітку коду, а користувачі частіше отримують оновлення.), було прийнято рішення використовувати GitHub actions.

GitHub Actions - це потужний інструмент для автоматизації розробки програмного забезпечення, який дозволяє створювати та налаштовувати робочі процеси безпосередньо в репозиторіях на GitHub.

GitHub Actions дозволяє створювати власні робочі процеси або використовувати готові шаблони для автоматизації завдань у проекті. Є можливість налаштовувати робочі процеси, щоб вони реагували на події, такі як коміти коду, створення або закриття пул-реквестів, створення або видалення віток.

Для перевірки якості коду на етапі розробки було вирішено використовувати інструмент ESLint. ESLint - це популярний інструмент для статичного аналізу коду в JavaScript проектах. Він призначений для виявлення потенційних проблем, помилок та неконсистентностей в коді з метою поліпшення якості і забезпечення відповідності стилю коду в рамках проекту або команди.

Основна мета ESLint - це допомогти розробникам писати чистий, зрозумілий та безпомилковий код. Його можна використовувати в будь-якому проекті, що використовує JavaScript. Це може бути веб-додаток, серверний код або скрипти командного рядка.

ESLint перевіряє код на основі набору правил, які можна налаштувати або використовувати один з популярних наборів правил, таких як Airbnb, Google, Standard тощо. Ці правила визначають, як код повинен бути оформлений, які конструкції потрібно уникати, які стилі найкраще використовувати тощо. ESLint допомагає виявляти такі проблеми, як неправильне використання змінних,

неоголошені змінні, некоректне використання функцій, несумісність між версіями ECMAScript, стилістичні неузгодженості тощо.

Одна з переваг ESLint - це його гнучкість. Є можливість налаштувати правила, виключати деякі правила, додавати свої правила або навіть створювати власні плагіни для додаткових перевірок.

ESLint можна використовувати з різними інструментами розробки. Цей інструмент можна запускати з командного рядка, інтегрувати зі своїм інтегрованим середовищем розробки (IDE) або використовувати його в якості допоміжної перевірки перед збіркою або розгортанням коду.

Щоб почати використовувати ESLint в проекті, потрібно встановити його як залежність проекту і налаштувати файл конфігурації зі списком правил. Після цього можна запустити ESLint, і він надасть звіт про будь-які виявлені проблеми в коді.

ESLint є потужним інструментом для покращення якості JavaScript коду. Він допомагає забезпечити консистентність, виявляє помилки та некоректне використання, а також полегшує спільну роботу в команді, де кожен розробник дотримується однакових стандартів оформлення коду.

3. Опис практичної частини

Реалізацію практичної частини було почато з створення Direnv (.envrc) файлу. Direnv - це утиліта для керування змінними середовища в командному рядку. Вона дозволяє автоматично встановлювати та змінювати змінні середовища, коли користувач переходить у певну директорію або виконує команди в цій директорії. Головна мета direnv - це полегшити розробку програмного забезпечення та управління залежностями, забезпечуючи чистоту та прозорість у кожному проекті. Конкретно в прикладі цієї роботи Direnv було використано для встановлення потрібної версії NodeJS, а саме NodeJS v16.

Далі у головній директорії проекту було створено дві директорії "frontend" і "backend". За допомогою утиліти Create React App (CRA) було згенеровано пустий ReactJS проект у директорії frontend. Create React App є набором інструментів, що дозволяє швидко налаштувати та створити новий проект React. Він надає попередньо налаштований набір інструментів та конфігурацію, що дозволяє швидко почати розробку без необхідності налаштовувати середовище з нуля.

Для бекенду було вирішено використовувати фреймворк ExpressJS. ExpressJS - це легкий та гнучкий веб-фреймворк, заснований на мові JavaScript та побудований на основі NodeJS. Він надає широкий набір функцій для створення веб-додатків та API. ExpressJS дозволяє легко маршрутизувати HTTP-запити, обробляти параметри запитів, робити перенаправлення запиту та налаштовувати middleware.

Наступним кроком було реалізовано автентифікацію користувачів за допомогою Google і видачу їм JWT токенів. Тут надано детальний опис кожного кроку цього процесу.

1. Користувач запускає веб-додаток, який відображає сторінку авторизації.
2. На цій сторінці розміщена кнопка "Sign In", яку користувач натискає для початку процесу авторизації.
3. Веб-додаток генерує URL-адресу з потрібними параметрами OAuth2, яка включає інформацію про додаток та дозволи, необхідні для авторизації.
4. Користувач переходить за цим URL-адресом і направляється до сторінки авторизації Google.
5. На сторінці авторизації Google користувач вводить свої облікові дані (наприклад, електронну пошту та пароль) для входу до свого облікового запису Google.
6. Після успішного входу користувач надає дозвіл веб-додатку на доступ до своїх облікових даних.
7. Google генерує "authorization_code" після успішної авторизації користувача і перенаправляє користувача назад до веб-додатку.
8. У веб-додаток передається "authorization_code" як параметр у URL-адресі повернення.
9. Веб-додаток передає "authorization_code" з вхідним запитом до свого бекенду, щоб отримати токен доступу.
10. Бекенд використовує "authorization_code", включений у запит, для звернення до сервера авторизації Google і обміну його на токен доступу.
11. За допомогою отриманого токену бекенд робить додаткові запити до Google, щоб отримати інформацію про користувача, включаючи його електронну адресу.
12. Веб-додаток перевіряє отриману електронну адресу зі своєю базою даних.
13. Якщо користувач ідентифікується успішно, веб-додаток генерує власний JWT токен доступу та зберігає його в HTTP-only cookie на боці клієнта. Це забезпечує безпеку токену, оскільки він не доступний для JavaScript.

14. Бекенд надсилає успішну відповідь на фронтенд зі статусом коду 200, підтверджуючи успішну авторизацію і доступ користувача до додатку.
15. Користувач отримує доступ до функцій і ресурсів веб-додатку, які вимагають авторизації.

Нижче додано код, який створює JWT токен і повертає його на фронтенд.

```
jwt.sign({ email: userEmail },
    process.env.JWT_SECRET,
    { expiresIn: `${TOKEN_EXPIRES_IN}s` },
    (err, token) => {
    // Обробку помилок пропущено для простоти прикладу
    res
      .cookie('auth', token, getAuthCookieSettings())
      .status(200)
      .send({
        authSuccess: true,
        tokenExpiresOn: moment().add(TOKEN_EXPIRES_IN,
'seconds').toDate(),
      });
  });
```

Для того, щоб перевіряти чи має користувач доступ робити конкретний запит, було реалізовано функцію, яка використовується в ExpressJS через патерн Middleware (проміжне програмне забезпечення). Middleware - це патерн проектування, який використовується для обробки запитів та відповідей у розподіленій системі або додатку. Він представляє собою ланцюг компонентів, через які проходять запити, виконуючись послідовно або паралельно.

Патерн middleware широко застосовується у багатьох галузях розробки програмного забезпечення, включаючи веб-розробку, серверні додатки та мікросервісні архітектури. Він надає модульну та гнучку архітектуру, що дозволяє

додавати, змінювати та видаляти компоненти проміжного програмного забезпечення без значних змін у основному коді додатка.

Код middleware функції:

```
const authorization = (req, res, next) => {
  const token = req.cookies.auth;
  if (!token) {
    return res.sendStatus(401);
  }

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded)
=> {
    if (err) {
      return res.sendStatus(401);
    }
    req.userEmail = decoded.email;
    return next();
  });
};
```

Використання функції оголошується перед оголошенням методів обробки запитів. Таким чином функція виконується першою і тільки у випадку успішного виконання починає виконуватися метод обробки запиту. Код:

```
app.use('/api/auth', authRouter);
app.use(authMiddleware);
app.use('/api/spendings', spendingsRouter);
```

Для зберігання налаштувань середовища використовується модуль Dotenv. Dotenv - це модуль, який використовується в Node.js для завантаження змінних середовища з файлу .env в процес додатку. Файл .env містить набір змінних середовища, які використовуються в додатку для налаштування його роботи, наприклад, конфігураційні параметри або секретні ключі. Це дозволяє зберігати конфіденційні дані або налаштування додатку у змінних середовища, що полегшує

управління конфігурацією та дозволяє зберігати ці значення поза вихідним кодом додатку. Крім того, це дозволяє розробникам мати різні конфігурації для різних середовищ, таких як розробка, тестування. Нижче додано приклад .env файлу цієї роботи.

```
NODE_ENV=
BACKEND_URL=
FRONTEND_URL=
ATLAS_URI=
DB_NAME=
TOKEN_COOKIE_DOMAIN=
USERS_THAT_HAVE_ACCESS_TO_THE_APP=
OAUTH_CLIENT_ID=
OAUTH_CLIENT_SECRET=
OAUTH_REDIRECT_URI=
JWT_SECRET=
PORT=
```

Ось опис кожного з параметрів:

1. NODE_ENV: Цей параметр використовується для визначення поточного середовища роботи додатка (наприклад, "development", "production", "test"). Він може використовуватися для налаштування різних параметрів залежно від середовища.
2. BACKEND_URL: Цей параметр містить URL-адресу для з'єднання з бекендом додатка. Він використовується для встановлення з'єднання між фронтендом та бекендом.
3. FRONTEND_URL: Цей параметр містить URL-адресу фронтенду додатка. Він використовується для налаштування доступу до фронтенду з інших серверів.
4. ATLAS_URI: Цей параметр містить URI-рядок підключення до бази даних MongoDB Atlas. Він використовується для встановлення з'єднання з базою даних.

5. DB_NAME: Цей параметр містить назву бази даних, з якою веб додаток має працювати. Він використовується для вказівки бази даних, з якою потрібно встановити з'єднання.
6. TOKEN_COOKIE_DOMAIN: Цей параметр вказує домен для куки-файлів, що містять JWT токени авторизації. Він використовується для налаштування домену, на якому куки будуть використовуватись.
7. USERS_THAT_HAVE_ACCESS_TO_THE_APP: Цей параметр містить список користувачів, які мають доступ до додатка. Він може використовуватись для обмеження доступу лише для певних користувачів.
8. OAUTH_CLIENT_ID: Цей параметр містить ідентифікатор клієнта для протоколу OAuth. Він використовується для авторизації додатка на сторонньому сервісі.
9. OAUTH_CLIENT_SECRET: Цей параметр містить секретний ключ для протоколу OAuth. Він використовується для авторизації додатка на сторонньому сервісі.
10. OAUTH_REDIRECT_URI: Цей параметр містить URI-адресу перенаправлення після авторизації з протоколом OAuth. Він використовується для перенаправлення користувача на потрібну сторінку після авторизації.
11. JWT_SECRET: Цей параметр містить секретний ключ для підпису та перевірки JWT-токенів (JSON Web Tokens). Він використовується для забезпечення безпеки та цілісності токенів.
12. PORT: Цей параметр визначає порт, на якому буде запущений сервер додатка. Він використовується для вказівки порту, на якому сервер буде слухати з'єднання.

Для автоматичної збірки і розгортання проекту було використано платформу GitHub Actions. Нижче наведено код з поясненнями до кожного блоку.

```yaml
name: Deploy to App Engine

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    permissions:
      contents: 'read'
      id-token: 'write'

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: 16

      - name: Install dependencies
        run: npm ci
        working-directory: ./frontend

      - name: Build frontend
        run: npm run build
        working-directory: ./frontend

      - name: Install dependencies
        run: npm ci
        working-directory: ./backend

      - name: Create .env file
        working-directory: ./backend
        run: |
          echo 'NODE_ENV=${{ secrets.NODE_ENV }}
            BACKEND_URL=${{ secrets.BACKEND_URL }}
```

```
            FRONTEND_URL=${{ secrets.FRONTEND_URL }}
            ATLAS_URI=${{ secrets.ATLAS_URI }}
            DB_NAME=${{ secrets.DB_NAME }}
            TOKEN_COOKIE_DOMAIN=${{
secrets.TOKEN_COOKIE_DOMAIN }}
            USERS_THAT_HAVE_ACCESS_TO_THE_APP=${{
secrets.USERS_THAT_HAVE_ACCESS_TO_THE_APP }}
            OAUTH_CLIENT_ID=${{ secrets.OAUTH_CLIENT_ID }}
            OAUTH_CLIENT_SECRET=${{
secrets.OAUTH_CLIENT_SECRET }}
            OAUTH_REDIRECT_URI=${{
secrets.OAUTH_REDIRECT_URI }}
            JWT_SECRET=${{ secrets.JWT_SECRET }}' > ./.env

      - name: 'Google cloud auth'
        uses: 'google-github-actions/auth@v1'
        with:
          credentials_json: '${{ secrets.GCP_SA_KEY }}'

      - name: 'Deploy to App Engine'
        id: 'deploy'
        uses: 'google-github-actions/deploy-appengine@v1'
        with:
          project_id: '${{ secrets.GCP_PROJECT_ID }}'
          working_directory: './backend'

      - name: Print deploy outputs URL
        run: 'curl "${{ steps.deploy.outputs.url }}"'
```

1. name: Deploy to App Engine: Назва блоку, який описує розгортання на App Engine.
2. on: Вказує подію, яка викликає розгортання. У цьому випадку, розгортання відбувається при зміні коду в гілці "main".
3. jobs: Описує список робочих задач, які повинні бути виконані.
4. deploy: Назва робочої задачі, яка відповідає за розгортання.
5. runs-on: Вказує операційну систему, на якій будуть виконуватись дії. У цьому випадку, використовується остання версія Ubuntu.

6. permissions: Встановлює права доступу до вмісту репозиторію та ідентифікаційного токена. В даному випадку, режим "read" встановлений для вмісту, а "write" для ідентифікаційного токена.
7. steps: Список дій, які необхідно виконати.
8. name: Checkout code: Завантажує код репозиторію за допомогою дії actions/checkout@v2.
9. name: Set up Node.js: Налаштовує середовище Node.js за допомогою дії actions/setup-node@v2. Встановлюється версія 16 Node.js.
10. name: Install dependencies: Встановлює залежності проекту. Спочатку встановлюються залежності для ./frontend, а потім для ./backend.
11. name: Build frontend: Виконує збірку фронтенду за допомогою команди npm run build.
12. name: Create .env file: Створює файл .env з конфігураційними параметрами, які беруться з сховища Secrets репозиторію.
13. name: 'Google cloud auth': Аутентифікується в сервісі Google Cloud за допомогою сертифікатів, які зберігаються в Secrets репозиторію.
14. name: 'Deploy to App Engine': Розгортає додаток на Google App Engine за допомогою дії google-github-actions/deploy-appengine@v1. Вказується ідентифікатор проекту Google Cloud.
15. name: Print deploy outputs URL: Виводить URL-адресу розгорнутого додатку, отриману з попереднього кроку.

Реалізований веб додаток дозволяє створювати, редагувати та переглядати витрати. Сторінка перегляду витрат також включає пошук, сортування і можливість розширити чи сховати вибрані колонки. На рисунку 3.1 відображено сторінку створення витрати. На рисунку 3.2 відображено список витрат. Кожна витрата має кнопку "Edit", яка відкриває сторінку редагування зображену на рисунку 3.3.

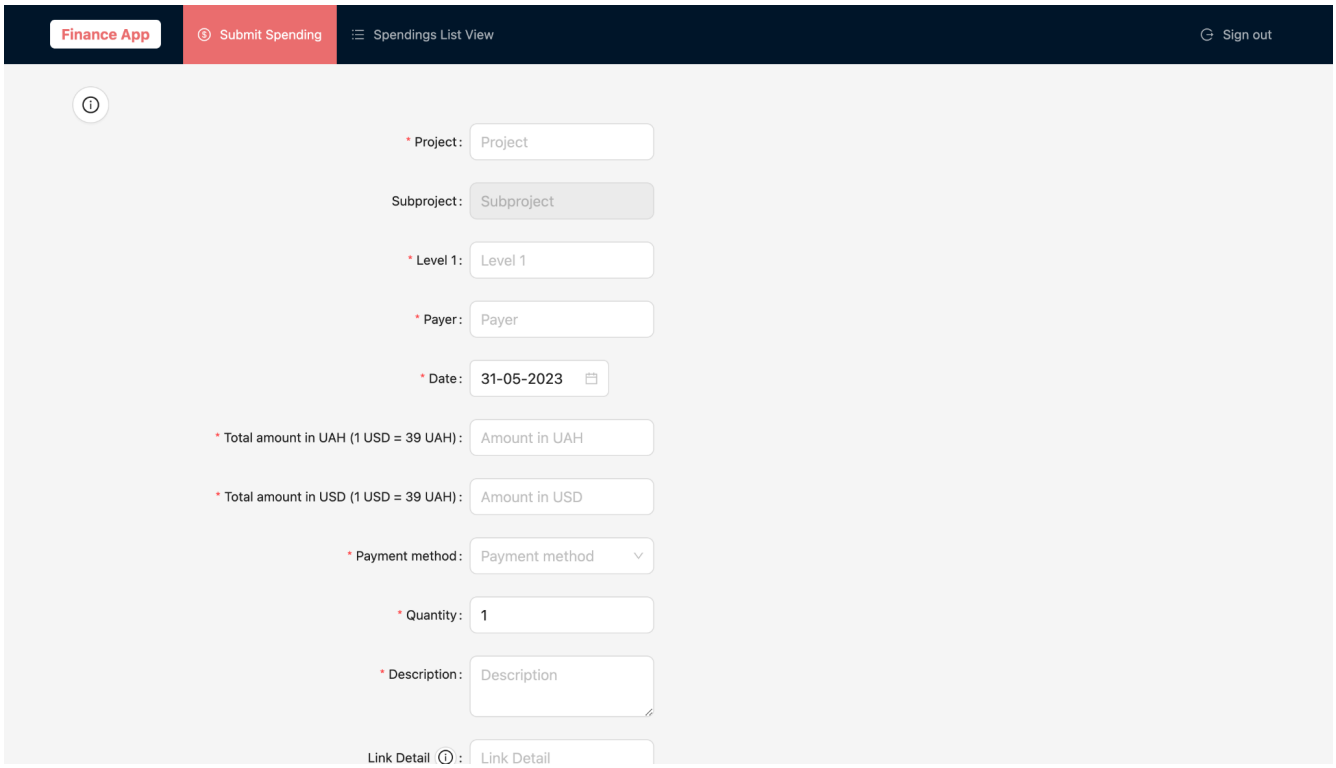
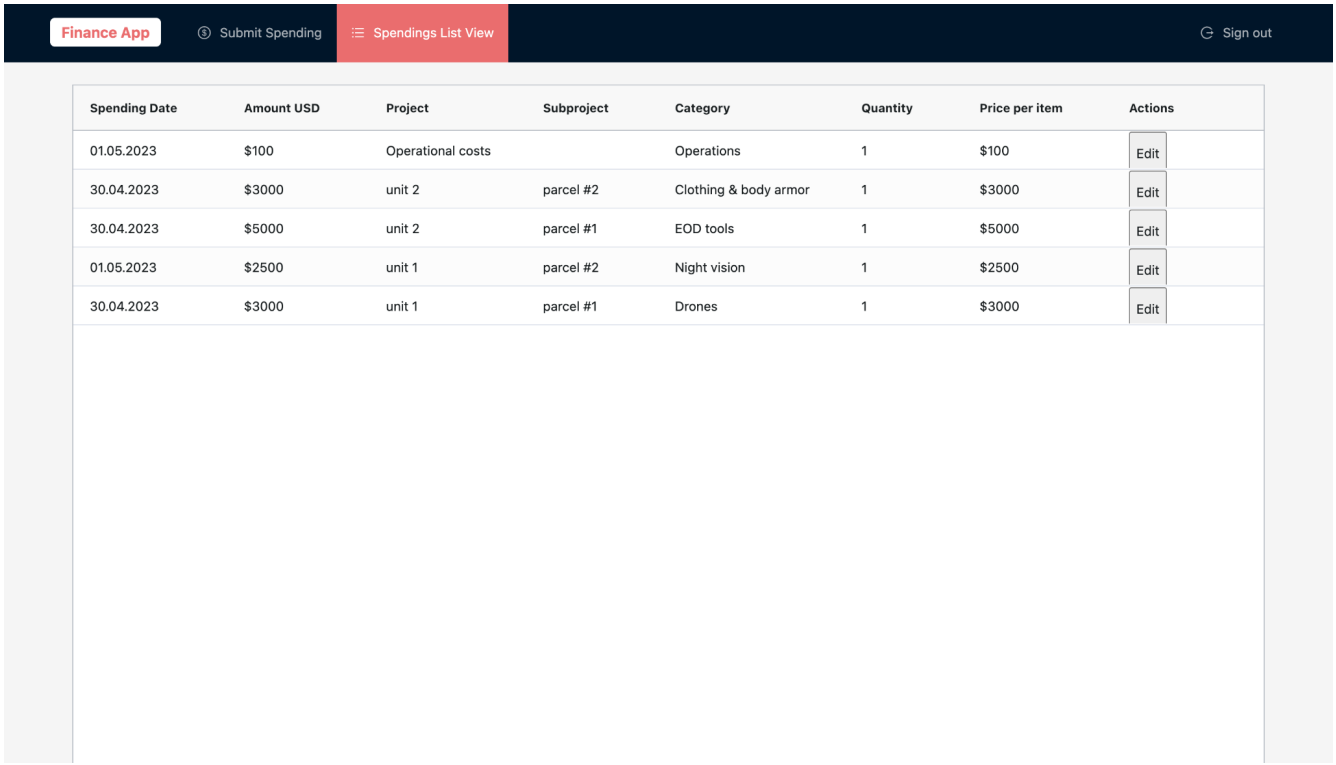


Рис. 3.1 Сторінка створення витрати



Spending Date	Amount USD	Project	Subproject	Category	Quantity	Price per item	Actions
01.05.2023	$100	Operational costs		Operations	1	$100	Edit
30.04.2023	$3000	unit 2	parcel #2	Clothing & body armor	1	$3000	Edit
30.04.2023	$5000	unit 2	parcel #1	EOD tools	1	$5000	Edit
01.05.2023	$2500	unit 1	parcel #2	Night vision	1	$2500	Edit
30.04.2023	$3000	unit 1	parcel #1	Drones	1	$3000	Edit

Рис. 3.2 Сторінка перегляду витрат

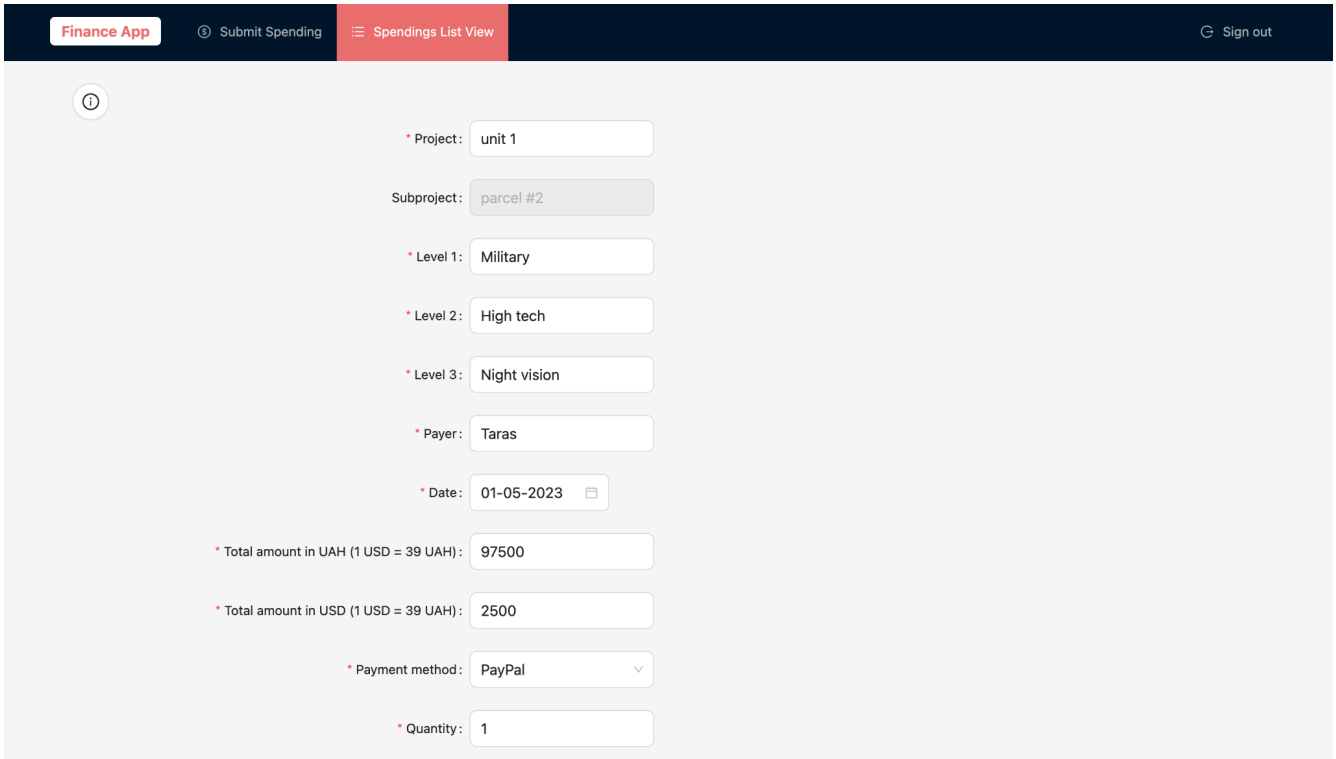


Рис. 3.3 Сторінка редагування витрати

Для більшого рівня безпеки тільки користувачі які знаходять в Google організації благодійного фонду можуть спробувати автентифікуватися в додатку. На рисунку 3.4 зображено, як виглядає помилка автентифікації, якщо користувач належить до іншої організації.

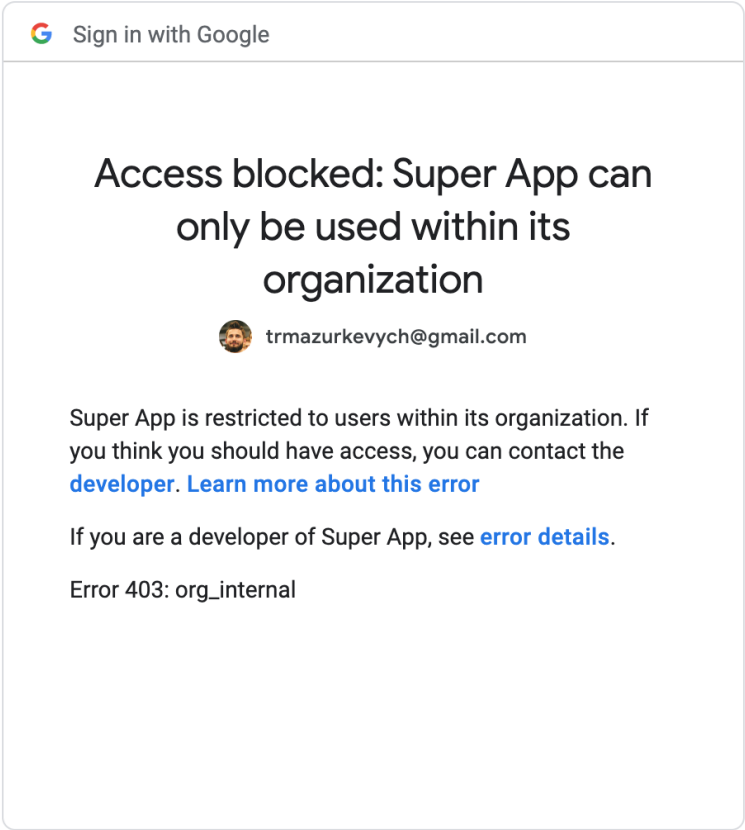


Рис. 3.4 Помилка автентифікації

Висновок

Система контролю витрат, розроблена в ході даної курсової роботи, вже успішно використовується благодійним фондом, що свідчить про її корисність та ефективність. Система не лише сприяє впровадженню прозорості у розподіл та використання благодійних коштів, але й підсилює довіру жертводавців до фонду, що може стимулювати збільшення пожертв.

Цей проект підтвердив актуальність та значимість використання сучасних технологій для оптимізації роботи благодійних організацій. Зокрема, реалізована система відкриває широкі можливості для автоматизації обліку витрат, спрощення контролю за їх розподілом та поліпшення зв'язку між фондом та його жертводавцями.

Плани на майбутнє включають розширення функціоналу системи контролю витрат, включаючи введення додаткових модулів аналітики для глибшого аналізу витрат. Це допоможе відслідковувати тенденції та виявляти можливі проблемні зони у використанні коштів. Також важливо зосередитися на розробці більш інтуїтивного та зручного інтерфейсу для кінцевих користувачів, що полегшить внесення даних про витрати та перегляд статистики.

Джерела

Ресурс Інтернету: Офіційна документація Google App Engine. URL: https://cloud.google.com/appengine/docs (дата звернення 10.03.2023)

Ресурс Інтернету: Офіційна документація GitHub Actions. URL: https://docs.github.com/en/actions (дата звернення 10.03.2023)
Ресурс Інтернету: Офіційна документація ReactJS. URL: https://react.dev/reference/react (дата звернення 10.03.2023)
Ресурс Інтернету: Офіційна специфікація JWT. URL: https://jwt.io/ (дата звернення 10.03.2023)
Ресурс Інтернету: Офіційна специфікація Oauth 2.0. URL: https://www.rfc-editor.org/rfc/rfc6749 (дата звернення 10.03.2023)