МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Допустити до захисту

В.о. завідувача кафедри

_____ Мнацаканян М.С.

(підпис) *(ПІБ завідувача кафедри)*

«___»_____20__ р.

«РОЗРОБКА СОЦІАЛЬНО-ГУМАНІТАРНОЇ ВЕБ-ПЛАТФОРМИ ДЛЯ ВПРОВАДЖЕННЯ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ»

Кваліфікаційна робота
здобувача вищої освіти
першого (бакалаврського) рівня
вищої освіти
освітньо-професійної програми
«____Комп'ютерні науки_____»
(назва освітньо-професійної програми)

___Ліскун Родіон Сергійович___
(прізвище, імя, по батькові здобувача вищої освіти)

Науковий керівник:
___Міщенко А.В., д.т.н., професор___
(прізвище, ініціали, науковий ступінь, вчене звання)

Рецензент:
___Лукашенко В.В., к.т.н., доцент___

(прізвище, ініціали, науковий ступінь, вчене звання, місце роботи)

Кваліфікаційна робота
захищена з
оцінкою_____
Секретар
ЕК_____ «____»
_____20__ р.

Київ -2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Рівень вищої освіти *бакалавр*
Шифр та назва спеціальності *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри __к.т.н.__
(науковий ступінь, вчене звання)
_____ ___Мнацаканян М.С.___
(підпис) *(ПІБ завідувача кафедри)*
«__» _____ 20__ р.

ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Ліскуна Родіона Сергійовича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка соціально-гуманітарної веб-платформи для впровадження волонтерської діяльності»
керівник роботи_Міщенко А.В., д.т.н., професор_,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Маріупольського державного університету від «___» _____20__ р. №__
2. Строк подання здобувачем роботи 30.05.2023 .
3. Вихідні дані до роботи (мета, об'єкт, предмет):

Об'єкт дослідження – соціально-гуманітарна веб-платформа.

Предмет дослідження – веб-платформа для впровадження волонтерської діяльності в соціально-гуманітарних проектах, яка розроблена з

використанням сучасних веб-технологій, таких як Prisma, Next.js, React та Tailwind CSS.

Мета кваліфікаційної роботи – на основі проведеного аналізу теоретичних аспектів волонтерства та веб-технологій, а також концепції соціально-гуманітарної веб-платформи для волонтерства, реалізувати соціально-гуманітарну веб-платформу з використанням розглянутих технологій.

4. Зміст роботи (перелік питань, які потрібно розробити)

Розділ 1. Аналіз теоретичних аспектів волонтерства та веб-технологій.

Розділ 2. Розробка концепції соціально-гуманітарної веб-платформи для волонтерства.

Розділ 3. Реалізація веб-платформи для волонтерства.

Розділ 4. Впровадження та оцінка ефективності веб-платформи.

5. Дата видачі завдання 01.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз предметної області та порівняння можливостей різних веб-платформ. Написання першого розділу.	23.03.2023	
2.	Розробка концепції соціально-гуманітарної веб-платформи для волонтерства. Написання другого розділу	30.03.2023	
3.	Розробка веб-платформи. Написання третього розділу	03.04.2023	
4.	Впровадження та оцінка ефективності веб-платформи Написання четвертого розділу	01.05.2023	
5.	Редагування пояснювальної записки кваліфікаційної роботи.	07.05.2023	
6.	Оформлення кваліфікаційної бакалаврської	14.05.2023	
7.	Розробка презентації	20.05.2023	

Здобувач _____ __Ліскун Р.С.__

(підпис) *(прізвище та ініціали)*

Науковий керівник роботи _____ _____Міщенко А.В_____

(підпис) *(прізвище та ініціали)*

РЕФЕРАТ

Дипломна робота на тему «Розробка соціально-гуманітарної веб-платформи для здійснення волонтерської діяльності».

Об'єкт дослідження – соціально-гуманітарна веб-платформа.

Предмет дослідження – веб-платформа для впровадження волонтерської діяльності в соціально-гуманітарних проектах.

Мета проекту – на основі проведенного аналізу теоретичних аспектів волонтерства та веб-технологій та концепції соціально-гуманітарної веб-платформи для волонтерства, реалізувати платформу з використанням розглянутих технологій.

Метод дослідження – комплексний аналіз літературних джерел, вивчення існуючих веб-платформ, проведення експериментальних досліджень та апробація розробленої соціально-гуманітарної веб-платформи.

Веб-платформа призначена для залучення та координації волонтерів, співпраці з організаціями та реалізації соціально-гуманітарних проектів. У роботі розглядаються питання розробки веб-платформи, що сприяє здійсненню волонтерської діяльності, визначення її функціональності, реалізація технічних аспектів та тестування отриманих результатів. Веб-платформа виконує функції залучення волонтерів, координації діяльності та співпраці з організаціями.

Програмне забезпечення реалізовано з використанням сучасних технологій та мов програмування, таких як TypeScript, React, NextJS, tRPC, Tailwind CSS, NodeJS.

У роботі досліджуються питання розробки функціональних модулів веб-платформи, що включають реєстрацію та авторизацію користувачів, створення профілів волонтерів та організацій, публікацію та пошук волонтерських проектів, спілкування між учасниками та відстеження статистики діяльності на платформі.

Веб-платформа розроблена з урахуванням принципів зручності користування, надійності та масштабованості. Також враховані питання безпеки та конфіденційності даних користувачів.

Результати проекту демонструють успішну реалізацію веб-платформи, яка сприяє залученню та координації волонтерів та організацій для реалізації соціально-гуманітарних проектів, а також створює сприятливі умови для розвитку волонтерської діяльності в цілому.

Ключові слова: ВОЛОНТЕРСЬКА ДІЯЛЬНІСТЬ, СОЦІАЛЬНО-ГУМАНІТАРНІ ПРОЕКТИ, ВЕБ-ПЛАТФОРМА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, КООРДИНАЦІЯ, БЕЗПЕКА ДАНИХ, ЕФЕКТИВНІСТЬ, РОЗРОБКА, TYPESCRIPT, REACT, NEXTJS.

ЗМІСТ

ВСТУП

У сучасному світі, що характеризується стрімким технологічним розвитком та глобалізацією, соціально-гуманітарна діяльність та волонтерство займають важливе місце. Волонтерство є одним із найефективніших способів допомоги тим, хто перебуває у складних життєвих ситуаціях та потребує підтримки. Проте виклики та потреби цієї сфери постійно змінюються та вимагають застосування нових підходів та технологій.

Волонтерство має величезний потенціал для розвитку та впровадження інноваційних рішень. Зі збільшенням доступу до Інтернету та розширенням використання цифрових технологій веб-платформи стають ефективним інструментом для залучення, координації та співпраці волонтерів усього світу. Застосування веб-платформ для волонтерства дозволяє полегшити процес реєстрації волонтерів, пошуку та вибору проектів, обміну інформацією та координації діяльності.

Один із важливих аспектів волонтерства полягає у залученні волонтерів та координації їх діяльності. Метою цієї платформи є покращення доступності, координації та ефективності волонтерських проектів, створення зручних інструментів для спілкування, обміну досвідом та спільної роботи волонтерів.

Предметом дослідження є створення веб-платформи, яка дозволяє залучати та координувати волонтерів для реалізації соціально-гуманітарних проектів. Ця платформа надасть зручний інтерфейс для реєстрації волонтерів, створення та пошуку проектів, спілкування та обміну інформацією. Вона сприятиме підвищенню активності волонтерів, забезпечуватиме систему оцінки результатів та мотивації для подальшого залучення.

Дослідження в цій галузі не нове, проте більшість наукових праць та практичних досліджень зосереджені на загальних аспектах волонтерства, механізмах його функціонування та ролі у суспільстві. Водночас веб-

платформи для волонтерства стають дедалі популярнішими, проте досліджень, присвячених їх розробці та впровадженню, відносно мало.

Основна мета дослідження – розробка соціально-гуманітарної веб-платформи для впровадження волонтерської діяльності з метою покращення організації та координації волонтерських проектів. Для досягнення цієї мети поставлено такі завдання:

1. Аналіз потреб та вимог волонтерів: у рамках дослідження буде проведено аналіз потреб різних груп волонтерів, їх вимог та очікувань від волонтерської діяльності. Це допоможе врахувати різноманітність потреб та створити зручні умови для реєстрації та участі волонтерів на платформі.
2. Проектування та розробка веб-платформи: на основі отриманих даних та вимог волонтерів буде розроблено веб-платформу, яка включає необхідні функції для реєстрації волонтерів, створення та пошуку проектів, спілкування між учасниками та обміну необхідною інформацією.
3. Реалізація системи оцінки та мотивації волонтерів: у рамках платформи буде створено систему оцінки результатів волонтерської діяльності та мотивації учасників. Це дозволить визнавати зусилля волонтерів, стимулювати їх до подальшої активності та підвищувати якість виконаної роботи.
4. Забезпечення безпеки та конфіденційності даних: особиста інформація волонтерів та інших учасників платформи зберігатиметься у захищеному середовищі, забезпечуючи конфіденційність та унеможливлюючи несанкціонований доступ до даних.
5. Апробація розробленої веб-платформи та аналіз її практичного значення: розроблена веб-платформа буде випробувана на реальних волонтерських проектах для оцінки її ефективності та практичного значення. Зібрані дані та відгуки волонтерів допоможуть удосконалити платформу та внести необхідні зміни.

Дослідження ґрунтуватиметься на аналізі наукових статей, публікацій, практичного досвіду та кращих практик у галузі волонтерства та розробки веб-платформ. Додатковими джерелами інформації будуть веб-ресурси, офіційні документи та дослідницькі звіти. Для створення веб-платформи використовуватимуться сучасні методи та інструменти веб-розробки, а також емпіричні дослідження для перевірки ефективності та функціональності платформи.

Отримані результати дослідження та розробки соціально-гуманітарної веб-платформи матимуть практичне значення для різних сфер діяльності. Використання цієї платформи дозволить покращити організацію та координацію волонтерських проектів, залучити більше людей до волонтерської діяльності та підвищити мотивацію. Результати дослідження можуть бути апробовані та оцінені на реальних волонтерських проектах.

РОЗДІЛ 1. Теоретичні аспекти волонтерства та веб-технологій

1.1. Визначення волонтерства та його значення для суспільства

Волонтерство – один із найважливіших аспектів сучасного суспільства. Це добровільна, безоплатна діяльність, яку особа або група осіб здійснює на благо суспільства, спрямована на підтримку та допомогу іншим людям, організаціям чи екосистемам. Волонтерство відіграє важливу роль у формуванні демократичних цінностей, культурного обміну, соціальної взаємодопомоги та зміцненні міжнародного співробітництва. Без волонтерів багато соціальних, культурних та екологічних проектів просто не могли б існувати.

Вона може набувати різних форм, включаючи індивідуальну допомогу, участь у групових проектах або підтримку неприбутковим організаціям. Волонтери можуть працювати у різних галузях, таких як освіта, охорона здоров'я, культура, захист довкілля, соціальна робота, спорт та інші.

Благодійність має велике значення для суспільства, оскільки сприяє розвитку активної громадянської позиції, забезпечує соціальну та економічну інтеграцію, підвищує рівень освіти та культури населення. Волонтерство також допомагає розвивати навички лідерства, комунікації, співробітництва та вирішення проблем, корисних у професійному житті.

Однією з ключових причин, чому волонтерство має велике значення для суспільства, є те, що воно допомагає зміцнювати спільноти та сприяти їхньому розвитку. Участь громадян у волонтерській діяльності сприяє зміцненню соціальних зв'язків, підтримці місцевих ініціатив та розвитку соціального капіталу. Це також може забезпечити розвиток мережі підтримки для осіб, які стикаються зі складними життєвими обставинами, такими як бездомність, хвороба або соціальне відчуження.

Також він є важливим джерелом інновацій та створення нових рішень для вирішення глобальних проблем. Наприклад, волонтери можуть брати

участь у розробці нових технологій, дослідженнях чи проектах, що сприяють сталому розвитку та боротьбі зі зміною клімату. Волонтерська діяльність також може допомогти виявити нові можливості для підприємництва та створення робочих місць.

Волонтерство сприяє особистісному розвитку та самоосвіті. Участь у волонтерських проектах допомагає людям розвивати нові навички, набувати практичного досвіду та відкривати нові горизонти. Це може бути особливо корисним для молоді, яка тільки починає свій професійний шлях, а також для осіб, які прагнуть змінити свою кар'єру або вдосконалити свої професійні навички. Волонтерство може вплинути на вирішення проблем, що стосуються суспільства в цілому, а також на поліпшення життя окремих осіб. Волонтери, працюючи разом, можуть змінити життя людей, розширити можливості, надати освіту та забезпечити матеріальну допомогу нужденним. Волонтерство може допомогти у формуванні мережі допомоги та підтримки для людей, які переживають кризу або страждають на хронічні проблеми.

Крім того, воно може допомогти у зміцненні міжнародного співробітництва та сприяти культурному обміну. Через волонтерську діяльність люди різних країн та культур можуть взаємодіяти, обмінюватися ідеями та знаннями, а також працювати разом над вирішенням спільних проблем.

Волонтерство може сприяти забезпеченню рівних можливостей для всіх учасників товариства. Це може допомогти забезпечити рівний доступ до ресурсів, освіти та підтримки для маргінальних груп населення, таких як люди з інвалідністю, незаможні та представники різних етнічних груп. Волонтерська робота може допомогти зміцнити соціальну згуртованість та покращити ставлення до різних груп населення тощо.

1.2. Сучасні тенденції волонтерства

Останнім часом волонтерство пройшло кілька значних змін, включаючи зростання віртуального волонтерства, корпоративного волонтерства,

міжнародного співробітництва, використання нових технологій та зосередження на екологічних та соціальних питаннях.

1. Віртуальне волонтерство: Завдяки розвитку інтернету та соціальних медіа, віртуальне волонтерство стає все більш поширеним. Це дозволяє людям залучатися до добровільної діяльності незалежно від їхнього географічного розташування та фізичної доступності. Віртуальні волонтери можуть займатися різноманітними завданнями, такими як розробка веб-сайтів, дизайн, написання статей, консультування, переклади та багато іншого. Це може бути особливо корисним для організацій з обмеженими ресурсами або для тих, хто прагне досягти глобальної аудиторії.
2. Корпоративне волонтерство Багато компаній починають розуміти важливість корпоративної соціальної відповідальності та підтримки волонтерства серед своїх співробітників. Корпоративне волонтерство може включати різні види діяльності, такі як участь у місцевих проектах, надання фінансової підтримки благодійним організаціям, надання працівників для допомоги у виконанні завдань або організація спільних заходів. Така участь може вплинути на репутацію компанії, мотивацію та задоволеність співробітників.
3. Міжнародне співробітництво: Волонтерство стає все більш глобалізованим, з волонтерами, які беруть участь у міжнародних проектах та програмах. Це може включати культурний обмін, співробітництво у сфері освіти та наукових досліджень, а також спільну роботу з розвитку стійких співтовариств. Міжнародне волонтерство може допомогти волонтерам розширити свій світогляд, зрозуміти інші культури та набути нових навичок. Разом з тим, місцеві спільноти отримують користь від знань та досвіду волонтерів, що сприяє їхньому розвитку.

4. Використання нових технологій: Волонтери та волонтерські організації все більше використовують нові технології для підтримки своєї діяльності. Це може включати використання мобільних додатків, соціальних медіа, віртуальної реальності та інших інноваційних інструментів для координації волонтерських заходів, навчання та обміну інформацією. Технології також можуть допомогти волонтерам більш ефективно залучатися до різних проектів та вирішувати глобальні проблеми, такі як зміна клімату, бідність та сталий розвиток.
5. Зосередженість на екологічних та соціальних питаннях: Сучасні волонтери все більше працюють над екологічними та соціальними проблемами, такими як збереження природних ресурсів, захист дикої природи, забезпечення постійних міст та співробітництво з маргінальними групами населення. Волонтерство може сприяти підвищенню екологічної свідомості, зміцненню соціальної згуртованості та розвитку екологічно стійких рішень. Волонтери можуть брати участь у різних проектах, таких як організація збору сміття, відновлення природних середовищ, проведення екологічних освітніх програм та взаємодія з місцевими спільнотами для розвитку екологічних ініціатив.
6. Фокус на освіті та розвитку навичок: Волонтерство може сприяти розвитку освіти та підвищенню рівня навичок серед учасників. Це може включати участь у програмах наставництва, надання допомоги у викладанні, проведення тренінгів та семінарів для різного віку та спеціалізацій. Освітні проекти можуть бути особливо корисними для молоді, людей з інвалідністю, незаможних та маргінальних груп населення. Волонтерство у сфері освіти може допомогти забезпечити рівний доступ до якісної освіти та розвитку потенціалу кожної людини.
7. Співпраця між волонтерськими організаціями та урядовими структурами: У сучасному світі волонтерство все більше співпрацює з урядовими організаціями та місцевою владою для досягнення спільних

цілей та вирішення суспільних проблем. Це може включати спільні проекти, обмін ресурсами, навчання та інші види співробітництва. Такі партнерства можуть посилити вплив волонтерських організацій та допомогти у координації різних ініціатив на місцевому, національному та глобальному рівнях. Співпраця з урядовими структурами може також сприяти розвитку ефективних стратегій та політик у сфері волонтерства та стимулювання зростання соціальної відповідальності.

8. Забезпечення стійких волонтерських програм: Щоб волонтерські програми були успішними та продуктивними, вони мають бути стійкими та забезпечувати волонтерам необхідні ресурси, підтримку та навчання. Організації, які працюють з волонтерами, мають забезпечити належне керівництво, оцінку та зворотний зв'язок своїм волонтерам. Крім того, важливо створити середовище, яке заохочує волонтерів до тривалої участі та надає їм можливості для особистого та професійного розвитку.

Враховуючи вищезазначені тенденції, стає очевидним, що сучасне волонтерство перетворюється та адаптується до нових викликів та можливостей. Волонтерство відіграє все більш важливу роль у суспільстві, забезпечуючи підтримку у рішенні глобальних проблем, розвитку сталих спільнот та реалізації цілей сталого розвитку. Надалі важливо розвивати волонтерство, сприяючи його інноваціям, співпраці та забезпечуючи сталий розвиток волонтерських програм.

1.3. Огляд та аналіз існуючих соціально-гуманітарних веб-платформ для волонтерства.

При аналізі існуючих соціально-гуманітарних веб-платформ для волонтерства важливо розглянути декілька критеріїв, що визначають їхню ефективність та зручність для користувачів. Розглянемо такі критерії:

1. Доступність та інклюзивність: Веб-платформа має бути доступна всім потенційним користувачам, незалежно від їх технічних можливостей, мовних знань та соціально-економічного статусу. Інклюзивність

передбачає створення зручного інтерфейсу, а також забезпечення доступу до ресурсів та інформації для різних груп населення.

2. Функціональність: Веб-платформа повинна мати широкий спектр функцій, які допомагають волонтерам та організаціям ефективно співпрацювати. Це може включати можливість створення профілів, пошук волонтерських можливостей за різними критеріями, реєстрацію на заходи, спілкування та обмін документами між учасниками, відстеження виконаних завдань та аналіз результатів.
3. Безпека та конфіденційність: Веб-платформа повинна гарантувати безпеку користувачів, їх особистих даних та інформації, що обмінюється на платформі. Це передбачає використання сучасних технологій шифрування, авторизації та аутентифікації, а також встановлення правил та процедур для обробки та зберігання даних.
4. Інтеграція з іншими сервісами та платформами: Веб-платформа для волонтерства має підтримувати інтеграцію з іншими сервісами та платформами, що спрощує процес співробітництва та координації діяльності між учасниками. Це може включати інтеграцію з соціальними мережами, календарями, електронною поштою, месенджерами та іншими інструментами, що використовуються волонтерами та організаціями для забезпечення ефективної комунікації та координації.
5. Масштабованість та гнучкість: Веб-платформа повинна мати можливість масштабуватися та пристосовуватися до змін потреб та очікувань користувачів. Це означає, що платформа має бути гнучкою, а її архітектура та код мають бути організовані таким чином, щоб можна було легко додавати нові функції та вдосконалити існуючі.
6. Підтримка спільноти та розвиток: Веб-платформа повинна активно підтримувати та розвивати співтовариство волонтерів та організацій, що співпрацюють на платформі. Це може включати різні навчальні матеріали, вебінари, конференції та інші заходи, спрямовані на

поширення знань і досвіду, а також стимулювання активності та залучення учасників.

Можна зазначити такі популярні платформи:

1. VolunteerMatch: Це одна з найбільших міжнародних платформ для волонтерства, яка забезпечує широкий спектр можливостей для волонтерів та організацій у різних сферах. VolunteerMatch пропонує зручний пошук за інтересами, географією та іншими критеріями, а також надає інструменти для співпраці та обміну досвідом між учасниками.
2. Idealist: Ця платформа фокусується на забезпеченні волонтерських можливостей у сфері соціального змін та некомерційних організацій. Idealist пропонує розширений пошук волонтерських можливостей, а також інструменти для моніторингу та відстеження результатів волонтерської діяльності.
3. HandsOn Network: Ця платформа є частиною Points of Light, однієї з найбільших волонтерських організацій у світі. HandsOn Network пропонує волонтерам широкий вибір проектів та ініціатив, а також надає організаціям інструменти для планування та організації волонтерських заходів.
4. Catchafire: Ця платформа спеціалізується на забезпеченні волонтерських можливостей для професіоналів, які хотіли би використовувати свої навички та досвід для допомоги некомерційним організаціям. Catchafire забезпечує зручний пошук волонтерських проектів за професійними навичками, галузевою спеціалізацією та іншими критеріями.
5. Волонтерська Платформа: Ця платформа призначена для швидкого пошуку гуманітарної та волонтерської допомоги в усіх областях України.

Аналізуючи ці платформи за вищезазначеними критеріями, можна зробити висновки про їх сильні та слабкі сторони, а також визначити можливі напрями для подальшого розвитку та удосконалення.

Відзначимо сильні сторони цих платформ:

1. Широкий вибір волонтерських можливостей, який дозволяє користувачам знайти проекти, що відповідають їхнім інтересам та навичкам.
2. Зручні інструменти для пошуку та фільтрації волонтерських можливостей за різними критеріями.
3. Інтеграція з соціальними мережами та іншими сервісами, яка спрощує процес реєстрації та спілкування між користувачами.
4. Надання підтримки та розвитку волонтерської спільноти через навчальні матеріали, заходи та спілкування.

Проте, є деякі слабкі сторони, які можуть бути відмічені в цих платформах:

1. Застарілий дизайн у деяких платформ, і як наслідок недостатньо зручний інтерфейс
2. Недостатня залученість користувачів, більшість з них
3. Недостатня масштабованість та гнучкість платформ, що може ускладнити адаптацію до змін у потребах користувачів.
4. Недостатня увага до безпеки та приватності користувачів, що може стати перешкодою для використання платформи.

На основі проведеного аналізу можна визначити такі напрями для подальшого розвитку та удосконалення веб-платформ для волонтерства:

1. Розробка гнучких та масштабованих рішень, що дозволять платформам легко адаптуватися до змін у вимогах та очікуваннях користувачів.
2. Забезпечення високого рівня безпеки та приватності користувачів, використовуючи сучасні технології шифрування, авторизації та

аутентифікації, а також розробляючи правила та процедури для обробки та зберігання даних.

3. Оновлення дизайну платформи, оскільки застарілий дизайн може вплинути на зручність користувачів та загальне сприйняття платформи. Сучасний, інтуїтивний дизайн забезпечить кращий користувацький досвід та залученість користувачів.
4. Розширення можливостей для інтеграції з іншими сервісами та платформами, що полегшить процес співпраці та координації між учасниками та допоможе під'єднати додаткові ресурси та інструменти.

Враховуючи вищезазначені напрями для подальшого розвитку та удосконалення веб-платформ для волонтерства, можна сприяти створенню більш ефективних та зручних сервісів, які відповідатимуть потребам та очікуванням користувачів та сприятимуть розвитку волонтерської діяльності у соціально-гуманітарній сфері.

РОЗДІЛ 2. Розробка концепції соціально-гуманітарної веб-платформи для здійснення волонтерської діяльності

2.1. Визначення основних функцій та вимог до платформи

Для створення ефективної соціально-гуманітарної веб-платформи для волонтерства насамперед необхідно визначити ключові функції та вимоги до платформи, які відповідають потребам користувачів. Це допоможе забезпечити простоту використання, доступність, а також високу корисність для організацій та волонтерів. Основні функції та вимоги до платформи можна поділити на такі категорії:

Реєстрація та профілі користувачів:

- Проста та швидка реєстрація з використанням електронної пошти або соціальних мереж;
- Створення детальних профілів для волонтерів та організацій, що відображають, навички, інтереси та контактну інформацію;

Пошук та фільтрація волонтерських можливостей:

- Пошук та фільтрація волонтерських проектів за різними параметрами: географічне розташування, сфера діяльності, тривалість, тип волонтерства (очно, віртуально) тощо;
- Інтеграція з календарем та іншими інструментами для планування волонтерської діяльності.

Співпраця та комунікація:

- Інструменти для безпосереднього зв'язку між волонтерами та організаціями, такі як повідомлення, та коментарі.
- Функція створення спільнот та груп для обговорення проектів, обміну досвідом та координації дій;

- Можливість надсилати заявки на участь у волонтерських проектах, переглядати статус заявок та отримувати зворотний зв'язок від організацій.

Зворотний зв'язок та відзнаки:

- Механізм отримання зворотного зв'язку від волонтерів та організацій щодо проектів, співпраці та платформи в цілому;
- Система відзнак та досягнень для мотивації волонтерів та визнання їхнього внеску в проекти;
- Рейтингова система для оцінки волонтерів та організацій з метою підвищення довіри та прозорості.

Враховуючи вищезазначені функції та вимоги, можна розробити веб-платформу, яка забезпечить ефективну взаємодію між волонтерами та організаціями, сприятиме залученню нових учасників та підтримці волонтерської діяльності. Забезпечення високої якості реалізації таких функцій та вимог вимагає аналізу технічних можливостей, вибору відповідних технологій та інструментів, а також ретельного планування розробки та впровадження платформи.

Окрім того, важливо забезпечити стабільність роботи платформи, її масштабованість та безпеку користувачів. Це включає захист від несанкціонованого доступу, шифрування даних, резервне копіювання та відновлення інформації в разі аварій або втрати даних.

Також, необхідно врахувати інтеграцію з іншими сервісами та платформами, які вже використовуються волонтерами та організаціями. Інтеграція може включати сумісність з календарями, соціальними мережами, системами електронної пошти, інструментами співпраці та комунікації тощо.

Враховуючи потреби користувачів та сучасні тенденції веб-технологій, розробка соціально-гуманітарної веб-платформи для волонтерства повинна бути здійснена з урахуванням принципів користувацького досвіду, адаптивного

дизайну та мобільної сумісності. Це дозволить забезпечити широке охоплення користувачів, зручність та задоволення від використання платформи, а також сприятиме розвитку волонтерства та підтримці соціально-гуманітарних проектів.

2.2. Проектування архітектури та структури платформи

Після визначення основних функцій та вимог до платформи та аналізу потреб користувачів, наступним етапом є проектування архітектури та структури веб-платформи. Це передбачає розробку загальної організації даних, інтерфейсів, сервісів та компонентів, які будуть взаємодіяти між собою для забезпечення функціональності платформи.

1. Архітектура платформи (рис 2.1):

- Вибір технологій та фреймворків для реалізації клієнтської та серверної частини платформи (React, Next.js, Next-Auth, Tailwind CSS, Prisma, tRPC);
- Розробка моделі взаємодії між серверною та клієнтською частинами, з використанням tRPC;
- Визначення стратегії зберігання та обробки даних, вибір системи керування базами даних (наприклад, PostgreSQL) та розробка схеми бази даних завдяки Prisma.

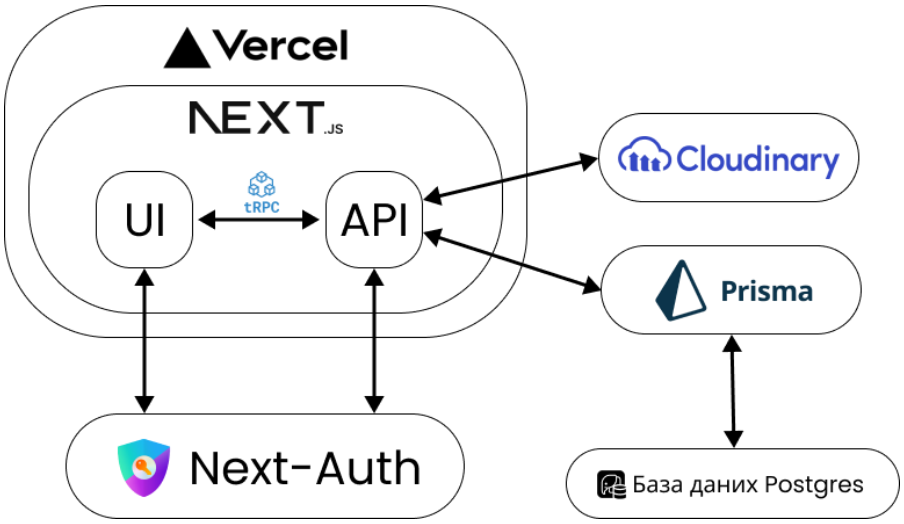

Рисунок 2.1 Візуалізація архітектури платформи

2. Структура платформи:

- Розробка дерева категорій та ієрархії веб-сайту, включаючи головну сторінку, розділи для волонтерів, організацій, проектів, ресурсів тощо;
- Розробка інтерфейсів та сторінок для кожної з основних функцій платформи, таких як реєстрація, авторизація, створення проектів, спілкування між користувачами, пошук та фільтрація волонтерських можливостей;
- Розробка адаптивного дизайну для забезпечення коректного відображення та взаємодії на різних пристроях (комп'ютери, планшети, смартфони).

3. Безпека та доступність:

- Розробка стратегій захисту даних, включаючи шифрування, резервне копіювання та відновлення даних, а також впровадження політик доступу до інформації;
- Забезпечення безпечної авторизації та аутентифікації користувачів, OAuth 2.0 та інших сучасних стандартів безпеки;
- Впровадження стратегій моніторингу та виявлення можливих атак, шахрайства та інших загроз для стабільності та безпеки платформи;
- Оптимізація продуктивності та доступності платформи, забезпечення масштабування та стабільності сервісів у разі збільшення кількості користувачів та навантаження на систему.

4. Впровадження:

- Розгортання та налаштування серверного середовища для запуску платформи.
- Забезпечення неперервного вдосконалення платформи, враховуючи відгуки користувачів, зміни у технологічних стандартах та нові можливості розвитку соціально-гуманітарного волонтерства.

Результатом цього етапу є візуалізація архітектури та структури платформи, що відображає взаємозв'язок між основними компонентами та функціональними можливостями. Надалі це полегшить розробку та реалізацію проекту, а також сприятиме належному управлінню ресурсами та забезпеченню стабільності, продуктивності та безпеки платформи.

5. Оцінка та звітність:

- Розробка системи збору та аналізу даних про використання платформи, оцінка ефективності волонтерських проектів, рівень задоволеності користувачів та вплив на соціально-гуманітарну сферу;
- Впровадження інструментів для відстеження результатів роботи платформи, таких як аналітика відвідувань, статистика взаємодії користувачів, звіти про виконання волонтерських проектів та оцінка впливу на соціум;
- Розробка механізмів звітності та прозорості для демонстрації успіхів платформи, забезпечення взаємодії з громадськістю, отримання зворотнього зв'язку та пропозицій щодо подальшого розвитку платформи.

У результаті проектування архітектури та структури платформи, команда розробників отримує чітке розуміння задач, етапів розробки та планування ресурсів. Це дає можливість забезпечити своєчасну реалізацію проекту та забезпечити високу якість кінцевого продукту, який буде відповідати потребам користувачів та забезпечувати ефективну взаємодію у соціально-гуманітарному волонтерстві.

РОЗДІЛ 3. Реалізація веб-платформи для волонтерства

Нині виникає дедалі більше технологій, що сприяють розвитку волонтерства. Проте, їхнє ефективне застосування вимагає створення спеціалізованих програмних рішень. У цьому контексті одним із ключових завдань є розробка веб-платформ, здатних ефективно поєднувати волонтерів та організації для спільної роботи над спільними цілями. Таке завдання передбачає включення всього циклу розробки веб-додатку: аналізу, проектування, розробки і тестування.

Усі ці аспекти будуть розглянуті у процесі створення веб-платформи для волонтерства. Базовим елементом такого додатка є база даних, вона відіграє важливу роль у зберіганні та обробці інформації про волонтерів, організації, події та інші аспекти функціонування платформи. Розробка цієї частини програми буде здійснена за допомогою Prisma - інструмента для роботи з базами даних у TypeScript та Node.js.

Доповнює базу даних серверну частину програми, яка відповідає за логіку взаємодії з базою даних та обробку запитів від користувачів. Для її розробки буде використаний Next.js, фреймворк, що забезпечує гнучкість та масштабованість серверного рішення.

Наступний крок - це розробка клієнтської частини, що є інтерфейсом додатка, з яким взаємодіють користувачі. Щоб зробити його привабливим і зручним для використання, в роботі буде використовуватися React - одна з найпопулярніших бібліотек для створення інтерфейсів користувача, а також Tailwind CSS - інструмент для швидкого створення сучасних дизайнів.

Безпека даних - це ще один критичний аспект, який буде забезпечений за допомогою Next-Auth. Ця бібліотека для Next.js дозволяє легко та безпечно реалізувати системи автентифікації та авторизації.

Основні функціональні можливості платформи, включаючи пошук волонтерів та організацій, участь у заходах, управління профілем, також будуть

уважно розроблені та ретельно протестовані. Така робота забезпечить зручність та надійність використання платформи.

Загалом, створення веб-платформи для волонтерства - це складне завдання, яке потребує комплексного підходу та обліку безлічі аспектів. Результат цієї праці – сучасна та надійна платформа, яка здатна покращити взаємодію між волонтерами та організаціями, а також зробити волонтерство більш доступним та ефективним.

3.1. Теоретичні відомості технологій

3.1.1. TypeScript

TypeScript - це суперсет JavaScript, що додає статичні типи до мови програмування. Використання TypeScript сприяє створенню більш надійних, стабільних та безпечних програм. Він полегшує розробку та підтримку коду завдяки системі типів, що дозволяє виявляти помилки на стадії компіляції, робити код більш читабельним та передбачуваним, полегшує рефакторинг. TypeScript інтегрується з різними інструментами розробки, що полегшує створення веб-додатків.

TypeScript додає ряд функцій та можливостей до JavaScript, роблячи його більш безпечним та допомагаючи розробникам писати чистіший та легко підтримуваний код. Ось деякі з основних функцій TypeScript:

1. **Статична типізація**: TypeScript дозволяє вказувати типи даних для змінних, функцій, класів, інтерфейсів тощо. Це допомагає підвищити безпеку коду та забезпечити його стабільність, виявляти та виправляти помилки на стадії розробки. Типи даних в TypeScript можуть бути примітивними, такими як `string`, `number`, `boolean`, `null`, `undefined`, а також складеними, такими як масиви, об'єкти, кортежі, перелічення та інші.
2. **Інтерфейси**: TypeScript дозволяє використовувати інтерфейси для визначення контрактів (набору властивостей та методів), які повинні реалізувати класи або об'єкти. Інтерфейси допомагають розробникам

створювати гнучкі та модульні програми, спрощуючи процес розробки та рефакторингу коду.

3. **Дженерики** (Generics): TypeScript дозволяє використовувати дженерики, що допомагають створювати гнучкі компоненти для повторного використання, які можуть працювати з різними типами даних. Застосування дженериків дозволяє забезпечити безпечність типів без зменшення продуктивності.
4. **Декоратори**: TypeScript включає декоратори, які дозволяють розробникам додавати спеціальні метадані або змінювати поведінку класів, методів, властивостей та параметрів. Декоратори полегшують розширення функціональності програми без внесення змін до її основного коду.
5. **Модульна система**: TypeScript підтримує систему модулів, що дозволяє розробникам організовувати код у окремі модулі, які можуть бути імпортовані та експортовані між різними частинами проекту. Модульна система сприяє створенню чистого, модульного та легко підтримуваного коду.
6. **Типи об'єднання** (Union Types) **та перетину** (Intersection Types): TypeScript дозволяє використовувати типи об'єднання та перетину, що дозволяє розробникам комбінувати та створювати нові типи з використанням вже існуючих. Це забезпечує гнучкість та можливість адаптації до різних сценаріїв.
7. **Опціональні та обов'язкові властивості**: TypeScript дозволяє вказувати, які властивості об'єкта є обов'язковими, а які можуть бути пропущені. Це поліпшує безпеку коду та спрощує розробку, оскільки розробники можуть бути впевнені в тому, які властивості об'єкта повинні бути завжди доступні.
8. **Мапи типів** (Mapped Types): TypeScript дозволяє створювати мапи типів, які можуть трансформувати один тип в інший на основі заданих правил.

Мапи типів можуть допомогти розробникам автоматизувати та спростити процеси рефакторингу та створення нових типів.

9. **Умовні типи** (Conditional Types): TypeScript включає умовні типи, які дозволяють створювати нові типи на основі умов, що залежать від інших типів. Умовні типи допомагають розробникам створювати гнучкі та ефективні програми, адаптуючи типи до різних сценаріїв.

Ці функції та можливості TypeScript допомагають розробникам створювати масштабовані, безпечні та легко підтримувані веб-додатки. Завдяки статичній типізації та іншим вдосконаленням, TypeScript може значно покращити якість коду, спрощуючи процес розробки та рефакторингу.

3.1.2. React

React - це популярна JavaScript-бібліотека, розроблена командою Facebook, для створення інтерфейсів користувача [1]. React відрізняється від інших бібліотек та фреймворків своїм декларативним підходом до розробки, який спрощує роботу з динамічними інтерфейсами користувача. Основні особливості та переваги React включають:

- **Компонентний підхід:** React дозволяє розробникам створювати веб-додатки, розбиваючи інтерфейс користувача на окремі, повторно використовувані компоненти. Кожен компонент відповідає за певний елемент інтерфейсу користувача та має власний стан та логіку рендеринга.
- **Віртуальний DOM:** React використовує віртуальний DOM (об'єктна модель документа), що дозволяє ефективно відстежувати зміни в інтерфейсі користувача та оптимізувати рендеринг. Коли стан компонента змінюється, React спочатку оновлює віртуальний DOM, а потім вносить мінімально можливі зміни в реальний DOM, що прискорює процес рендеринга (рис. 3.1).
- **Однонаправлені дані:** React використовує однонаправлені дані, які спрощують управління станом та забезпечують більш прозорий потік

даних в додатку. Зміни в стані компонента просуваються вниз по дереву компонентів, забезпечуючи неперервність даних та спрощуючи логіку стану.

- **JSX:** React використовує JSX - синтаксичне розширення для JavaScript, що дозволяє змішувати HTML-розмітку з JavaScript-кодом. JSX забезпечує зручний та інтуїтивно зрозумілий спосіб створення веб-інтерфейсів, оскільки розмітка виглядає подібно до звичайного HTML. JSX перетворюється на звичайний JavaScript-код під час компіляції.
- **Контрольовані компоненти:** React пропонує підхід до роботи з формами, заснований на контрольованих компонентах. В контрольованих компонентах стан введення користувача зберігається в стані компонента та оновлюється за допомогою обробника подій. Це дозволяє розробникам легко відстежувати та маніпулювати даними форми, надаючи більше контролю над процесом введення.
- **Життєвий цикл компонентів:** React надає набір методів, з якими можна працювати під час життєвого циклу компонента, включаючи моменти створення, оновлення та знищення компонента. Ці методи дозволяють розробникам керувати ресурсами, запускати асинхронні запити та реагувати на зміни стану чи властивостей компонента.
- **Хуки:** React вводить хуки, які дозволяють використовувати стан та інші можливості React у функціональних компонентах, спрощуючи створення та використання компонентів. За допомогою хуків можна розділяти логіку компонента на менші та повторно використовувані функції. Найбільш поширені хуки - це useState, useEffect та useContext.

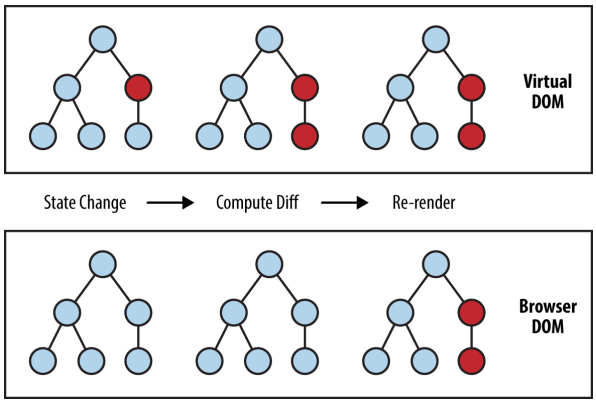

Рисунок 3.1 Віртуальний React DOM

3.1.3. NextJS

Next.js - це фреймворк для розробки серверно-рендерених React-додатків. Next.js дає змогу створювати веб-програми з повним стеком(full - stack), розширюючи найновіші функції React та інтегруючи потужні інструменти JavaScript на основі Rust для найшвидшої збірки [2]. Основні особливості Next.js включають:

- **Автоматичний серверний рендеринг:** Next.js автоматично обробляє серверний рендеринг для React-компонентів, що покращує продуктивність та SEO. Це також спрощує розробку, оскільки розробники можуть працювати з одним кодом для клієнта та сервера.
- **Статична генерація сторінок:** Next.js підтримує статичну генерацію сторінок, що дозволяє створювати сторінки, які не потребують серверного рендеринга. Це може покращити продуктивність та надійність веб-додатків, зменшуючи навантаження на сервер.
- **Динамічний імпорт:** Next.js дозволяє динамічно імпортувати компоненти або модулі, що забезпечує оптимізацію розміру пакетів та продуктивності додатку. За допомогою динамічного імпорту можна відкладати завантаження компонентів до моменту, коли вони дійсно потрібні.
- **API-маршрути:** Next.js надає підтримку API-маршрутів, що дозволяє створювати API-кінцеві точки безпосередньо в рамках фреймворку.

Завдяки цьому можна легко створювати веб-додатки, які мають власний API для обробки даних та взаємодії з сервером.

- **Вбудована оптимізація:** Next.js включає вбудований оптимізатор зображень, шрифтів, та скриптів, який автоматично забезпечує оптимальну компресію для різних пристроїв та типів підключень. Це покращує продуктивність та користувацький досвід, оскільки зображення завантажуються швидше та споживають менше трафіку.
- **Інтернаціоналізація:** Next.js підтримує вбудовану інтернаціоналізацію, що дозволяє створювати багатомовні веб-додатки з локалізованим контентом та маршрутами. Це забезпечує підвищення доступності та глобальної аудиторії.
- **Підтримка TypeScript:** Next.js підтримує TypeScript з коробки, що спрощує процес впровадження типів та розробки за допомогою TypeScript в React-додатках. Це забезпечує кращу якість коду, продуктивність розробки та підтримку.

Завдяки цим та іншим особливостям, Next.js стає відмінним вибором для розробки сучасних веб-додатків на базі React. Він спрощує процес розробки, покращує продуктивність та надає ряд корисних функцій, які можуть допомогти розробникам створювати швидкі, стабільні та масштабовані додатки.

3.1.4. Next-Auth

Next-Auth - це легка бібліотека для аутентифікації в Next.js-додатках. Вона надає простий та гнучкий спосіб реалізації аутентифікації та авторизації з використанням різних провайдерів аутентифікації, таких як Google, Facebook, Twitter тощо [3].

Основні особливості Next-Auth:

- **Підтримка багатьох провайдерів:** Next-Auth підтримує широкий спектр провайдерів аутентифікації, таких як Google, Facebook, Twitter,

GitHub та інші. Це дозволяє розробникам легко впроваджувати аутентифікацію за допомогою найпопулярніших служб.

- **JWT аутентифікація:** Next-Auth використовує JSON Web Tokens (JWT) для забезпечення безпечної аутентифікації користувачів. JWT забезпечує простий та надійний спосіб передачі та перевірки даних аутентифікації між клієнтом та сервером.
- **Вбудовані хуки:** Next-Auth надає вбудовані хуки для роботи з аутентифікацією користувачів, такі як useSession, useSignIn, useSignOut та інші. Це дозволяє розробникам легко інтегрувати аутентифікацію в свої компоненти та логіку додатку
- **Адаптивність:** Next-Auth дозволяє легко налаштувати різні аспекти аутентифікації та авторизації, такі як налаштування провайдерів, редагування сторінок аутентифікації, додавання власної логіки аутентифікації тощо. Це робить Next-Auth гнучким рішенням для різних вимог проектів.
- **Безпека:** Next-Auth розроблено з урахуванням безпеки та приватності користувачів. Це забезпечує безпечне зберігання та передачу даних аутентифікації, а також допомагає запобігти атакам та витокам даних.

3.1.5. Prisma

Prisma - це ORM (Object-Relational Mapping) та доступ до даних для Node.js та TypeScript [4]. Він допомагає розробникам легко інтегрувати бази даних у свої проекти, пропонуючи зручний та типобезпечний API для виконання операцій з даними.

Основні переваги Prisma:

- Типобезпечність: Prisma автоматично генерує типи для вашої бази даних на основі схеми бази даних, забезпечуючи типобезпечний доступ до даних та запобігаючи помилкам під час розробки.

- Зручний синтаксис запитів: Prisma надає зручний синтаксис для створення та виконання запитів до бази даних, який легко зрозуміти та прочитати.
- Підтримка різних баз даних: Prisma підтримує різні типи баз даних, такі як PostgreSQL, MySQL, SQLite та MongoDB. Це робить його гнучким рішенням для розробників, які працюють з різними базами даних.
- Міграція бази даних: Prisma Migrate дозволяє розробникам легко створювати, застосовувати та відкатувати міграції бази даних, що забезпечує зручний спосіб керування структурою бази даних протягом часу.
- Інтеграція з Next.js: Prisma може легко інтегруватися з Next.js-додатками, що дозволяє розробникам забезпечити надійний доступ до даних та взаємодію з базами даних в рамках своїх проектів.
- Обробка проблем продуктивності: Prisma пропонує зручні інструменти для розв'язання проблем продуктивності, такі як N+1 запити. Ви можете контролювати, які дані завантажуються, та уникати непотрібних запитів до бази даних.
- Інтеграція із сучасними технологіями: Prisma відмінно інтегрується із сучасними технологіями, такими як TypeScript, GraphQL та сервери Node.js.

3.1.6. Tailwind CSS

Tailwind CSS - це утилітарний CSS-фреймворк, який дозволяє розробникам швидко та легко створювати стильові та адаптивні інтерфейси користувача [5]. Tailwind CSS працює шляхом сканування всіх HTML-файлів, компонентів JavaScript та будь-яких інших шаблонів на предмет імен класів, генеруючи відповідні стилі, а потім записуючи їх у статичний файл CSS [6]. Він пропонує набір класів, які можна використовувати для стилізації HTML-елементів без потреби в написанні власного CSS.

Основні переваги Tailwind CSS включають:

- **Швидка розробка:** Tailwind CSS дозволяє розробникам швидко створювати інтерфейси користувача, використовуючи готові класи стилів. Це зменшує час на написання та налаштування CSS, пришвидшуючи процес розробки.
- **Налаштування:** Tailwind CSS дозволяє легко налаштувати стилі, кольори, шрифти та інші елементи дизайну за допомогою конфігураційного файлу. Це дозволяє створювати консистентний та унікальний дизайн відповідно до потреб проекту.
- **Адаптивність:** Tailwind CSS підтримує адаптивні стилі за допомогою медіа-запитів, які можна легко використовувати в класах. Це дозволяє створювати інтерфейси користувача, які працюють добре на різних розмірах екранів та пристроях.
- **Спільнота та екосистема:** Tailwind CSS має велику спільноту та активно розвивається. Це означає, що розробники мають доступ до ресурсів, плагінів та підтримки, які можуть допомогти їм у роботі з фреймворком.
- **Інтеграція з React:** Tailwind CSS може легко інтегруватися з React-додатками, забезпечуючи зручний спосіб створення стильових компонентів. Це робить його ідеальним вибором для React-розробників, які шукають швидкий та легкий спосіб стилізації своїх додатків.

3.1.7 tRPC

tRPC - це набір інструментів для створення API кінцевих точок з використанням JavaScript/TypeScript. Він дозволяє будувати безпечні та ефективні API, ґрунтуючись на перевагах статичної типізації [7]. Ключові переваги використання tRPC:

- **Типобезпека**: Одна з найбільших переваг tRPC полягає в його типобезпеці. Код API та клієнтів генерується на основі одних і тих самих визначень типів, що забезпечує цілісність типів між сервером та клієнтом.

- **Підтримка TypeScript:** tRPC розроблявся з урахуванням TypeScript, тому його інтеграція з проектами TypeScript дуже гладка. Підтримка TypeScript дозволяє автоматично генерувати типи для ваших запитів та відповідей.
- **Простота використання:** tRPC намагається бути якомога простіше у використанні. Він надає простий і зрозумілий API, який дозволяє швидко та легко створювати ефективні та безпечні API.
- **Позбавлення надлишкового коду:** Оскільки tRPC генерує типи для API, вам не потрібно писати надлишковий код для перевірки типів даних або перетворення даних. Це може значно скоротити кількість коду, який вам потрібно написати та підтримувати.
- **Автоматична серіалізація та десеріалізація даних:** tRPC автоматично серіалізує та десеріалізує дані при надсиланні та отриманні запитів. Це спрощує роботу з даними та знижує ймовірність помилок.
- **Продуктивність:** tRPC було розроблено з урахуванням продуктивності. Він мінімізує кількість запитів та даних, які необхідно передати, що може призвести до покращення продуктивності.
- **Масштабованість:** tRPC дає можливість легко масштабувати програми завдяки своїй модульній архітектурі.
- **Безпека:** tRPC включає безліч функцій для забезпечення безпеки ваших API, таких як вбудована підтримка захисту від CSRF-атак.

3.2. Розробка бази даних та серверної частини з використанням Prisma, Next.js та tRPC

Для створення структури веб-платформи для волонтерства були використані Prisma, Next.js та tRPC. У даному контексті, Next.js забезпечує серверну частину програми, у той час, як Prisma є основою для створення та управління базою даних. З метою упорядкування проекту всі основні файли були згруповані в папці src.

Prisma зарекомендувала себе потужним інструментом для роботи з базою даних. Це сучасний ORM (Object-Relational Mapping) для Node.js та TypeScript. Prisma дозволяє працювати з базою даних, як із звичайними об'єктами JavaScript, приховуючи складності SQL. Її особливість полягає у потужній системі типів, інтуїтивно зрозумілому синтаксисі та ефективному контролі за виконанням запитів.

Одним з важливих етапів налаштування проекту перед створенням схеми Prisma та налаштуванням серверної частини є конфігурація змінних оточення або `env` змінних. Це змінні, які визначаються в оточенні, де виконуються додаток, і можуть бути використані для зберігання такої інформації, як секретні ключі, шляхи ресурсів та конфігураційні налаштування. Існує багато способів керування `env` змінними в Node.js та JavaScript проектах, але одним із найнадійніших та найпотужніших способів є використання бібліотеки zod. Zod являє собою бібліотеку для валідації та парсингу даних у TypeScript та JavaScript, яку можна використовувати для визначення схеми ваших змінних оточення та для перевірки того, що всі вони встановлені правильно та мають правильні значення [8].

Визначено дві схеми змінних оточення: `server` та `client`. Схема `server` включає всі змінні оточення, які мають бути доступні на сервері, включаючи URL-адреси бази даних, секретні ключі аутентифікації та інші параметри конфігурації. Схема `client` включає змінні оточення, які мають бути доступні за клієнта. Потім, використовуючи метод `safeParse` із zod, перевіряється, що всі змінні оточення встановлені та мають правильні значення. Якщо якісь змінні оточення відсутні або мають неправильні значення, то генерується помилка і програма не запуститься. Це забезпечує, що всі необхідні змінні оточення встановлені правильно перед запуском програми.

Після імпорту створеного файлу у файл next.config.mjs: !process.env.SKIP_ENV_VALIDATION && (await import("./src/env.mjs")).

Змінні оточення, які були перевірені та пройшли валідацію, експортуються з модуля, щоб вони могли бути використані в інших частинах програми. Цей підхід до управління та валідації `env` змінних забезпечує надійність та безпеку при роботі з важливими параметрами конфігурації у додатку.

Створення схеми Prisma – є наступним кроком на шляху до побудови структури бази даних. Схема Prisma є файлом конфігурації, в якому описуються моделі даних програми. Кожна модель відповідає таблиці в базі даних, а поля моделі є стовпцями цієї таблиці. Створення схеми забезпечує ясність та структуру даних, що критично важливо для забезпечення надійності та продуктивності програми. Визначення схеми даних відбувається у файлі ‘prisma/schema.prisma’. У наведеній схемі даних (рис 3.2) Prisma визначено різні моделі, які представляють таблиці у базі даних. Наприклад, моделі User, Account, Session та VerificationToken представляють користувачів, акаунти, сесії та токени верифікації відповідно, вони є обов’язковими для авторизації користувачів за допомогою Next-Auth. Кожне поле у цих моделях представляє стовпець у відповідній таблиці.

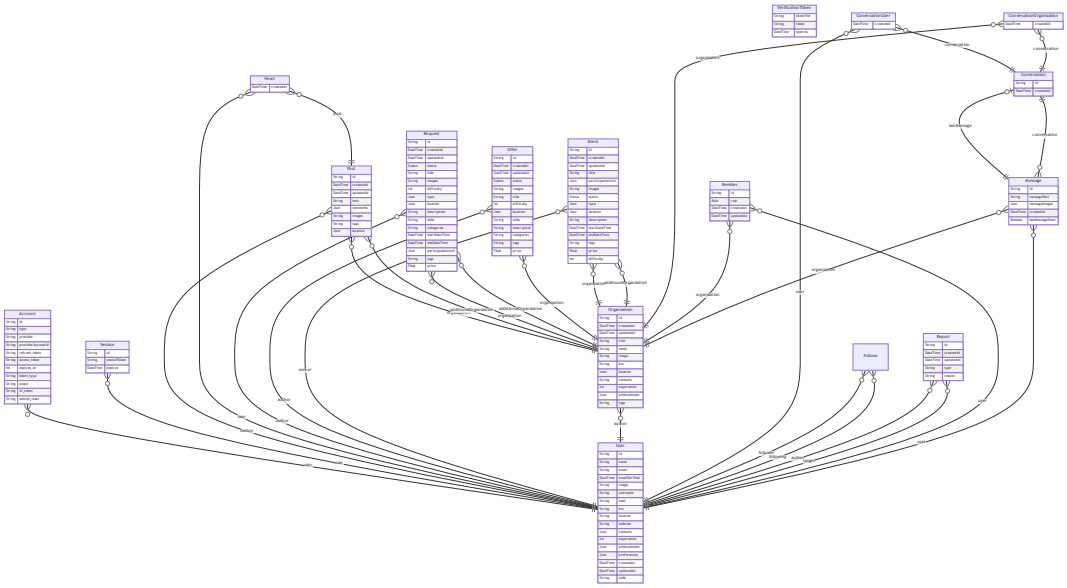

Рисунок 3.2 Візуалізація схеми даних Prisma

Візьмемо, наприклад, модель User:

```
model User {
  id            String    @id @default(cuid())
  name          String?
  email         String?   @unique
  emailVerified DateTime?
  image         String?
  accounts      Account[]
  sessions      Session[]
  username      String?   @unique
  hash          String?
  bio           String?
  location      String?
  contacts      Json?
  experience    Int       @default(10)
  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt
  followers     Follows[]         @relation("following")
  following     Follows[]         @relation("follower")
  posts         Post[]
  organizations Organization[]
  member        Member[]
  hearts        Heart[]
  requests      Request[]
```

```
  offers        Offer[]
  events        Event[]          @relation("author")
  messages      Message[]
  conversations ConversationUser[]
  reports       Report[]         @relation("author")
  reported      Report[]         @relation("target")
  participants  Event[]          @relation("participants")
  participantsRequest  Request[]        @relation("participantsRequest")
  participantsOffer  Offer[]        @relation("participantsOffer")
}
```

Тут User має поле id з атрибутами @id та @default(cuid()), що означає, що id є унікальним ідентифікатором, і його значенням є унікальний ідентифікатор, згенерований функцією cuid(). Інші поля, такі як name, email, bio, та location є типом String?, що означає, що вони можуть набувати null значення. Поле email має атрибут @unique, що означає, що всі значення у цьому стовпці мають бути унікальними.

У моделі User також визначено стосунки з іншими моделями. Наприклад, Session[] вказує, що у користувача може бути кілька пов'язаних сесій авторизації, що представляє відношення "один до багатьох" між користувачами та сесіями авторизації. Це ставлення визначається моделі Session за допомогою атрибута @relation.

Схеми даних Prisma також підтримують перерахування (enum), які представляють обмежений набір значень. Наприклад, Status є переліком, який може приймати значення PENDING, ACCEPTED, REJECTED, DONE та REWARDED.

Короткий опис кожної моделі у схемі:

- User: Подає користувача в системі. Містить різні атрибути користувача, включаючи ім'я, електронну пошту, інформацію про обліковий запис та інші пов'язані дані.
- Account: Показує обліковий запис користувача. Може бути пов'язаний з різними постачальниками послуг (наприклад, Google, Facebook), які вказані в полі provider.
- Session: Слідкує за активними сесіями користувача. Включає дані про те, коли була створена сесія і коли вона закінчується.
- VerificationToken: Використовується для зберігання токенів, які необхідні для підтвердження, таких як підтвердження електронної пошти або скидання пароля.
- Follows: Представляє відносини "підписники/підписки" між користувачами. Містить інформацію про те, хто на кого підписаний.
- Post: Представляє пости, які користувачі публікують у системі. Включає інформацію про вміст посту, його автора та інші пов'язані дані.
- Organization: Надає організації, які можуть бути пов'язані з користувачами.
- Member: Описує учасників в організаціях.
- Heart: Представляє "лайки" або "серця", які користувачі можуть ставити постам.
- Request: Описує запити, які можуть створювати користувачі. Запити можуть бути надіслані до організацій або інших користувачів.
- Offer: Подає пропозиції, які користувачі можуть робити у відповідь на запити.
- Event: Події, які можуть бути створені користувачами або організаціями.
- Message: Показує повідомлення між користувачами в рамках системи.
- Conversation: Показує розмови між користувачами, що складаються з кількох повідомлень.

- Report: Подає скарги або звіти, які користувачі можуть подавати на інших користувачів або контент.

Після створення моделі даних Prisma, наступним кроком є генерація клієнта Prisma. Клієнт Prisma – це згенерований на основі схеми Prisma код, що дозволяє взаємодіяти з базою даних. Важливо, що клієнт Prisma автоматично згенерує типи TypeScript на основі схеми даних, забезпечуючи таким чином строгу типізацію в коді та підвищуючи безпеку та надійність програми.

Для створення Prisma клієнта використовується команда “npx prisma generate”. Після її виконання згенерований код буде доступний для імпорту в код проекту, і його можна використовувати для виконання запитів до бази даних.

Після визначення моделей та генерації клієнта Prisma, починається етап налаштування серверної частини програми за допомогою Next.js та tRPC. У файлі `/src/server/db.ts` створюється екземпляр PrismaClient, який використовується для взаємодії з базою даних за допомогою Prisma. PrismaClient є автоматично згенерованим класом, який надає методи для виконання запитів до бази даних відповідно до схеми Prisma. Цей екземпляр `PrismaClient` створюється один раз при запуску програми та зберігається в глобальній змінній `globalForPrisma.prisma`, щоб можна було повторно використовувати його у всіх місцях програми. Це робиться для забезпечення оптимального використання ресурсів, оскільки кожен екземпляр PrismaClient містить пул з'єднань з базою даних.

Next.js – це універсальний фреймворк, який підтримує кілька видів рендерингу, включаючи серверний рендеринг (Server Side Rendering, SSR), статичний рендеринг (Static Site Generation, SSG) та рендеринг на стороні клієнта (Client Side Rendering, CSR). У цьому контексті ми використовуємо Next.js для налаштування API-роутів на сервері, які будуть обробляти вхідні HTTP-запити та взаємодіяти з базою даних за допомогою Prisma.

tRPC у свою чергу надає потужний шар для побудови типізованих API на основі функцій TypeScript, що значно спрощує процес розробки. Замість написання коду для обробки HTTP-запитів та відповідей розробники визначають функції, які приймають вхідні параметри та повертають результати. Ці функції потім автоматично перетворюються tRPC на API-маршрути. При використанні Next.js разом з tRPC функції, які ми визначаємо в tRPC, можуть бути безпосередньо пов'язані з API-роутами Next.js. Це забезпечує потужну, гнучку та зручну систему для створення серверної частини програми.

Створення сервера tRPC починається з визначення обробників API. Обробники API – це функції, які приймають вхідні дані, виконують деяку логіку (як правило, взаємодію з базою даних) та повертають результат. У tRPC ці обробники визначаються у вигляді методів у класі чи об'єкті.

Потім, для інтеграції з Next.js, tRPC надає спеціальні функції, такі як createNextApiHandler, які дозволяють зв'язати обробники tRPC з API-роутами Next.js.

Останнім етапом інтеграції tRPC, у проект є створення раутеру. Він відповідає за маршрутизацію запитів до відповідних обробників API, які ми визначили в попередніх кроках.

В даному випадку, ми створюємо основний раутер для додатка, котрий з'єднує декілька підраутерів. Кожен з цих підраутерів представляє собою окремий модуль або частину API, таку як 'user', 'chat', 'post' і так далі. Ця структура дозволяє логічно структурувати API і забезпечує легке масштабування додатка.

З цими раутерами, ми можемо визначити деталізовану логіку серверу для різних аспектів додатка. Наприклад, userRouter може обробляти операції, пов'язані з користувачами, такі як авторизація, отримання даних профілю та оновлення даних користувача. Тим часом chatRouter може відповідати за

роботу з повідомленнями та чатами, включаючи створення нових чатів, надсилання повідомлень та отримання історії чату.

3.3. Інтеграція аутентифікації та авторизації з використанням Next-Auth

Реалізація системи автентифікації веб-додатків — це складне та критично важливе завдання, яке вимагає дотримання правил безпеки та даних. Замість самостійного написання аутентифікації використання NextAuth.js забезпечує просту, безпечну і швидку реалізацію аутентифікації для Next.js.

NextAuth.js - це бібліотека, що полегшує інтеграцію OAuth та Email аутентифікації з Next.js додатками, при цьому керування сесіями, CSRF захистом та захистом від атак типу "Man In The Middle" здійснюється "з коробки".

Створення серверної частини аутентифікації в Next.js, використовуючи NextAuth, починається з установки пакету `next-auth`. Наступним кроком є налаштування аутентифікації у файлі `pages/api/auth/[...nextauth].ts`, який імпортує NextAuth та налаштування аутентифікації з файлу `~/server/auth.ts`. Цей файл auth.ts є основою системи аутентифікації програми, заснованої на бібліотеці NextAuth.js. За допомогою цієї бібліотеки проводиться інтеграція зовнішніх сервісів аутентифікації і створюються сесії. На самому початку файлу визначаються інтерфейси для об'єктів Session, JWT і User, які розширюють стандартні інтерфейси, що надаються NextAuth.js. Це дозволяє зберігати додаткові дані користувача, такі як ідентифікатор користувача, ім'я користувача, електронну пошту, зображення та ім'я.

Наступна значна частина файлу – об'єкт `authOptions`. Він визначає параметри, необхідні для налаштування бібліотеки NextAuth.js. У цьому об'єкті задаються функції зворотного виклику `session` та `jwt`, які обробляють дані сесії та JWT токена відповідно. В `authOptions` також визначаються провайдери для автентифікації. У цьому випадку використовуються Google та Credentials провайдери. Google провайдер використовує змінні оточення для `clientId` та `clientSecret`. Credentials провайдер забезпечує автентифікацію на

основі облікових даних, таких як електронна пошта та пароль користувача. При цьому проводиться перевірка на наявність користувача в базі даних і звіряння введеного пароля зберігається в базі, що здійснюється за допомогою функції verifyPassword. Налаштування сесії в `authOptions` дозволяють задати максимальний час життя сесії, період оновлення та стратегію, та інші параметри. Використовується стратегія "jwt", що означає, що сесії зберігаються в JWT-токені. Це забезпечує безпеку та ефективність при зберіганні даних сесії.

JWT (JSON Web Token) - це відкритий стандарт (RFC 7519) для безпечної передачі даних між двома сторонами у вигляді об'єкта JSON. Ці дані можуть бути перевірені та довірені, оскільки вони мають унікальний цифровий підпис. JWT може бути підписаний з використанням секретного ключа (з використанням алгоритму HMAC) або пари відкритого/закритого ключа за допомогою RSA або ECDSA [9].

JWT складається з трьох частин: заголовка (Header), корисного навантаження (Payload) та підпису (Signature). Кожна частина розділена точками (.), утворюючи таким чином три частини: xxxxx.yyyyy.zzzzz.

1. Заголовок (Header) зазвичай складається з двох частин: типу токена, який є JWT, та алгоритму підпису, такого як HMAC SHA256 або RSA. Потім цей JSON кодується в Base64Url.
2. Вміст (Payload) складається з елемента JSON який описує твердження. Твердження - це заяви про сутність (зазвичай користувачеві) та додаткову метаінформацію. Є три типи тверджень: зарезервовані, публічні та приватні твердження. Корисне навантаження також кодується в Base64Url.
3. Підпис (Signature) створюється з використанням закодованих заголовків, корисного навантаження, секрету та алгоритму, зазначеного в заголовку.

JWT зазвичай використовується для аутентифікації та авторизації. При вході до системи користувач надсилає свої облікові дані. Сервер перевіряє цю

інформацію і, якщо все гаразд, створює JWT та повертає його назад клієнту. Надалі клієнт використовуватиме цей токен для підтвердження своєї ідентичності. Після аутентифікації, при кожному запиті на захищений ресурс, клієнт повинен включати JWT, зазвичай, у заголовку авторизації. Сервер перевіряє JWT і, якщо він вірний, надає доступ до захищених ресурсів.

Перевагою JWT є те, що вони можуть бути самодостатніми. Інформація, що міститься в токені, дозволяє уникнути повторного звернення до бази даних для автентифікації кожного запиту. Це значно прискорює роботу програми та спрощує процес аутентифікації та авторизації.

В кінці файлу визначається функція getServerAuthSession, яка використовує функцію getServerSession від NextAuth.js для доступу до даних сесії на стороні сервера. Це дозволяє керувати сесією користувача та забезпечує безпеку при передачі даних між клієнтом та сервером.

У файлі `context.ts` ми працюємо над створенням контексту для нашої програми. Контекст у цьому випадку - це об'єкт, який передається у всі точки маршруту у програмі й зазвичай використовується для забезпечення доступу до різних рівнів програми, таких як бази даних, сесії користувача та інші загальні дані та функції.

У даному файлі `context.ts` ми визначаємо типи для наших опцій контексту та включаємо в нього сесію та базу даних. Причина, через яку база даних включена в контекст, полягає в тому, що це забезпечує універсальний доступ до бази даних з будь-якої точки нашої програми без необхідності повторного імпорту або встановлення з'єднання з базою даних. Це робить наш код чистішим та ефективнішим, оскільки ми можемо отримати доступ до бази даних у будь-якому місці нашої програми просто використовуючи контекст. Сесії також включені до контексту, оскільки вони є важливою частиною управління ідентифікацією користувача. Сесії дозволяють відстежувати стан входу в систему для кожного користувача і надавати досвід користувача на

основі їх даних. Це може включати збереження переваг користувача, надання доступу до певних функцій на основі їх ролей та інше.

Функція createContextInner надає базову функцію створення контексту, яка може бути використана в тестах або для створення статичних сторінок. Вона приймає об'єкт опцій та повертає об'єкт контексту із сесією та базою даних. Функція createContext забезпечує нашу основну логіку створення контексту. Вона використовує функцію getServerAuthSession для отримання сесії з Next.js і потім передає отриману сесію в функцію createContextInner для створення остаточного контексту. Тип `Context` використовується для створення контексту, який повертається функцією `createContext`. Це допомагає TypeScript визначити, що саме знаходиться в нашому контексті, і забезпечує строгу типізацію для нашого контексту, що допомагає запобігти помилкам при використанні контексту в нашому додатку.

Під словом "контекст" ми зазвичай маємо на увазі спеціальний об'єкт, який стає своєрідним дзеркалом нашої програми, відображаючи всі його ключові області. Цей контекст є універсальним засобом доступу до елементів нашої програми, таких як сесії користувачів та база даних. Так, не маючи потреби у повторному імпорті або налаштуванні з'єднань з базою даних, ми полегшуємо наш код, знижуючи його складність та збільшуючи зручність читання та підтримки.

Сесії користувачів – це життєво важливий елемент будь-якої системи аутентифікації. Вони допомагають нам відстежувати, коли користувач увійшов до системи, і надавати йому персоналізований досвід, що базується на його даних. Завдяки цим сесіям ми можемо створити гнучкіші та адаптивні інтерфейси, які відповідають індивідуальним потребам кожного користувача.

При цьому включення бази даних у контекст надає нам централізований спосіб взаємодії з нашими даними. Тепер ми можемо легко отримати доступ до бази даних з будь-якої точки нашої програми, не турбуючись про те, як саме встановлено з'єднання з базою даних або які параметри використовуються. Це

спрощує роботу з базою даних і робить наш додаток більш потужним та гнучким.

У файлі `context.ts` ми визначаємо функцію `createContextInner`, яка є основою створення нашого контексту. Ця функція може бути використана у різних сценаріях, таких як тестування чи створення статичних сторінок. Це дозволяє нам гнучко налаштовувати контекст залежно від поточних потреб, зберігаючи при цьому однаковість і цілісність нашого додатка. Потім, функція createContext розширює базову функцію, використовуючи getServerAuthSession для отримання поточної сесії користувача з Next.js, а потім передає цю сесію в функцію createContextInner. В результаті ми отримуємо повністю сформований контекст, який можна використовувати у будь-якому місці нашої програми.

Завершивши налаштування контексту, роутерів tRPC та систему авторизації, можна розпочати основну частину — реалізацію серверної логіки. Цей процес полягає у створенні функцій та процедур, які будуть обробляти різні запити від клієнтів та керувати діями у додатку. Саме в цьому місці додаток починає оживати, стаючи функціональним та корисним. При розробці серверної логіки програми важливо забезпечити безпеку, надійність і зручність роботи з даними. Бібліотека tRPC є ефективним інструментом для вирішення цих завдань. Він забезпечує безперервну взаємодію між сервером та клієнтом, забезпечуючи строгу типізацію на всіх етапах роботи з даними, що допомагає уникати помилок, пов'язаних з неправильним використанням або інтерпретацією даних.

Серверна логіка починає оживати зі створення функцій та процедур, які обробляють різні запити від клієнтів та керують діями у додатку. У файлі `server/api/router/user.ts`, наведений код демонструє, як можна розробити роутер користувача `userRouter` з використанням tRPC, який оброблятиме запити, пов'язані з діями користувача.

Для роботи з даними користувача створюється процедура `getSession`. Це базова функція, яка забезпечує доступ до даних користувача під час його сесії. Вона просто повертає інформацію про поточну сесію користувача з контексту. Потім слідує процедура `getData`, яка відповідає за отримання даних користувача з бази даних. Для цього використовується Prisma, що дозволяє легко працювати з базою даних. Запит до бази даних виконується асинхронно, тому необхідно використовувати ключове слово Async.

Реєстрація нового користувача — важлива частина будь-якої програми. У роутері це забезпечує процедура `signUp`. Вона приймає на вхід електронну пошту та пароль нового користувача, перевіряє, чи є у базі даних користувач із такою поштою, і якщо ні – створює нового. При цьому пароль користувача не зберігається у відкритому вигляді, а шифрується за допомогою надійного алгоритму bcrypt.

У процедурі `changeUserData` реалізовано можливість зміни персональних даних користувача. Тут цікаво відзначити використання хмарного сховища Cloudinary зображень для збереження зображення користувача. Це забезпечує зручне та надійне зберігання зображень, без необхідності організації власного сховища.

Інші процедури роутера також виконують важливі функції. `getByUsername` дозволяє знаходити користувачів на ім'я користувача. `follow` дозволяє підписуватись на інших користувачів або відписуватись від них. `search` надає можливість пошуку користувачів та організацій.

І, нарешті, процедури `isFollowing`, `getFollowers` та `getFollowing` дозволяють працювати з підписками користувачів. Вони дозволяють перевірити, чи підписаний користувач на іншого користувача, а також отримати списки підписників та підписок конкретного користувача.

Завершуючи, можна сказати, що розвиток веб-додатків досяг нового рівня завдяки ефективному поєднанню Prisma, Next.js і tRPC. Prisma відкриває

надійний доступ до даних з урахуванням типів, а Next.js піклується про всю серверну логіку, оптимізуючи роботу та підвищуючи продуктивність. tRPC додається до цієї екіпи, спрощуючи розробку API та забезпечуючи безпечне спілкування між клієнтом та сервером.

3.4. Розробка клієнтської частини за допомогою React та Tailwind CSS

Коли мова йде про розробку веб-платформи, вибір відповідного інструменту для стилізації може бути досить складним завданням. Існує безліч бібліотек і фреймворків, кожен з яких має унікальний набір функцій і можливостей. Однак, в останній час, особливу увагу привертає такий інструмент, як Tailwind CSS. Ця бібліотека стилізацій є зовнішнім утилітарним підходом, що відрізняється від більшості традиційних CSS-фреймворків.

Для початку процес впровадження Tailwind CSS починається з встановлення бібліотеки за допомогою менеджера пакетів, такого як npm або yarn. Слід зазначити, що для роботи Tailwind CSS потрібен Node.js версії 12.13.0 або вище. Для додавання Tailwind CSS в проект слід використовувати команду `npm install tailwindcss`.

Після встановлення наступним кроком буде налаштовано Tailwind CSS. Це досягається за допомогою створення конфігураційного файлу, який забезпечує можливість детальної настройки різних аспектів роботи Tailwind, включаючи кольорову палітру, відступи, розміри шрифтів і т.д. Команда `npx tailwindcss init` буде корисною для створення конфігураційного файлу `tailwind.config.js` в кореневому каталозі проекту. Файл `tailwind.config.js`, надає можливість налаштування різних аспектів фреймворку, включаючи медіа-запити, палітри кольорів, анімації та ін. У наведеному файлі конфігурації описується кілька основних елементів. `content` - це масив шляхів до файлів, в яких Tailwind CSS шукатиме класи для використання при генерації CSS.

В даному випадку вказані всі файли з розширеннями js, ts, jsx, tsx у директорії `src`. `theme` - визначає налаштування теми за замовчуванням, які використовуються Tailwind CSS. У секції plugins вказуються використовувані

плагіни. В цьому випадку використовуються плагіни "daisyui" та "tailwind-scrollbar". Плагін "tailwind-scrollbar" дозволяє стилізувати смуги прокручування.

DaisyUI - це плагін для Tailwind CSS, що надає набір готових UI-компонентів і утиліт, які полегшують і прискорюють процес розробки інтерфейсу користувача. DaisyUI розширює можливості Tailwind, додаючи готові стилі для кнопок, форм, карток, модальних вікон, табів та інших елементів інтерфейсу, які зазвичай використовують у веб-додатках.

Однією з особливостей DaisyUI є підтримка тем. Ви можете легко визначити власні теми, а потім застосовувати їх до всіх компонентів DaisyUI в проекті. У прикладі, який ви надали, у конфігураційному файлі Tailwind CSS визначено дві теми: "light" та "dark". Кожна тема визначає власні кольори для різних типів елементів, таких як основні, додаткові, акцентні і т.д.

Далі, щоб під'єднати Tailwind CSS до проекту React, використовується PostCSS. У файлі `postcss.config.js` додаються `tailwindcss` і `autoprefixer` до списку плагінів. Потім стилі Tailwind імпортуються за допомогою директиви `@import` у файл стилю проекту.

Таким чином, в проекті React тепер доступна система стилізації Tailwind CSS. Завдяки цьому класи Tailwind можуть бути використані для додатків стилів до компонентів React. Це додає більше гнучкості та над зовнішнім виглядом додатків, що робить Tailwind CSS прекрасним вибором для контролю роботи з React.

3.4.1 Визначення структури проекту та створення основних компонентів.

Структура файлів та папок проекту в директорії `src` представлена таким чином:

1. `pages` – коренева директорія, в якій розташовуються всі сторінки програми. Усередині неї знаходяться піддиректорії та файли, що відображають різні розділи сайту та їх функціональність:

- [name]/index.tsx – сторінки користувачів(/@user).
- api/trpc/[trpc].ts... – API-роути програми, що взаємодіють із tRPC.
- api/auth/[…nextauth].ts... – API-роути програми, що взаємодіють із Next-Auth.
- _app.tsx – головний компонент програми, який обертає решту сторінок.
- editProfile.tsx, feed.tsx, index.tsx, login.tsx, startup.tsx, search – компоненти, які представляють відповідні сторінки сайту.
- events/[index.tsx, my.tsx] – сторінка подій.
- discover/[index.tsx, my.tsx] – сторінки розділу "discover". Де можна знайти пропозиції та запити користувачів, та організацій.
- chat – сторінки чату.
    1) index.tsx – сторінка усіх чатів користувача.
    2) [id].tsx – чат між користувачем та користувачем чи організацією.
- organizations
    1) index.tsx - сторінка з усіма організаціями.
    2) create.tsx – сторінка створення організації.
    3) [name]:
        1. edit.tsx – сторінка редагування організації. (/organizations/@organizationName/edit)
        2. index.tsx – сторінка організації. (/organizations/@organizationName)
        3. [chat] - сторінки чату організації.

2. `components` – папка, що містить усі компоненти React, що використовуються у проекті.
3. `constants` - місце для зберігання констант, що використовуються в різних частинах програми.
4. `server` – містить код серверної частини програми.

5. `styles` – у цьому каталозі знаходяться всі стилі програми, створені за допомогою Tailwind CSS та DaisyUI.
6. `types` – тут зберігаються типи TypeScript, які у додатку.
7. `utils` – допоміжні функції та утиліти, які можуть бути використані в різних частинах програми.

pages - це важлива директорія в Next.js, яка є основою маршрутизації на стороні сервера в цьому фреймворку. Всі файли в цій папці автоматично перетворюються на маршрути програми на основі їх імен та розташування. Наприклад, файл `pages/index.tsx` буде представлений як головна сторінка програми (`/`), а `pages/login.tsx` буде доступний за маршрутом `/login`. Структура підпапок також відбиває маршрутизацію. Якщо файл називається `pages/posts/[id].tsx`, він буде представлений як маршрут, наприклад `/posts/1`, де `1` - це динамічний параметр. Також варто відзначити, що в директорії `pages` є спеціальний файл `_app.tsx`. Це кореневий компонент програми, який обертає решту сторінок. В `_app.tsx` зручно розміщувати загальні для всієї програми елементи, такі як загальні стилі, контексти або компоненти макета.

Крім цього, Next.js надає спеціальні API-маршрути, які визначаються в піддиректорії pages/api. Це серверні маршрути, які дозволяють створювати обробники запитів на стороні сервера без необхідності налаштовувати окремий сервер. В даному випадку `api/trpc/[trpc].ts` - це файл, де налаштовані обробники tRPC, `api/auth/[…nextauth].ts` - це файл, налаштування Next-Auth.

Компонент Layout у NextJS служить для обертання основного вмісту сторінки, надаючи єдиний макет для різних сторінок. Це дуже корисно при створенні часток інтерфейсу користувача, які залишаються незмінними на більшості або всіх сторінках сайту, таких як навігаційні панелі (navbar), шапки (header), футери (footer) або бокові панелі (sidebar).

У файлі src/components/layout.tsx. Визначено два компоненти `MainLayout` і `layout`.

`MainLayout` приймає наступні пропси(реквізити): `children`, `session` і `route`. Реквизити схожі на аргументи функції, які надсилаються у компонент як атрибути. Цей компонент повертає загальний макет для сторінки, що складається з компонентів `Drawer`, `Navbar`, `LeftSideBar` та основного вмісту, представленого через проп `children`. Компонент `Drawer` служить контейнером для вкладених компонентів, включаючи `Navbar` і `LeftSideBar`. `Navbar` представляє собою навігаційну панель, в той час, як `LeftSideBar` - бокову панель. `session` і `route` передаються в `LeftSideBar` як пропси.

`layout` - це функція вищого порядку, яка приймає в якості аргументу `children` і обробляє їх у `MainLayout`, додаючи логіку управління станом сесії та маршрутизації. Цей компонент використовує хук `useSession` з `next-auth` для отримання поточної сесії користувача та хук `useRouter` для доступу до об’єкта маршрутизатора.

Якщо статус сесії - `loading`, відображається компонент `Spinner`, який, є індикатором завантаження. Якщо обліковий запис аутентифіковано, але в ньому немає імені користувача, користувач перенаправляється на сторінку `/startup`, для доповнення інформації користувача. Якщо користувач автентифікований і у нього є ім’я користувача, `MainLayout` повертається з компонентом children. Якщо користувач не аутентифікований, він перенаправляється на сторінку `/login`.

Компонент Drawer являє собою адаптивний сітчастий макет, що забезпечує функціональність бокової панелі (sidebar) завдяки бібліотеці daisyUI, яка може бути показана або прикрита. На великих екранах за замовчуванням закривається бокова панель, а замість неї не відображається компонент LeftSideBar, в той час як на маленьких екранах вона зазвичай закривається, і може відображатися при натисканні кнопки в навігаційній панелі (Navbar). Це дозволяє створити адаптивний і зручний інтерфейс користувача, який підлаштовується під різні розміри екранів і забезпечує зручну навігацію за додатком.

Як приклад розробки динамічного веб-програми на Next.js, розглянемо сторінку /organizations/[name]/index.tsx. Ця сторінка використовує підхід, який називається динамічною маршрутизацією, який позначається в URL-адресі як [name]. Це означає, що [name] може набувати будь-якого значення, наприклад, імені організації, і на основі цього шаблону можна створити безліч унікальних сторінок. При переході користувача на одну із цих сторінок на сервері викликається функція `getServerSideProps`. Ця функція отримує контекст запиту, і в цьому контексті `context.params.name` є значенням, підставленим замість [name] URL. Це ім'я організації, яке використовується для динамічної побудови сторінки.

Функція `getServerSideProps` спочатку перевіряє, чи починається ім'я з параметрів "@". Якщо ні, то повертається результат із пропсом `id` та `notFound: true`, що призведе до відображення сторінки 404. Якщо перевірка проходить успішно, створюється екземпляр(instance) createServerSideHelpers, використовуючи для цього роутер appRouter і контекст, створений функцією createContextInner. Цей екземпляр потім використовується для виконання запиту до сервера з метою отримання даних організації на ім'я, вказане в URL.

На сторінці використовується компонент `Layout` як основа, всередині якого розміщуються компоненти `OrganizationPage` та `UpcomingEvents`, що відображають інформацію про організацію та майбутні події відповідно. Ці компоненти отримують дані організації у вигляді пропсів та використовують їх для відображення відповідної інформації.

Продовжуючи аналіз коду сторінки організації, перейдемо до основного компоненту, що представляє сторінку організації - OrganizationPage. Цей компонент приймає як пропс `organization`, дані якого були отримані на сервері і передані через `getServerSideProps`. Усередині компонента `OrganizationPage` ми спочатку ініціалізуємо деякі стани за допомогою хуку `useState`. Наприклад, `isLoading` використовується для відображення індикатора завантаження під час початкового завантаження сторінки, а `selectedInput`

визначає, які пости (події, пропозиції, запити або звичайні пости) зараз відображаються.

Далі використовуємо хук `useSession` з бібліотеки `next-auth`, який дозволяє отримати інформацію про поточну сесію користувача. Це може бути корисним, наприклад, для визначення, чи має поточний користувач права на створення нових постів чи подій у цій організації. Також використовується хук `useEffect` для встановлення стану `isLoading` у `false` після початкового рендерингу компонента, що призводить до відображення вмісту сторінки замість індикатора завантаження.

Також використовуються хуки `useMemo` та `useEffect` для повторної вибірки даних на основі обраного вхідного значення `selectedInput`. На основі цього вхідного значення вибирається тип контенту, який потрібно відобразити, і виконується відповідний запит повторної вибірки даних.

Для кожного типу контенту (пости, події, запити, пропозиції) створюються свої нескінченні запити за допомогою хука useInfiniteQuery з бібліотеки tRPC, при заході на сторінку за замовчуванням підвантажуються лише данні постів. Ці запити виконуються з урахуванням ID організації, щоб отримати відповідні пости чи події цієї організації. При зміні вхідного значення `selectedInput`, підвантажуються данні відповідно значенню `selectedInput`.

У `return` компонента `OrganizationPage` використовується тернарний оператор для відображення компонента `Spinner`, якщо дані все ще завантажуються, інакше відображається вміст сторінки. Цей вміст включає заголовок, кнопки для вибору типу контенту, які при натисканні змінюють значення `selectedInput`, і обраний компонент введення, який залежить від `selectedInput` і відображає форму для створення нового посту або події, запиту, чи пропозиції. Потім відображаються вибрані пости, які також залежать від `selectedInput` (рис 3.3).

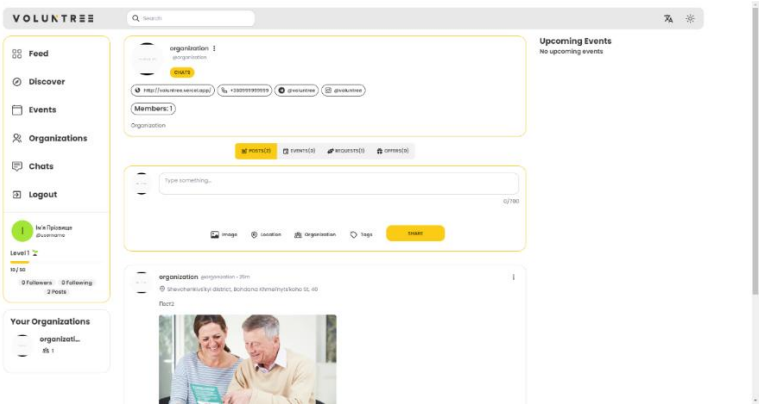

Рисунок 3.3 Сторінка організації

3.4.2 Розробка компонента для відображення списку постів, подій, запитів, та пропозицій.

За приклад візьмемо відображення постів організації. За схожим принципом працює відображення пропозицій, подій та запитів, трохи відрізняючись зовнішнім виглядом для різних типів постів.

Компонент `Posts` призначений для відображення списку постів. Цей компонент отримує властивості список постів та функцію для завантаження наступної порції даних. Усередині цього компонента використовується хук `useEffect` спільно з `IntersectionObserver` для відстеження, коли користувач доскролив до кінця списку, щоб викликати функцію для завантаження наступної порції постів.

У розділі `return` компонента `Posts` бачимо перевірку, чи є пости для відображення. Якщо немає постів, виводиться повідомлення що пости не знайдені. В іншому випадку, для кожного посту у списку постів відображається компонент `Post`.

Сам компонент `Post` реалізований як функціональний компонент. Кожен пост відображається всередині свого блоку з інформацією про організацію, час створення посту, текст посту, зображення, теги та автора. Компонент також надає можливість лайкати пости, використовуючи хук `likePost` з бібліотеки tRPC. У стані `liked` зберігається інформація про те, чи пост лайкнутий поточним користувачем, і на основі цього стану змінюється

відображення кнопки лайка і кількість лайків `heartCount` (рис 3.4), за схожим принципом працюють і події (рис 3.5) та інші типи постів.

Рисунок 3.4 Пост організації

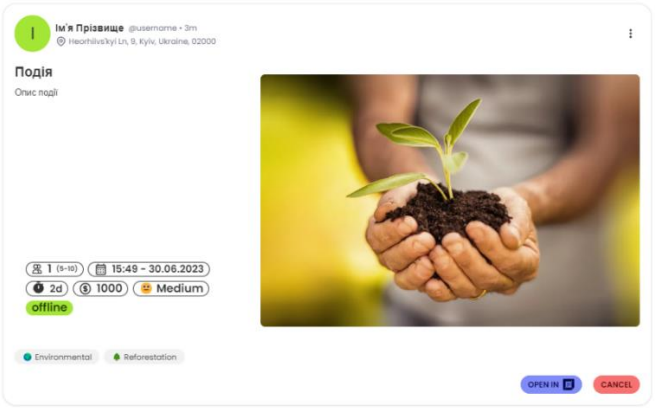

Рисунок 3.5 Подія створена користувачем

3.4.3 Створення компонента для створення нового поста, події, запиту, та пропозиції.

В цей раз візьмемо за приклад компонент `OfferInput`, який дозволяє користувачеві створювати нові пропозиції. Пропозиції містять різні дані, такі як заголовок, опис, зображення, місцеперебування, організацію, навички та теги. Це все збирається в цьому компоненті та посилається на сервер.

Спочатку ми ініціалізуємо низку станів для змісту введених користувачем даних:

- value: містить опис запиту.
- location: містить інформацію про місцеперебування.
- images: масив для зберігання зображень, пов'язаних із запитом.

- selectedSkills, selectedTags, selectedOrganization: ці стани використовуються для зберігання обраних користувачем навичок, тегів та організації.
- paid: цей стан використовується для зберігання інформації про ціну, якщо платний запит.

Після цього ми створюємо функцію handleOfferCreation, що відповідає за відправку інформації про новий запит на сервер. Вона збирає всі введені користувачем дані, перетворює їх у відповідний формат і відправляє на сервер за допомогою мутації 'createOffer'.

Також ми маємо функцію compressImages, яка стискає зображення перед відправкою їх на сервер. Це допомагає оптимізувати розмір даних, надісланих на сервер, та зменшує час завантаження. Компонент `OfferInput` включає в себе ряд допоміжних компонентів, що дозволяють користувачам вводити різноманітну інформацію для створення нового запиту. Спочатку взаємодіємо з компонентом `Avatar`. Він служить для відображення зображення профілю користувача. Якщо аватар користувача відсутній, цей компонент використовує першу букву імені користувача в якості заповнювача. Потім маємо компонент `OrganizationInput`. Цей компонент показує список організацій, створених користувачем, надаючи можливість вибрати одну з них для асоціації зі створюваним запитом.

В наступному рядку ми маємо два пов'язаних компоненти: `TagsInput` та `SkillsInput`. Вони виводять списки заздалегідь визначених тегів та навичок відповідно. Ці списки дають користувачам можливість вибрати наявні варіанти або ввести власні, що забезпечує гнучкість та персоналізацію в процесі створення нового запиту. Що стосується компонента `MiniMap`, він використовується для визначення місцеперебування за допомогою карти Google. Користувач може встановити маркер на карті, щоб точно вказати розташування, яке потім зберігається в стані `location`.

І, нарешті, ми генеруємо форму введення даних, що містить поля для введення різних видів даних, таких як опис, місцеперебування, зображення, навички, теги таке інше. Користувач також може вказати, чи є запит платним, а якщо так, ввести ціну. Після введення всіх даних, користувач може натиснути кнопку "Create", щоб створити новий запит. За таким же принципом працює створення нового поста, події, запиту, та пропозиції для організацій та користувачів (рис 3.6).

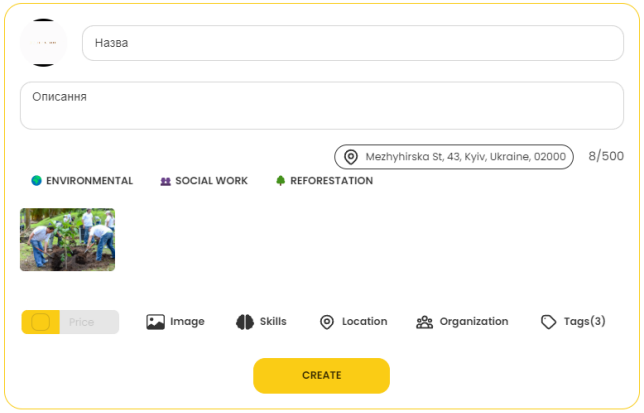

Рисунок 3.6 Форма створення пропозиції

3.5 Фільтрація даних та інтеграція з Google Maps

При натисканні на кнопку фільтрації в інтерфейсі користувача, дані, які повинні бути використані для фільтрації, передаються в функцію, яка викликає відповідний запит для повторного отримання даних у tRPC, після чого виконується запит на сервері, де завдяки запиту у Prisma findMany яка приймає об’єкт where відбувається фільтрація таких даних як: теги, навички, континенти, країни, тип події, початок та кінець події, мінімальний рівень користувача, розташування за радіусом, кількість потрібних учасників, і т.д (рис 3.7).

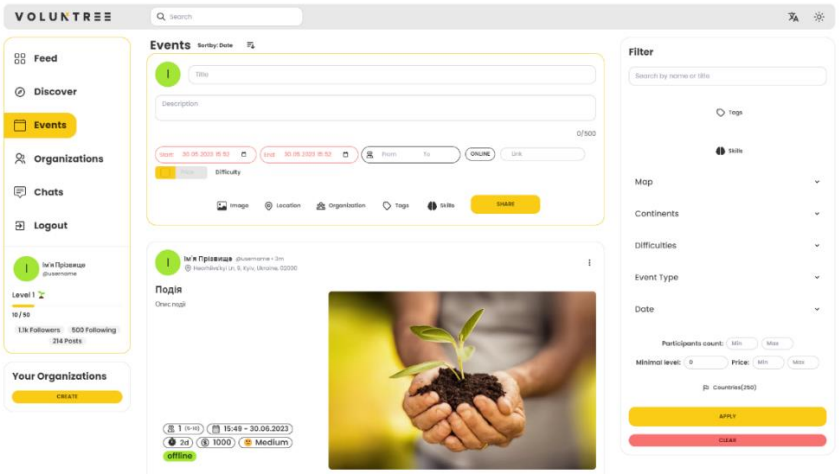

Рисунок 3.7 Сторінка для пошуку, створення та фільтрації подій

Пошук за радіусом відбувається завдяки інтеграції з Google Maps, для цього використовуються дві основні бібліотеки: @react-google-maps/api та use-places-autocomplete. @react-google-maps/api використовується для відображення карт Google та об'єктів на них (рис 3.8), таких як маркери, а use-places-autocomplete використовується для реалізації функціональності автозаповнення під час пошуку місць (рис 3.9).

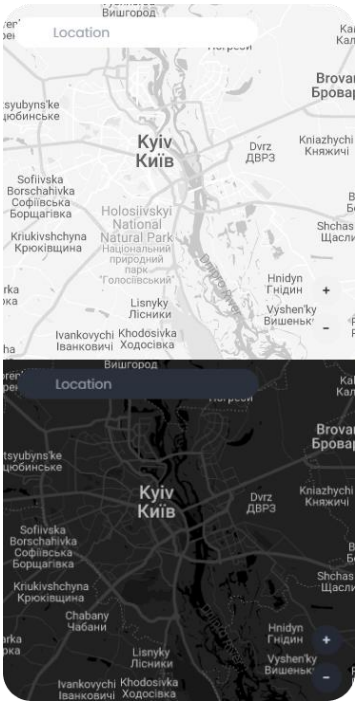

Рисунок 3.8 Стилізований UI-Елемет Google карт для світлої та темної теми додатка.

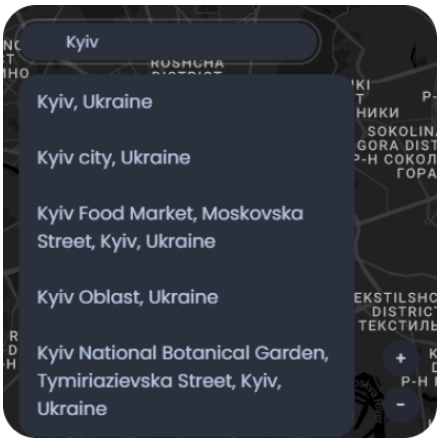

Рисунок 3.9 UI-Елемет для пошуку місця.

Компонент PlacesAutocomplete, дозволяє користувачеві ввести назву місця і вибрати одне із запропонованих у списку. Коли користувач вибирає місце, його географічні координати (широта та довгота) визначаються за допомогою функцій `getGeocode` та `getLatLng`. Ці координати використовуються для відображення маркера на карті.

На рис. 3.10 зображено спосіб вибору координат та радіусу для пошуку подій, використовуючи Google Maps.

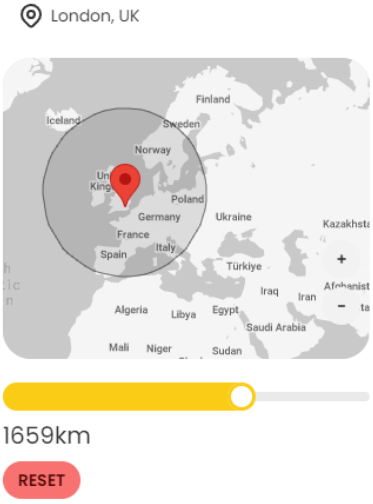

Рисунок 3.10 UI-Елемент для визначення координат та радіусу для фільтрації

Фільтрація виконується при умові якщо маркер встановлений на карті. Після цього виконується функція ‘getRadiusCords’. Ця функція приймає об'єкт з координатами центру (широта `lat` та довгота `lng`) та радіусом `radius` у

кілометрах. Вона повертає об'єкт із чотирма значеннями: `maxLat`, `minLat`, `maxLng`, `minLng`. Координати обчислюються таким чином [10]:

$$radiansPerDegree = \frac{\pi}{180}, newRadius = radius \times \pi, earthRadiusKm = 6371$$

$$maxLat = lat + \left(\frac{newRadius}{earthRadiusKm}\right) \times radiansPerDegree$$

$$minLat = lat - \left(\frac{newRadius}{earthRadiusKm}\right) \times radiansPerDegree$$

$$maxLng = lng + \frac{\left(\frac{newRadius}{earthRadiusKm}\right) \times radiansPerDegree}{\cos\left(lat \times radiansPerDegree\right)}$$

$$minLng = lng - \frac{\left(\frac{newRadius}{earthRadiusKm}\right) \times radiansPerDegree}{\cos\left(lat \times radiansPerDegree\right)}$$

Ці значення становлять координати найвищої (північної) та найменшої (південної) точок широти (`maxLat` і `minLat` відповідно), а також найвищої (східної) та найменшої (західної) точок довготи (`maxLng` та `minLng` відповідно), які утворюють межі квадрата.

Цей квадрат оточуватиме коло з центром у зазначених координатах і з радіусом, що дорівнює `radius`. Таким чином, всі точки всередині цього квадрата також будуть знаходитися всередині кола радіусом `radius`, центрованого на `lat` та `lng` (рис 3.11).

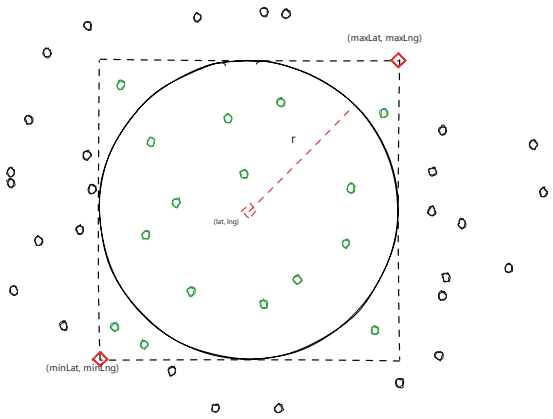

Рисунок 3.11 Візуалізація фільтрації подій за радіусом

3.6 Стиснення та завантаження зображень

Процес завантаження зображень є невід'ємною частиною більшості веб-платформ, що є актуальним для соціальних мереж, волонтерських платформ та багатьох інших веб-додатків. Важливим завданням тут оптимізація зображень, тобто. зменшення їхнього розміру без істотної втрати якості. Цей процес також називають "стисненням" зображень.

У поданому проекті реалізовано дві основні функції для стиснення зображень: `compressImage` та `compressImageAndHash`.

Функція `compressImage` приймає як аргументи файл зображення, максимальні розміри та параметр якості стиснення. Під час роботи функції зображення спочатку зчитується як Data URL, а потім підганяється під задані розміри. Потім створюється HTML-полотно, на якому малюється зображення за допомогою контексту 2D. Після цього зображення конвертується у формат `webp` із зазначеним рівнем якості.

WebP – це новий формат зображень, розроблений Google, оптимізований для більш швидкого розміщення в Інтернеті зображень меншого розміру. Зображення WebP приблизно на 30% менше за розміром у порівнянні з зображеннями PNG та JPEG з еквівалентною візуальною якістю. Крім того, формат зображення WebP має паритет функції з іншими форматами. Він підтримує [11]:

- Стиснення з втратами: Стиснення з втратами базується на кодуванні ключових кадрів VP8. VP8 – це формат стиснення відео.
- Стиснення без втрат: Формат стиснення без втрат розроблено командою WebP.
- Прозорість: 8-бітовий альфа-канал корисний для графічних зображень. Альфа-канал можна використовувати разом з RGB з втратами, функція, яка на даний момент недоступна в жодному іншому форматі.
- Анімації: підтримує анімовані зображення зі справжнім кольором.

- Метадані: можуть містити метадані EXIF та XMP (наприклад, використовувані камерами).
- Колірний профіль: може мати вбудований профіль ICC(International Color Consortium).

Функція `compressImageAndHash` доповнює роботу `compressImage`, додаючи генерацію хеша зображення. Після стиснення зображення до розмірів 128x128 пікселів з якістю 0.4 та подальшого стиснення з встановленими параметрами зі стисненого зображення витягуються пікселі. Ці пікселі потім кодуються за допомогою алгоритму BlurHash.

BlurHash – це відкритий алгоритм, розроблений для створення плейсхолдерів(placeholder) зображень [12]. Це дозволяє відображати колірний і світловий розподіл зображення до повного завантаження, покращуючи візуальний досвід користувача. Хеш BlurHash є коротким рядком, який можна ефективно зберігати і передавати.

Завантаження зображень на сервер здійснюється за допомогою функції uploadImage, яка використовує бібліотеку Cloudinary.

Cloudinary - це рішення програмного забезпечення як послуги (SaaS) для керування всіма медіа-активами веб-сайту або мобільного додатка в хмарі. Cloudinary пропонує наскрізне рішення для всіх потреб у зображенні та відео, включаючи завантаження, зберігання, адміністрування, трансформацію та оптимізовану доставку. Завантаження, обробка та доставка медіафайлів здійснюються на серверах Cloudinary, які автоматично масштабуються для обробки високого навантаження та стрибків трафіку [13].

Деякі з основних переваг та можливостей Cloudinary:

- Безшовна інтеграція: Cloudinary пропонує великі SDK та бібліотеки для різних мов програмування та фреймворків, включаючи Node.js, Python, PHP, Ruby, .NET, iOS та Android. Це спрощує інтеграцію служби до існуючих процесів розробки.

- Автоматична оптимізація: Cloudinary може автоматично оптимізувати зображення та відео для доставки за допомогою CDN (Content Delivery Network), що підвищує продуктивність та покращує користувальницький досвід.
- Трансформація зображень та відео: Cloudinary надає потужні інструменти для зміни розмірів, обрізки, накладання водяних знаків, перетворення формату та інших видів трансформації медіафайлів. Ці операції можна виконувати на льоту прямо при запиті.
- Широкий спектр форматів, що підтримуються: Cloudinary підтримує безліч форматів зображень і відео, що спрощує роботу з різними типами медіафайлів.
- Зберігання та доставка: Cloudinary надає надійне та безпечне сховище для зображень та відео, а також забезпечує швидку та ефективну доставку вмісту користувачам по всьому світу через CDN.
- AI та машинне навчання: Cloudinary використовує технології штучного інтелекту та машинного навчання для покращення якості медіафайлів, автоматичного кадрування, розпізнавання осіб та об'єктів та інших функцій.
- Безпека: Cloudinary пропонує безліч функцій безпеки, включаючи керування доступом, підтримку HTTPS та автентифікацію для забезпечення безпеки ваших медіафайлів.

Cloudinary надає широкий набір можливостей для роботи із зображеннями та відео, але існують й інші сервіси, які можуть бути використані як альтернативи.

Ось кілька із них:

- **Amazon S3 (Simple Storage Service):** Це хмарне сховище даних від Amazon, що пропонує масштабоване, доступне та надійне рішення для зберігання зображень та інших типів даних.

- **Google Cloud Storage:** Це хмарне сховище даних від Google, яке пропонує гнучкість, надійність та масштабованість, аналогічні Amazon S3. Це інтегрується з іншими послугами Google Cloud, такими як Google Cloud Vision для аналізу зображень.
- **Imgix:** Цей сервіс пропонує рішення щодо обробки зображень у реальному часі. Imgix дозволяє змінювати розміри зображень, налаштовувати їх якість, застосовувати ефекти та багато іншого прямо в URL зображення.
- **Fastly:** Це глобальна мережа доставки контенту (CDN), яка також пропонує можливості обробки зображень в реальному часі.
- **ImageKit.io:** Це ще один сервіс обробки зображень у реальному часі, що пропонує автоматичну оптимізацію, трансформацію та стиснення зображень.
- **Kraken.io:** Цей сервіс фокусується на оптимізації зображень, пропонуючи автоматичне стиснення та оптимізацію зображень для покращення продуктивності та швидкості завантаження.

Вибір між цими та іншими альтернативами залежить від специфічних вимог проекту, таких як вартість, необхідність певних функцій обробки зображень, інтеграція з іншими сервісами і т.д.

Загалом це є сучасним рішенням для оптимізації та завантаження зображень на сервер, використовуючи передові технології та принципи, включаючи стиснення WebP та алгоритм BlurHash. Приклад стиснення PNG зображення розміром 3840 × 2160 за допомогою функції compressImageAndHash. В результаті цієї операції отримуємо стиснене зображення розміром 1024 × 576, у формі рядка base64 та відповідний хеш (рис. 3.12).

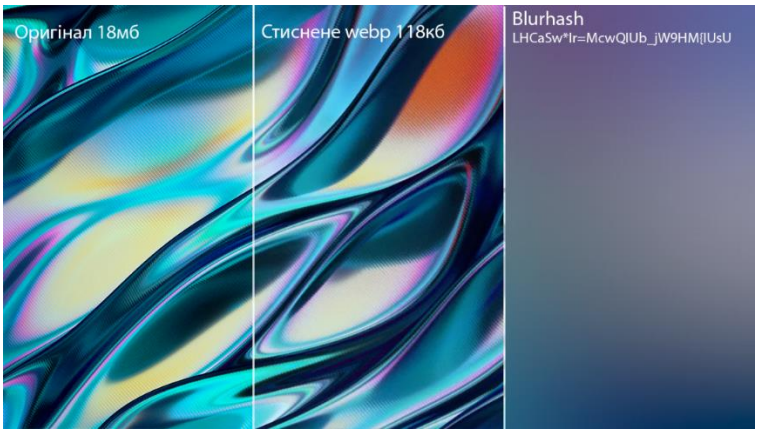

Рисунок 3.12 Порівняння типів зображень

3.7 Розгортання платформи на платформі Vercel

Vercel — це хмарна платформа, яка надає послуги з розгортання веб-програм. Вона залучає розробників, пропонуючи можливості автоматичної оптимізації додатків і прискорення процесу розгортання.

Суть роботи Vercel полягає в тому, що він підтримує так звану "serverless" архітектуру. У такому разі всі серверні функції стають незалежними, автономними блоками, які автоматично масштабуються відповідно до вимог трафіку.

Щоб завантажити проект на Vercel, потрібно зробити кілька простих кроків. По-перше, потрібно зареєструватися на сайті Vercel та створити нову команду або приєднатися до існуючої. Потім вибрати опцію "Import Project", вказати посилання на репозиторій GitHub, GitLab або Bitbucket і слідувати інструкціям з налаштування проекту. Після завершення налаштування Vercel автоматично розгорне проект. Також, за допомогою автоматичного розгортання та попереднього перегляду, кожен `commit` у гілку репозиторію автоматично збирається та розгортається на унікальному URL. Це дозволяє швидко та легко тестувати та демонструвати зміни без впливу на робочий проект (рис 3.13).

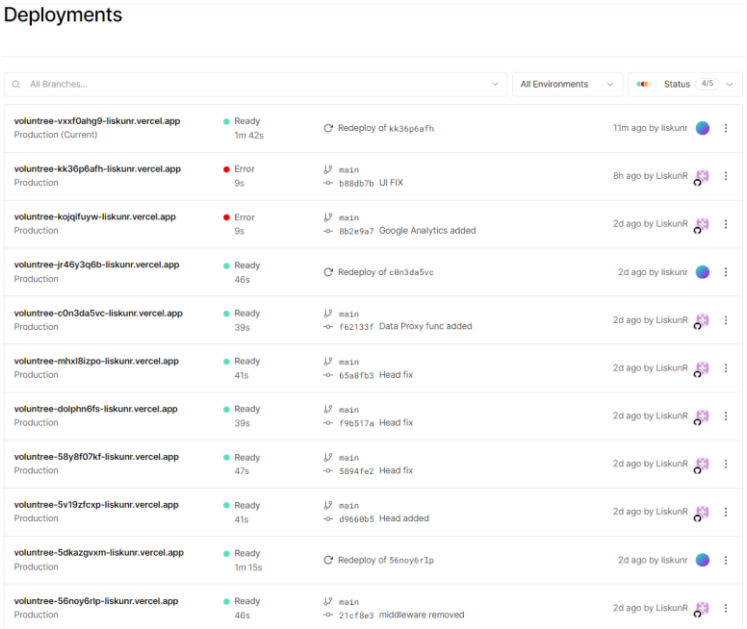

Рисунок 3.13 Панель розгортання проекту

Вибір Vercel як платформа для розгортання може бути обумовлений низкою причин. По-перше, він простий у використанні та пропонує відмінну інтеграцію з популярними системами керування версіями, такими як GitHub, GitLab та Bitbucket. По-друге, Vercel оптимізує веб-програми, надаючи швидке завантаження та низьку затримку. Нарешті, Vercel пропонує функції "serverless", які спрощують розгортання та масштабування веб-застосунків.

Вартість послуг Vercel може змінюватись в залежності від ваших потреб. Є безкоштовний план, який включає всі основні функції та підходить для невеликих проектів або проектів у стадії розробки. Платні плани починаються від 20 доларів на місяць і пропонують додаткові функції, такі як збільшені ліміти ресурсів, пріоритетна підтримка та багато іншого.

Vercel орієнтований на використання саме з Next.js, адже його засновники – це ті самі люди, які створили та підтримують Next.js. Це означає, що Vercel та Next.js працюють разом бездоганно. Next.js надає безліч можливостей і використання Vercel дозволяє повною мірою реалізувати потенціал цього фреймворку. Vercel має вбудовану підтримку багатьох функцій Next.js, таких як серверне та статичний рендеринг, створення API-маршрутів та оптимізація зображень. Це забезпечує гладку інтеграцію та значно спрощує

процес розгортання та обслуговування Next.js додатків. Наприклад, при використанні Next.js для створення API-маршрутів, Vercel автоматично обробляє та розгортає ці маршрути як серверні функції. Це скорочує час, необхідний для налаштування та керування серверною інфраструктурою, і дозволяє сконцентруватися на розробці програми.

У порівнянні з іншими платформами, такими як AWS, Google Cloud або Azure, Vercel може запропонувати простіший і швидший процес розгортання. Це робить його чудовим вибором для розробників та команд, які хочуть сконцентруватися на розробці, а не на управлінні інфраструктурою. Крім того, вартість послуг Vercel може бути більш привабливою для невеликих команд і стартапів, тоді як більші організації можуть віддати перевагу більш масштабованим і гнучким рішенням, пропонованим AWS, Google Cloud і Azure.

РОЗДІЛ 4. Впровадження та оцінка ефективності веб-платформи

У веб-платформи впровадження та оцінка ефективності платформи стають важливими аспектами успіху проекту. Кожний проект веб-платформи вимагає виваженої стратегії впровадження та підтримки, адже вони відіграють ключову роль у досягненні поставлених цілей і визначенні успішності платформи в довгостроковій перспективі.

4.1. Стратегії впровадження та підтримки платформи

Впровадження та підтримка веб-платформи – це складний процес, що вимагає уваги до деталей, стратегічного мислення та вміння підтримувати баланс між поточними потребами та перспективами для майбутнього. Цей процес починається з розробки маркетингової стратегії, яка є основним каменем для будь-якого виду бізнесу у мережі.

Маркетингова стратегія має бути чітко сформульована та орієнтована на конкретну цільову аудиторію. Вона допомагає побудувати шлях до успіху, визначити ключові точки взаємодії з клієнтами та виявити унікальні переваги платформи перед конкурентами. Але маркетингова стратегія - це не тільки просування продукту, це ще й робота над його покращенням, на основі зворотного зв'язку від користувачів та аналізу їхньої поведінки.

Наступний важливий аспект – технічна підтримка та обслуговування сайту. Технічне обслуговування забезпечує працездатність платформи, відповідає за усунення помилок, забезпечення безпеки та зручність використання. Якісна підтримка допомагає утримувати користувачів, підвищує їхню задоволеність і посилює їхню лояльність до бренду. Без цього елемента будь-яка, навіть найпродуманіша маркетингова стратегія, може обернутися провалом.

Управління спільнотою – це ще одна важлива частина процесу. Створення та підтримка активної та залученої спільноти користувачів

допомагає не лише збільшити охоплення та рівень залученості, але й отримати цінний зворотний зв'язок, який допомагає покращити продукт та поглибити розуміння потреб користувачів. Модерація, комунікація та взаємодія з користувачами – все це важливі аспекти управління спільнотою.

Потім іде моніторинг даних, отриманих під час моніторингу діяльності на платформі, можна робити висновки про ефективність поточної стратегії, виявляти слабкі точки та знаходити нові можливості для зростання. На даному етапі збираються, аналізуються та інтерпретуються всі дані - від користувача поведінки та відгуків до метрик веб-аналітики. Дослідження даних користувачів допомагає краще зрозуміти їхні потреби та переваги, що дозволяє покращити платформу та зробити її більш чуйною до потреб користувачів. Аналітика може допомогти також в ідентифікації трендів, прогнозуванні майбутньої поведінки користувачів та прийнятті обґрунтованих рішень.

Зрештою, довгострокове планування розвитку та оптимізація платформи – це включає стратегічне планування для забезпечення адаптивності та стійкості платформи в довгостроковій перспективі. Оптимізація включає пошук і виправлення проблем, підвищення ефективності та поліпшення якості послуг. Довгострокове планування та оптимізація допомагають переконатися, що платформа продовжить зростати і розвиватися разом із ринком, що змінюється, і потребами користувачів. Без стратегічного бачення і зусиль з оптимізації, що продовжуються, навіть найуспішніші платформи можуть зіткнутися з проблемами і втратити свою актуальність.

Загалом кожен з цих аспектів має свою важливість у процесі впровадження та підтримки веб-платформи. Необхідно збалансоване та старанне застосування всіх цих елементів, щоб забезпечити успіх веб-платформи у довгостроковій перспективі.

4.2. Методики оцінки ефективності веб-платформи

Оцінка ефективності веб-платформи – це ключова складова успішного розвитку та управління проектом. Розуміння того, наскільки добре платформа

відповідає потребам користувачів та досягає поставлених бізнес-цілей, дозволяє визначати пріоритети для подальшого розвитку, посилювати сильні сторони та працювати над слабкими. Існує кілька методик, які допомагають у цій важливій задачі.

Першим і, можливо, найвідомішим методом є веб-аналітика. Сучасні інструменти аналітики, такі як Google Analytics, Vercel Analytics (рис 4.1), AWStats, Hotjar та інші, надають глибоке занурення даних про поведінку користувачів на сайті. Скільки часу проводять користувачі на сайті, які сторінки і як часто вони відвідують, скільки дій роблять - це лише верхівка айсберга доступних даних. Вивчення цих метрик допомагає зрозуміти, які функції та контент на сайті найбільш привабливі для користувачів, і таким чином, як можна покращити сайт для підвищення його ефективності.

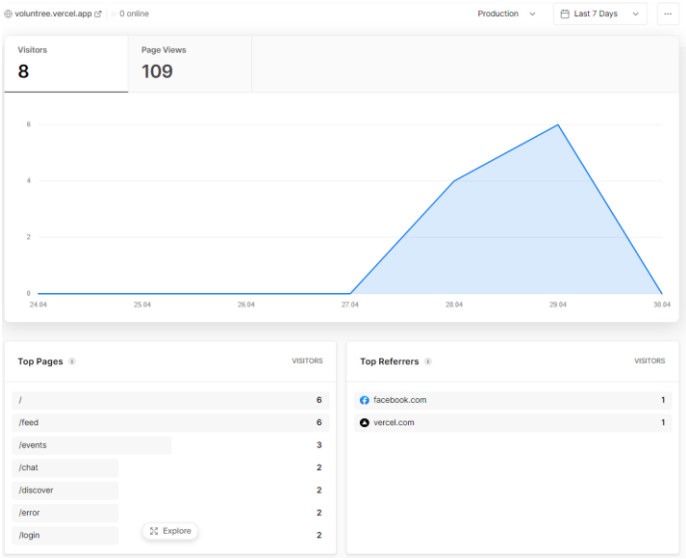

Рисунок 4.1 Панель Vercel Analytics

Другий метод – це аналіз поведінки користувачів. Спостереження за тим, як користувачі взаємодіють із сайтом, може дати багато корисної інформації. За допомогою інструментів відстеження поведінки, таких як Heatmaps, Scrollmaps або Recordings, можна побачити, куди клацають користувачі, які блоки перегортають, які елементи відволікають. Це допомагає зрозуміти, які елементи інтерфейсу працюють та сприймаються користувачами як повинні, а які – ні.

Третім методом є збір та аналіз відгуків користувачів. Зворотний зв'язок від реальних користувачів – це незамінне джерело інформації про те, що саме їм подобається чи не подобається на сайті, які проблеми чи складності вони стикаються. Різні форми опитувань, відгуків, форми зворотний зв'язок – це інструменти збирання цієї цінної інформації.

Четвертий метод – це A/B тестування. Цей метод дозволяє порівняти дві версії сторінки або елемента на сторінці, щоб визначити, яка з них є більш ефективною. A/B тестування – це науковий підхід до покращення сайту. Він дозволяє зробити ґрунтоване на даних рішення про те, що саме потрібно змінити на сайті для підвищення його ефективності.

Всі вищезазначені методи працюють разом, доповнюючи один одного. Тільки комплексний підхід дозволяє отримати об'єктивну та повну картину про те, наскільки ефективна веб-платформа. З іншого боку, оцінка ефективності – це одноразове дію, а постійний процес. Ринок, тренди, потреби та поведінка користувачів змінюються з часом, тому важливо регулярно проводити аналіз та коригувати стратегію розвитку веб-платформи.

4.3 Рекомендації щодо подальшого розвитку та оптимізації платформи

Для успішного розвитку та оптимізації соціально-гуманітарної веб-платформи необхідно враховувати безліч факторів. Важливо аналізувати поточний стан платформи та прогнозувати можливі тренди та напрямки.

Одним із основних аспектів є розширення функціональності платформи. Для цього слід продовжувати розробку нових функцій та інструментів, які будуть сприяти взаємодії користувачів та організацій. Важливо звернути увагу на впровадження платних підписок для користувачів і організацій, що дозволить збільшити фінансові ресурси платформи для розширення функціональності та покращення наданих послуг. Однак, необхідно точно підходити до цього питання, щоб не створювати бар'єрів для залучення нових користувачів.

Крім того, платформа може запропонувати функцію для збору пожертвувань для різних соціальних і екологічних проектах. Це стане додатковим джерелом фінансування для організацій та ініціатив, зареєстрованих на платформі, і дозволить їх успішно реалізувати. Таким чином, впровадження платних підписок і зборів пожертвувань буде важливим кроком у розширенні функціональності платформи.

Додатково, варто розглянути можливість розширення можливостей проведення онлайн-заходів, інтеграції платформи з соціальними мережами і месенджерами для розширення комунікацій і розширення аудиторії. Також слід звернути увагу на використання штучного інтелекту та машинного навчання для аналізу даних і надання персоналізованих рекомендацій користувачам.

Окрім розширення функціональності, для успішного розвитку платформи важливо працювати над покращенням якості контенту та досвіду користувача. Забезпечення актуальності та корисності інформації для користувачів, оптимізація навігації та доступності ресурсів для різних категорій користувачів є основою для створення платформи позитивного сприяння.

Далі звертаємо увагу на розвиток партнерських зв'язків. Співпраця з іншими організаціями, компаніями та державними структурами дозволить розширити можливості платформи, залучити нових користувачів і засоби для реалізації проектів. Важливо займатися розробкою партнерських експертних програм і спільних ініціатив, залучати наставників і менторів, а також взаємодіяти з медіа та іншими інформаційними каналами.

Нарешті, слід поділити увагу впровадження технологій та інновацій. У сучасному світі технології постійно розвиваються, і платформа має бути готовою до змін. Інтеграція нових комунікаційних інструментів та аналітичних систем допоможе оптимізувати роботу платформи та покращити її взаємодію з користувачами.

Реалізація запропонованих рекомендацій сприятиме успішному розвитку та оптимізації соціально-гуманітарної веб-платформи, покращить її сприйняття та довіру з боку користувачів, а також забезпечить стійкий розвиток та адаптацію до умов, що змінюються та потребам ринку.

Висновок

У ході дослідження було розроблено соціально-гуманітарну веб-платформу для волонтерства, яка сприяє поліпшенню організації та координації волонтерських проектів, залученню більшої кількості учасників та підвищенню їх мотивації. Розроблена платформа має низку переваг, які сприяють її ефективному функціонуванню та підтримці волонтерського руху.

У процесі аналізу теоретичних аспектів волонтерства та веб-технологій було встановлено, що волонтерство є важливим інструментом соціальної допомоги та сприяє покращенню суспільства. Були визначені сучасні тенденції волонтерства, які враховують зміну потреб та викликів у цій сфері.

Для розробки соціально-гуманітарної веб-платформи була створена концепція, де визначено основні функції та вимоги до платформи. Здійснено проектування архітектури та структури платформи, що допомогло впровадити необхідні елементи для забезпечення її функціональності.

Розробка веб-платформи здійснювалась з використанням сучасних технологій, включаючи Prisma, Next.js, tRPC, Next-Auth, React та Tailwind CSS. Були розроблені база даних та серверна частина, інтеграція систем аутентифікації та авторизації, клієнтська частина з використанням реактивних компонентів. Також було реалізовано функціонал фільтрації даних та інтеграції з Google Maps, а також оптимізацію роботи зображень.

Для успішного впровадження платформи та оцінки її ефективності були розроблені стратегії впровадження та підтримки, а також методики оцінки її роботи. Додатково, було сформульовано рекомендації з подальшого розвитку та оптимізації платформи, які враховують змінні потреби волонтерського середовища та технологічний прогрес.

Веб-платформа надає зручний інтерфейс для реєстрації волонтерів та пошуку цікавих проектів. Вона дозволяє волонтерам легко знайти проекти, що відповідають їхнім інтересам та навичкам, сприяючи більшій активності та

залученню нових учасників. Крім того, зручний інтерфейс платформи сприяє зрозумілості та доступності для широкого кола користувачів, забезпечуючи відкритий доступ до волонтерських можливостей.Також, важливим аспектом платформи є можливість спілкування та обміну інформацією між волонтерами та організаторами проектів. Це дозволяє ефективно обговорювати деталі проектів, ділитись досвідом та ідеями, сприяючи спільній роботі та досягненню поставлених цілей. Крім того, платформа може бути розширена та доповнена новими функціональними можливостями, забезпечуючи гнучкість та адаптивність до змінних потреб волонтерського середовища.

Проте, в процесі дослідження було виявлено деякі проблеми та недоліки, які потребують уваги та подальшого вдосконалення. Однією з основних проблем є забезпечення безпеки та конфіденційності даних на платформі. З огляду на значне зростання кількості особистої та чутливої інформації, необхідно приділяти належну увагу заходам з шифрування, захисту та контролю доступу до цих даних.

Крім того, при використанні безсерверної платформи, такої як Vercel, реалізація чатів на веб-платформі відбувається без використання WebSocket, що може вплинути на ефективність та миттєвість спілкування між волонтерами.

WebSocket - це комунікаційний протокол, який дозволяє встановлювати постійне з'єднання між клієнтом та сервером, що забезпечує миттєвий обмін повідомленнями. Для поліпшення якості комунікації рекомендується розглянути варіанти, такі як аренда окремого сервера для WebSocket або використання сервісів таких як Pusher або Podium, що забезпечить зручний та миттєвий обмін повідомленнями через WebSocket.

Також було виявлено потребу в аналізі волонтерської діяльності для кожного учасника та можливості порівняння результатів з іншими користувачами. Наявність інструментів для самоаналізу та оцінки своєї волонтерської діяльності може стати цінним додатком до веб-платформи. Такі

функціональні можливості дозволять користувачам оцінити свою активність, прогрес та вплив на проекти, а також забезпечать стимули для покращення та досягнення нових цілей. Крім того, аналіз діяльності інших користувачів може допомогти знайти взаємодію та співпрацю з ними, а також надати інсайти та ідеї для подальшого розвитку та покращення волонтерського середовища.

Крім цього, важливо врахувати питання масштабованості платформи. З врахуванням потенційного зростання числа користувачів та проектів, необхідно забезпечити, щоб платформа була готова ефективно працювати при великому обсязі даних та трафіку. Це може включати оптимізацію бази даних, використання кешування, горизонтальне масштабування та інші техніки для забезпечення швидкості та доступності платформи.

Також важливим аспектом є залучення організацій та спонсорів до платформи. Партнерство з організаціями, які займаються волонтерством, або зі спонсорами може забезпечити додаткові ресурси, фінансування та підтримку для волонтерських проектів. Платформа може надати інструменти для співпраці та комунікації з цими партнерами, сприяючи розвитку та розширенню волонтерського руху.

Для підвищення залучення та мотивації волонтерів, платформа може використовувати систему нагород та визнання. Вона може надавати волонтерам бейджі, рейтинги, сертифікати або інші форми визнання за їхню активність, внесок та досягнення. Така система може стимулювати волонтерів до подальшої участі та розвитку.

Для забезпечення успішного функціонування платформи важливо також залучити команду підтримки та модерації. Це можуть бути адміністратори, модератори чи інші фахівці, які забезпечують відповідну роботу платформи, моніторинг діяльності користувачів, вирішення конфліктів та надання допомоги волонтерам та організаторам проектів.

Загалом, розробка соціально-гуманітарної веб-платформи для волонтерства є складним і багатогранним процесом. Вона потребує ретельного аналізу потреб та вимог користувачів, ефективного використання сучасних технологій та уважного врахування проблем, які можуть виникнути під час розробки та експлуатації платформи. Шлях до успіху полягає в постійному вдосконаленні, слуханні користувачів та адаптації до змінних потреб волонтерського середовища.

Список використаних джерел

1. React — Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/React
2. Next.js by Vercel – The React Framework [Електронний ресурс] – Режим доступу до ресурсу: https://nextjs.org
3. Introduction | NextAuth.js [Електронний ресурс] – Режим доступу до ресурсу: https://next-auth.js.org/getting-started/introduction
4. Prisma Documentation | Concepts, Guides, and Reference [Електронний ресурс] – Режим доступу до ресурсу: https://www.prisma.io/docs
5. Tailwind CSS: What It Is, Why Use It & Examples [Електронний ресурс] – Режим доступу до ресурсу: https://blog.hubspot.com/website/what-is-tailwind-css
6. Installation – Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу: https://tailwindcss.com/docs/installation
7. Build end-to-end typesafe APIs with tRPC - DEV Community [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/nexxeln/build-end-to-end-typesafe-apis-with-trpc-3o3f
8. How to validate your form data with Zod [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@brianridolcedev/how-to-validate-your-form-data-with-zod-add25f90e7b3
9. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction.
10. Janmatuschek [Електронний ресурс] – Режим доступу до ресурсу: http://janmatuschek.de/LatitudeLongitudeBoundingCoordinates
11. Compression Techniques | WebP | Google for Developers [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/speed/webp/docs/compression
12. Blurhash [Електронний ресурс] – Режим доступу до ресурсу: https://blurha.sh/

13. Get Started with Cloudinary | Cloudinary [Електронний ресурс] – Режим доступу до ресурсу: https://cloudinary.com/documentation/cloudinary_get_started

Додаток А

Посилання на код проекту: https://github.com/LiskunR/voluntree