МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет водного господарства та природокористування
Навчально-науковий інститут кібернетики,
інформаційних технологій та інженерії
Кафедра комп'ютерних технологій та економічної кібернетики

Допущено до захисту:

Завідувач кафедри
комп'ютерних технологій та
економічної кібернетики
д. е. н., проф. П. М. Грицюк

_____
«____» _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬО-КВАЛІФІКАЦІЙНОГО РІВНЯ «МАГІСТР»

Розробка системи локалізації Web-додатків з використанням генеративного ШІ

Виконав:
здобувач вищої освіти за ОПП
«Інформаційні технології в бізнесі»
спеціальності 126
«Інформаційні системи та технології»
Перебенесюк Микола Васильович

Керівник:
к.е.н., доцент Волошин В.С.

Рецензент:
к.т.н., доцент Гладка О.М.

Рівне – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 44 с., 22 рис., 1 табл., 20 літературних джерел, 2 додатки.

Актуальність теми даної магістерської роботи полягає у використанні одного із сучасних підходів до розробки системи локалізації Web – додатків з використанням генеративного ШІ, що дозволяє значно спрощувати процес локалізації додатків на велику кількість підтримуваних мов ще на етапі розробки.

Об'єкт дослідження цієї роботи – система локалізації Web - додатків.

Предметною областю роботи є використання середовища NodeJS та бібліотеки ReactJS для розробки надійного Single Page Application (далі SPA) - додатку, включно з серверною частиною.

Метою магістерської роботи є створення системи локалізації додатків (сайтів, mobile та web - додатків) задля спрощення процесу перекладу тексту на велику кількість мов, а також для спрощення процесу зміни контенту (тексту).

У магістерській роботі: обґрунтовано доцільність використання представленого і реалізованого методу локалізації веб сайтів, здійснено огляд існуючих платформ для локалізації, проведено аналіз існуючих платформ для розробки системи, з подальшим вибором найбільш підходящих. Розглянуто ключові моменти розробки, а також проведено мінімальне тестування системи на предмет відповідності заявленим вимогам.

КЛЮЧОВІ СЛОВА: WEB-ДОДАТОК, СИСТЕМА ЛОКАЛІЗАЦІЇ, NODE JS, REACT JS, ГЕНЕРАТИВНИЙ ШІ

ЗМІСТ

ВСТУП

У сучасному світі локалізація веб-додатків стала ключовим аспектом, що забезпечує доступність і зручність користування для широкого кола користувачів з різних культур та мовних середовищ. Завдяки глобалізації, усе більше компаній та організацій прагнуть забезпечити користувачів можливістю взаємодії з продуктами та послугами рідною мовою, що значно покращує користувацький досвід і підвищує залучення аудиторії. У зв’язку з цим виникає потреба у розробці ефективних систем локалізації, які здатні швидко адаптувати текстовий контент веб-додатків до потреб міжнародних користувачів.

Метою даної магістерської роботи є створення веб-додатку, який забезпечує користувачам інструмент для створення та редагування локалізацій тексту на сайтах з використанням генеративного штучного інтелекту. Цей інструмент дозволить додавати текстові значення у форматі JSON, що можуть бути відредаговані та перекладені на різні мови за допомогою ChatGPT. Подальша інтеграція локалізованих текстів у проекти на React або Next.js здійснюється за допомогою спеціального npm пакета, що забезпечує динамічне завантаження даних з API та миттєве відображення змін на сайті.

Об’єктом магістерської роботи є процес локалізації контенту веб-додатків, а предметом – система для інтеграції та керування локалізаціями в середовищі веб-додатків.

Поставлені завдання для виконання магістерської роботи:

1. Дослідити основні вимоги до локалізації веб-додатків у контексті сучасних тенденцій і потреб глобального ринку.
2. Проаналізувати взаємозв'язок понять «JSON» та «API».
3. Провести порівняльну характеристику бібліотек локалізації веб-додатків.

4. Розробити архітектуру веб-додатку, що забезпечує ефективне керування текстовими даними.
5. Забезпечити динамічне оновлення контенту за допомогою API та виконати інтеграцію з React/Next.js для застосування локалізованого контенту в реальних веб-додатках.

Логіка дослідження **зумовила** структуру магістерської роботи: вступ, три розділи, висновки та список використаних джерел. У вступі аргументується актуальність теми, визначаються об’єкт та предмет дослідження, формулюється мета та завдання роботи. Перший розділ аналізує основні концепції локалізації веб-додатків і можливості генеративного штучного інтелекту в цьому контексті. Другий розділ присвячений побудові архітектури та функціоналу веб-додатка. У третьому розділі представлено практичну реалізацію рішення, зокрема опис API та інтеграцію з React/Next.js. Висновки містять результати роботи та підсумок виконаних завдань, а список використаної літератури – перелік джерел, що використовувалися в ході дослідження.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ СИСТЕМИ ЛОКАЛІЗАЦІЇ WEB-ДОДАТКІВ

1.1. Поняття системи локалізації Web-додатків

Локалізація (від англ. «localization») – це процес адаптації контенту та інтерфейсу продукту до специфічних мовних, культурних і регіональних особливостей цільової аудиторії. У контексті веб-додатків локалізація може полягати в перекладі текстів, зміні форматів дат, часу, чисел, а також врахуванні правових та культурних особливостей, для забезпечення зручності користування для користувачів з різних куточків світу.

Сучасний ринок веб-додатків вимагає наявності локалізованих продуктів, які можуть адаптуватися до різноманітних культур та мов. Це дозволяє збільшити охоплення, сприяти міжнародному розширенню бізнесу, а також покращити досвід користувачів за рахунок використання рідної мови та зручного інтерфейсу. Зважаючи на швидкий розвиток технологій і зростаючі очікування користувачів, важливо, щоб процес локалізації відбувався максимально швидко та ефективно. Але це не може відбуватись ефективно, поки зміною контенту зайняті саме розробники, а не менеджери. Оскільки зазвичай всі тексти веб - додатку редагуються в файлах проекту, виникають ситуації коли одна і та ж робота робиться двічі - перший раз тоді, коли менеджер визначає які тексти потрібно змінити та як, а другий раз - коли розробник виконуючи завдання від менеджера безпосередньо займається цим.

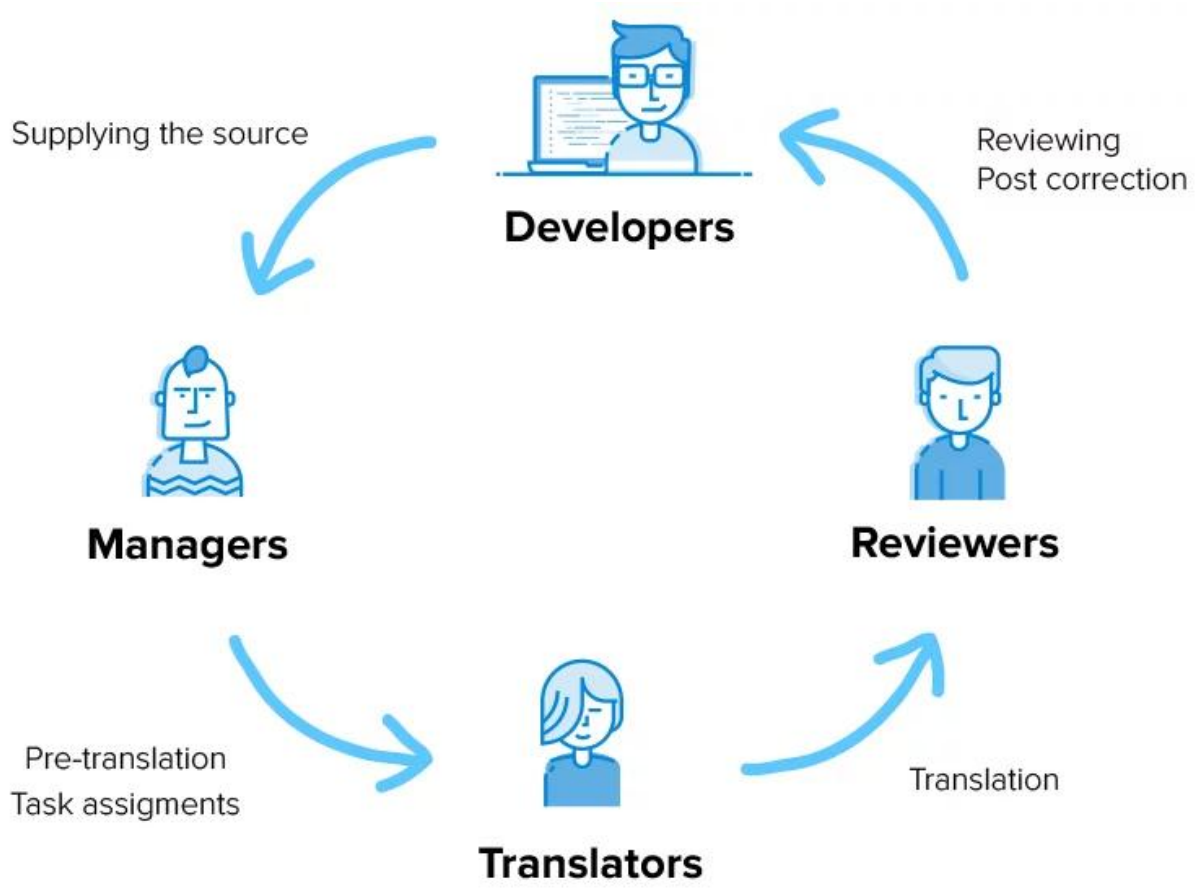

Рис 1.1. Стандартний процес локалізації тексту

Цей процес можна значно скоротити, якщо дати можливість менеджерам змінювати текст безпосередньо на сайті.

Системи локалізації зазвичай складаються з таких компонентів:

- **Система керування перекладами**. Цей компонент забезпечує зберігання, організацію та управління перекладами текстів. Вона дозволяє відслідковувати версії текстів, керувати термінами та забезпечувати автоматизацію деяких процесів перекладу.
- **API для отримання текстів локалізації**. Використання API для динамічного завантаження та оновлення текстів локалізації дозволяє забезпечити швидке та зручне підключення до будь-якого фронтенд-додатку.

- **Формати зберігання даних (JSON, XML)**. Найбільш поширеним форматом для зберігання локалізованих текстів є JSON, оскільки він легко інтегрується з JavaScript, який широко використовується у веб-розробці.
- **Генеративні моделі штучного інтелекту для автоматизованого перекладу**. Застосування генеративних моделей ШІ, як-от ChatGPT чи Gemini, значно спрощує та прискорює процес перекладу контенту, роблячи його доступним для великої кількості мовних варіантів.
- **Інтерфейс для редагування та перевірки текстів**. Забезпечує зручний доступ для редагування локалізованих текстів, їх перегляду та внесення змін.

Серед основних викликів локалізації веб-додатків можна виділити такі:

- **Масштабованість**. Розширення кількості підтримуваних мов та регіонів є важливим, особливо для глобальних продуктів, і потребує адаптованої архітектури локалізації.
- **Автоматизація перекладу**. Незважаючи на розвиток генеративних ШІ, повна автоматизація перекладу залишається викликом, оскільки деякі аспекти перекладу вимагають людського втручання.
- **Інтеграція з іншими системами**. Локалізація повинна легко інтегруватися з різноманітними платформами та технологіями, такими як React, Node.js, чи серверні середовища.

Для кращого розуміння основних характеристик сучасних систем локалізації веб-додатків, була створена порівняльна таблиця з основними можливостями популярних рішень.

Таблиця 1.1

Порівняння найпопулярніших систем для локалізації

Назва системи	Підтримувані формати	Інтеграція з фреймворками	Використання генеративного ШІ	Особливості
Transifex	JSON, YAML, XML	React, Angular	Ні	Потужна система керування термінами та контролю версій
Phrase	JSON, iOS strings	React, iOS, Android	Так	Підтримка багатьох мов і API для інтеграції
Weblate	JSON, XML, PO	Django, React	Ні	Спільна робота над перекладами

Така порівняльна характеристика показує, що сучасні системи локалізації активно використовують автоматизацію та забезпечують інтеграцію з популярними фреймворками та форматами даних. Застосування генеративного ШІ дозволяє ще більше прискорити процеси перекладу, що є важливим фактором для динамічних веб-додатків.

Попри численні переваги сучасних систем локалізації, існують певні недоліки, які можуть обмежувати їхню ефективність. Наприклад, це висока

вартість впровадження та обслуговування. Багато систем локалізації, таких як Lokalise чи Phrase, вимагають платної підписки, що може стати значним фінансовим тягарем для стартапів або невеликих компаній. Окрім того, зростання кількості мов та користувачів призводить до збільшення витрат на обслуговування системи.

Іншим неприємним недоліком є обмежена інтеграція генеративного ШІ. Хоча деякі системи підтримують автоматизований переклад, такі можливості часто обмежені в гнучкості. Наприклад, інструменти можуть не підтримувати всіх мов або забезпечувати лише базовий рівень перекладу. Це може призвести до недостатньої точності перекладу, особливо в складних або контекстуально важливих випадках [2, с. 43].

Також мають місце труднощі у забезпеченні якості перекладу. Автоматичний переклад, навіть за допомогою генеративного ШІ, може мати недоліки, такі як неправильне тлумачення значення чи відсутність культурної адаптації. Це вимагає ручного перегляду та корекції перекладів, що збільшує витрати часу та зусиль. Нижче наведений графік, який відображає порівнянні якості перекладу різноманітних мовних моделей / сервісів.

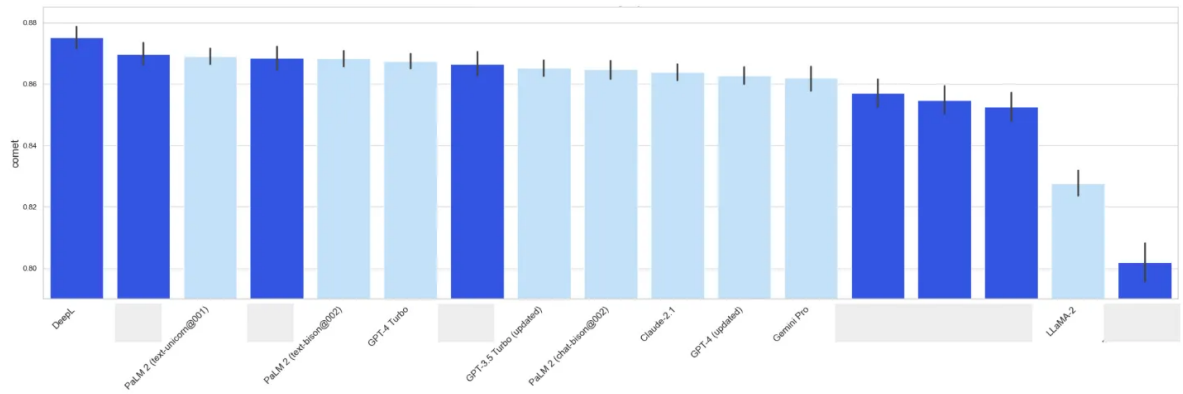

Рис 1.2. Порівняння якості перекладів тексту

Крім того, багато систем локалізації розроблені для певних мов і форматів файлів, і масштабування під час підтримки нових мов може стати

викликом. Інтеграція нових мов або складних регіональних налаштувань часто вимагає значних змін у коді чи структурі бази даних. Також існує проблема обмеження функціональності в безкоштовних версіях. Більшість популярних платформ, таких як Transifex чи Phrase, пропонують безкоштовні тарифи, які мають серйозні обмеження: обмежену кількість мов, обмежену кількість користувачів або обмеження на обсяг даних. Це може стримувати компанії від повного використання систем локалізації, оскільки для більшого функціоналу доведеться перейти на платний тариф.

Також, є два менш значущих недоліка - це залежність від хмарних рішень, та складність у інтеграції з індивідуальними системами. У першому випадку, багато платформ локалізації є хмарними сервісами, які залежать від інтернет-з'єднання. Це може створити проблеми для команд, що працюють в умовах обмеженого інтернету або з конфіденційною інформацією, де потрібна повна автономність.

Щодо інтеграції, деякі платформи, хоч і підтримують популярні фреймворки, можуть бути важко інтегровані в індивідуально розроблені системи, які мають унікальні вимоги. Це може вимагати створення додаткових API чи проміжного програмного забезпечення для забезпечення сумісності.

Вищенаведені недоліки вказують на те, що хоча сучасні системи локалізації пропонують безліч переваг для веб-додатків, важливо враховувати їхні обмеження під час вибору відповідної платформи для конкретного проекту.

1.2. Взаємозв’язок понять «JSON» та «API»

У сучасних веб-додатках локалізація відіграє вирішальну роль, забезпечуючи доступність продукту для користувачів з різних країн та мовних середовищ. Два ключові поняття, що лежать в основі такої системи локалізації, є JSON (JavaScript Object Notation) та API (Application Programming Interface). Розуміння взаємозв'язку між цими технологіями є критично важливим для розробки та підтримки локалізаційних рішень, особливо в умовах глобалізації цифрових продуктів.

JSON набув популярності завдяки своїй простоті, легкості використання та сумісності з більшістю мов програмування. Він є текстовим форматом для передачі структурованих даних, який має просту та логічну структуру з використанням ключів і значень, що дозволяє ефективно зберігати текстові локалізації. Для системи локалізації JSON забезпечує зручність і гнучкість при створенні ключів, які відповідають за конкретні мовні одиниці в інтерфейсі веб-додатку.

Ключовою перевагою JSON як формату зберігання даних є його здатність інтегруватися з більшістю систем, включаючи ті, які написані на відмінних від JavaScript мовах програмування - PHP, Python, Java і т.д. JSON файли можуть бути збережені на сервері або в локальному сховищі, де з них легко здійснювати запити API, що підвищує ефективність доступу до цих даних у веб-додатку [3, с. 43].

Рис 1.3. Передача JSON - даних від клієнту до серверу

JSON можна використовувати при написанні асинхронних веб-додатків, які обмінюються даними з сервера на веб-сторінку і потребують дуже швидкого доступу до даних - наприклад, веб-сайти з динамічним контентом, який повинен оновлюватися на основі введення даних користувачем, або пропозиції на основі користувацьких уподобань.

Синтаксис JSON передбачає наступне:

- Об'єкти JSON записуються у фігурних дужках {curl}.
- Кожен елемент є парою ключ-значення.
- Ключі та значення рядкового типу записуються у подвійних лапках. Інші типи даних, такі як цілі та логічні, не потрібно брати в лапки.
- Кожен елемент відокремлюється від наступного за допомогою коми (,). Після останнього елемента кома не ставиться.
- Масиви всередині JSON-рядків записуються у квадратних дужках.
- Об'єкти і масиви можуть бути вбудовані в об'єкт

На відміну від об'єктів JavaScript, JSON не може приймати функції, тип дати та невизначений тип. Дати можна конвертувати в рядок у форматі ISO і зберігати.

Приклад JSON об’єкту з даними про певну країну на континенті:

```
{
  "continentName": "North America",
  "area": 24.71,
  "countries": [
    {
      "countryName" : "Mexico",
      "countryCode" : "+52",
      "location" : "south of north america",
      "languagesSpoken" : ["spanish", "maya", "nahuatl"]
    }
  ]
}
```

API (що розшифровується як application programming interface) - це набір протоколів, які дозволяють різним програмним компонентам спілкуватися та передавати дані. Розробники використовують API для подолання розривів між невеликими дискретними фрагментами коду, щоб створювати потужні, стійкі, безпечні додатки, здатні задовольнити потреби користувачів. API є скрізь - вони безперервно працюють у фоновому режимі, щоб забезпечити цифровий досвід, який є невід'ємною частиною сучасних веб - додатків [4, с. 43].

API працюють, обмінюючись даними між програмами, системами та пристроями. Це відбувається через цикл запитів і відповідей. Запит надсилається до API, який отримує дані і повертає їх користувачеві. Ось загальний огляд того, як працює цей процес.

Клієнт API відповідає за початок розмови шляхом надсилання запиту на сервер API. Запит може бути ініційований різними способами. Наприклад, користувач може ініціювати API-запит, ввівши пошуковий запит або натиснувши кнопку. Запити до API також можуть бути ініційовані зовнішніми подіями, наприклад, сповіщенням від іншої програми.

Запит до API буде виглядати і поводитися по-різному в залежності від типу API, але зазвичай він включає в себе наступні компоненти:

1. **Кінцева точка API** - це спеціальна URL-адреса, яка надає доступ до певного ресурсу. Наприклад, кінцева точка /articles у додатку для ведення блогу міститиме логіку обробки всіх запитів, пов'язаних зі статтями.
2. **Метод запиту** вказує на тип операції, яку клієнт хоче виконати над певним ресурсом. REST API доступні за допомогою стандартних HTTP-методів, які виконують звичайні дії, такі як отримання, створення, оновлення та видалення даних.
3. **Параметри** - це змінні, які передаються кінцевій точці API для надання конкретних інструкцій для обробки API. Ці параметри

можуть бути включені в запит до API як частина URL-адреси, в рядок запиту або в тіло запиту. Наприклад, кінцева точка /articles API для блогів може прийняти параметр «topic», який буде використовуватися для доступу і повернення статей на певну тему.

4. **Заголовки запиту** - це пари ключ-значення, які надають додаткову інформацію про запит, наприклад, тип вмісту або облікові дані для автентифікації.
5. **Тіло (body)** - це основна частина запиту, яка містить фактичні дані, необхідні для створення, оновлення або видалення ресурсу. Наприклад, якщо ви створюєте нову статтю в додатку для ведення блогу, тіло запиту, найімовірніше, міститиме вміст статті, її назву та автора.

Клієнт API надсилає запит на сервер API, який відповідає за обробку автентифікації, перевірку вхідних даних, а також отримання або маніпулювання даними. Нарешті, сервер API надсилає відповідь клієнту. Зазвичай відповідь API містить такі компоненти:

1. **Коди статусу HTTP** - це тризначні коди, які вказують на результат виконання запиту API. Деякі з найпоширеніших кодів статусу включають 200 OK, що означає, що сервер успішно повернув запитувані дані, 201 Created, що означає, що сервер успішно створив новий ресурс, і 404 Not Found, що означає, що сервер не зміг знайти запитаний ресурс.
2. **Заголовки** - використовуються для надання додаткової інформації про відповідь сервера.
3. **Тіло відповіді** містить фактичні дані або вміст, які запитував клієнт, або повідомлення про помилку, якщо щось пішло не так.

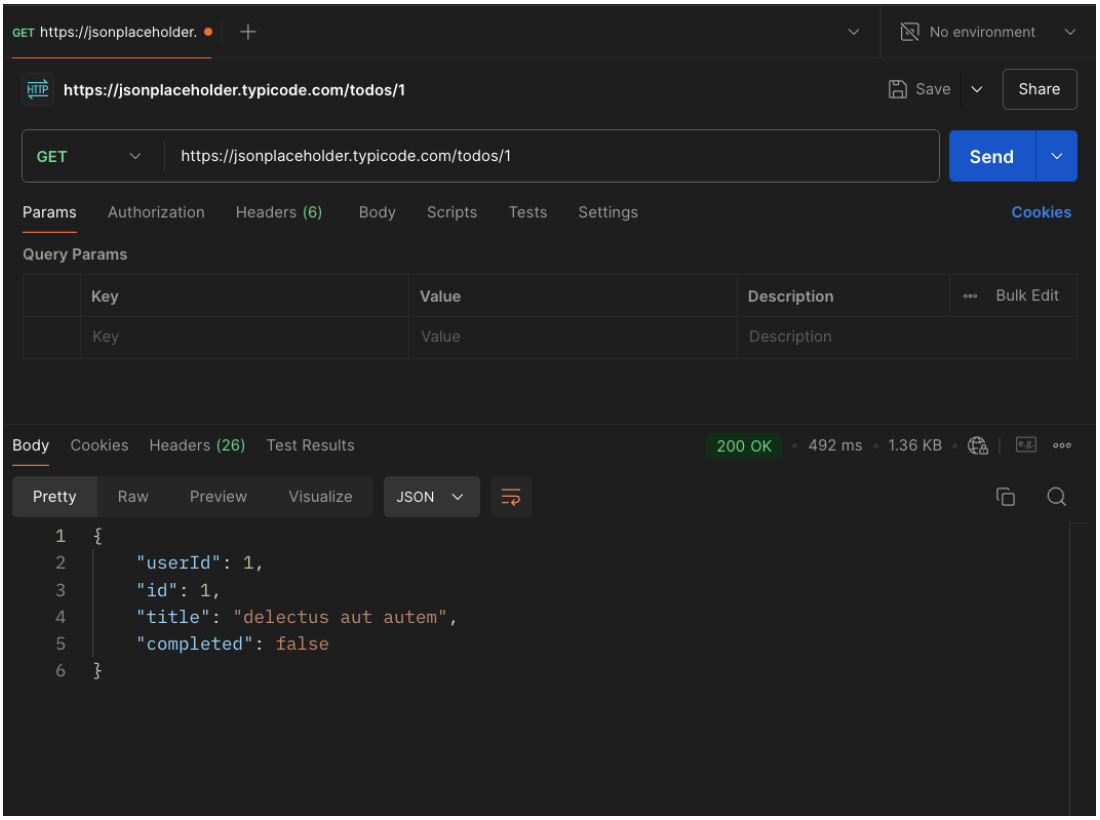

Рис 1.4. Приклад запиту до безкоштовного API

У наведеному прикладі, здійснено API виклик до кінцевої точки https://jsonplaceholder.typicode.com/todos/1. Запит - GET, без додаткових параметрів чи тіла запиту. У відповіді був отриманий JSON - об’єкт з полями userId, id, title та completed.

У контексті системи локалізації веб - додатків, JSON забезпечує чітку структуру даних, легко інтегрується з API та дозволяє організовувати переклади у вигляді зрозумілих ключів і значень. API забезпечує простий і швидкий доступ до цих даних, підтримуючи їхню актуальність і доступність для користувачів. Така інтеграція також знижує потребу в постійному оновленні фронтенду та дозволяє використовувати автоматизовані системи перекладу.

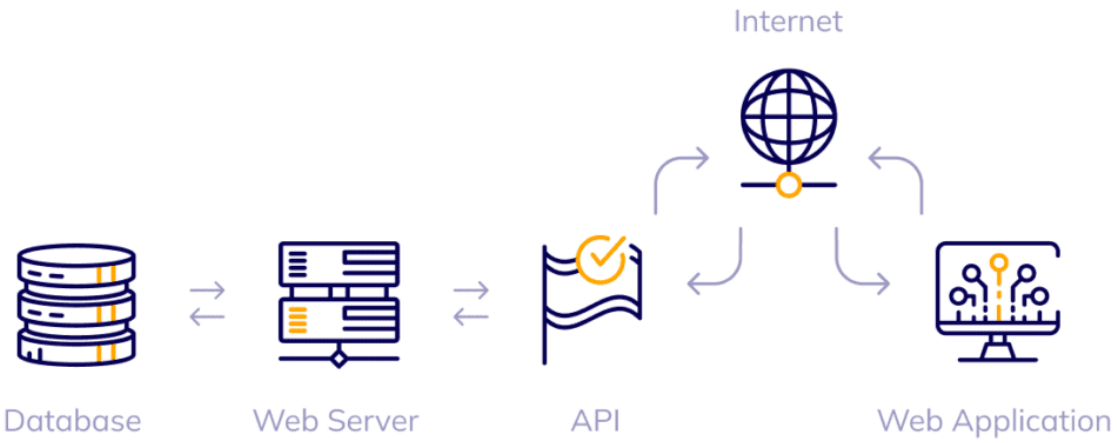

Рис 1.5. Приклад найпростішої клієнт - серверної взаємодії

Висновок по розділу 1

У цьому розділі розглянуто основні теоретичні аспекти системи локалізації веб-додатків. Локалізація є ключовим процесом для забезпечення доступності веб-продуктів для користувачів з різних культурних та мовних середовищ. Адаптація контенту, інтерфейсу та врахування культурних особливостей дає змогу значно розширити географічну аудиторію та покращити користувацький досвід. Проте, процес локалізації має свої складнощі, зокрема у забезпеченні ефективної автоматизації та інтеграції з іншими технологіями, а також у мінімізації витрат на обслуговування та обмежень, що виникають при використанні певних систем.

Застосування генеративних моделей штучного інтелекту, таких як ChatGPT, є значним кроком у напрямку автоматизації перекладу, що прискорює процес локалізації, хоча й має обмежену гнучкість у деяких аспектах. Вибір системи локалізації для конкретного проекту має враховувати не тільки її можливості, але й обмеження, зокрема в контексті вартості, інтеграції з індивідуальними системами та якості перекладу.

Основні компоненти системи локалізації, такі як API для отримання текстів, зберігання даних у форматах JSON та інтеграція з фреймворками,

забезпечують ефективну реалізацію локалізаційних рішень. Розуміння взаємозв'язку між JSON та API є критичним для успішної інтеграції та підтримки локалізації веб-додатків у динамічному середовищі сучасних технологій.

З огляду на швидкий розвиток цієї сфери, важливо продовжувати вдосконалювати системи локалізації, що дозволить створювати інтернаціоналізовані веб-продукти високої якості, доступні для глобальних користувачів.

РОЗДІЛ 2. БІБЛІОТЕКИ ЛОКАЛІЗАЦІЇ WEB-ДОДАТКІВ

2.1. Фреймворк інтернаціоналізації «I18next»

i18next – це потужний JavaScript-фреймворк для інтернаціоналізації, що забезпечує гнучкі та масштабовані рішення для локалізації веб- та мобільних додатків. Його система підтримує різні платформи, такі як React, Angular, Vue.js, Node.js, а також мобільні платформи (iOS, Android). Основні можливості включають автоматичне визначення мови користувача, завантаження та кешування перекладів, а також обробку складних форматів [5, с. 43].

Завдяки модульній структурі та розширенням, i18next підходить для великих проєктів, дозволяючи розділяти переклади на кілька файлів і завантажувати їх за потреби. Інтеграція з locize пропонує керування перекладами як послугу, що дозволяє легко підтримувати актуальні переклади [6, с. 43].

Рис 2.1. Підтримувані I18next платформи

Основні переваги **I18next:**

Стабільність. i18next було створено наприкінці 2011 року, це означає що всі функціональні речі системи уже давно реалізовані та перевірені. Кількість багів або неочікуваних проблем є мінімальною.

Розширюваність. З другої версії i18next розробники повністю перебудували i18next, щоб бути максимально розширюваним. Роблячи це, i18next можна використовувати в будь-якому середовищі javascript (і в деяких без JavaScript - .NET, iOS, Android, тощо.) з будь-якою структурою інтерфейсу користувача, з будь-яким форматом i18n.

Функціональність. Коли звичайні фреймворки i18n працюють так:

- Ви передаєте всі переклади та використовувану мову
- Ви викликаєте функцію, яка повертає правильний переклад на основі переданих вами перекладів і наданих значень для множини та інтерполяції.

i18next працює наступним чином:

- Переклади розбиваються на кілька файлів. Потрібно лише завантажити текст.
- Існують плагіни для визначення мов для більшості середовищ (браузер, сервер, мобільні пристрої). Це дозволяє встановити пріоритет місця виявлення та навіть дозволяє кешувати встановлені мови над запитами/відвідуваннями.
- Існують багато плагінів для завантаження перекладу з сервера, файлової системи, тощо. Серверні модулі можуть навіть забезпечити додатковий рівень для локального кешування, наприклад. у localStorage.
- Підтримка об'єктів і масивів
- Повний контроль над керуванням збереженими перекладами.
- Багата система подій для реагування на зміни.

Мінуси I18next:

- Складність. Документація I18next є доволі великою, і реалізація підключення такого рішення до своєї системи може бути складним завдання в силу великої кількості функціональних можливостей бібліотеки.
- Перевантаженість. Хоча широкий спектр функціональних можливостей і є плюсом, але це також є проблемою для малих проектів та всі можливості не потрібні. Це створює додаткове навантаження на розробника, і в кінцевому результаті бібліотека буде використана не на всі 100%.

Враховуючи результати аналізу, можна зробити висновок, що i18next, попри свою функціональну гнучкість та модульну структуру, може бути недостатньо оптимальним рішенням для систем локалізації, де важливими є простота та мінімальна витрата ресурсів. Основні виклики, пов’язані з i18next, зумовлені його багатофункціональністю, що додає складності інтеграції та збільшує потребу в налаштуванні для ефективного використання.

По-перше, велика кількість можливостей i18next, таких як визначення мов користувача, завантаження та кешування перекладів, керування форматуванням і множиною, а також підтримка різних плагінів, забезпечує потужний інструмент для локалізації. Однак це призводить до збільшення обсягу коду і часу, необхідного для налаштування. Для проєктів, що вимагають простоти та швидкості, значна кількість додаткових функцій i18next може бути зайвою та обтяжливою. Інтеграція з платформами, такими як locize для централізованого керування перекладами, додатково ускладнює архітектуру проєкту, що може вимагати серйозних технічних знань від розробників.

По-друге, через складність документації i18next та багатоступеневий підхід до локалізації, цей фреймворк може стати перешкодою для команд, яким потрібно швидке рішення без значних витрат на навчання та

підтримку. Навіть досвідченим розробникам може знадобитися час на ознайомлення з усіма функціональними можливостями i18next, що не завжди відповідає вимогам проєктів, які прагнуть швидкої адаптації локалізації без зайвих витрат на вивчення.

З іншого боку, система локалізації веб-додатків, орієнтована на простоту і швидкість, могла б виграти від мінімалістичного підходу. Така система не повинна бути перенасичена функціями, які можуть залишитися невикористаними у більшості проєктів. Замість цього, вона повинна забезпечувати базові можливості інтернаціоналізації, достатні для ефективної локалізації без надмірної обтяженості, якої часто не потребують невеликі команди та маломасштабні проєкти.

Таким чином, використання i18next виправдане в проєктах, де необхідно забезпечити масштабованість і багатофункціональність, але для системи локалізації, яка має бути простою, швидкою та легко підтримуваною, такий підхід може бути надлишковим і потребуватиме великих технічних ресурсів. З урахуванням потреб надійної локалізації для швидкого реагування на зміни, варто розглянути рішення, які б не мали такої складності налаштування та функціонального перевантаження, що характерні для i18next.

2.2. Polyglot як рішення інтерполяції

Polyglot.js – це компактна і ефективна бібліотека для локалізації, створена з акцентом на мінімалізм та простоту. Вона ідеально підходить для невеликих веб-додатків, де основною метою є простий і швидкий процес локалізації. На відміну від масштабних фреймворків, таких як i18next, Polyglot.js надає базові функціональні можливості та сфокусована на обмеженому наборі задач: форматування тексту, підтримка множини, динамічна інтерполяція значень, і простий механізм для обробки відсутніх перекладів [7, с. 43].

Рис 2.2. Приклад використання Polyglot.js

Polyglot.js відома своєю невеликою вагою та швидкістю роботи, що мінімізує навантаження на систему і забезпечує швидке завантаження сторінок. Це особливо важливо для проєктів, які мають мінімальні вимоги до локалізації та працюють на обмежених ресурсах або прагнуть забезпечити високу продуктивність. Polyglot.js не вимагає значної кількості зовнішніх залежностей, що також спрощує процес її інтеграції. Крім того,

вона надає основні функції для налаштування множини та інтерполяції змінних, що є достатнім для більшості невеликих застосунків.

Переваги Polyglot:

- Підтримка множини та інтерполяції. Однією з ключових функцій є можливість працювати з множиною, що дозволяє автоматично відображати різні текстові форми, залежно від значень змінних. Наприклад, для різних числових значень можуть відображатися різні варіанти тексту, що є корисним при локалізації для мов з різними правилами множини. Крім того, функція інтерполяції змінних дає можливість динамічно вставляти значення в текстові рядки.

- Контроль над відсутніми перекладами. Polyglot.js надає функцію зворотного виклику для обробки відсутніх перекладів, що дозволяє розробникам легко контролювати процес локалізації та оперативно оновлювати відсутні переклади. Це особливо корисно на етапі розвитку додатка, коли нові рядки можуть з’являтися в інтерфейсі часто, і потрібно відслідковувати їх відсутність.

- Простота інтеграції. Завдяки мінімальному коду та низькій кількості функцій, Polyglot.js легко інтегрувати в прості проєкти. Бібліотека не має складної архітектури, що значно полегшує процес її впровадження. Для локалізації, яка передбачає лише переклад фраз і множини, цей інструмент забезпечує всі необхідні можливості.

Недоліки та обмеження Polyglot:

- Обмежений функціонал. Polyglot.js не підтримує завантаження перекладів із сервера та автоматичне визначення мови користувача, що значно обмежує її застосування для великих проєктів. На відміну від i18next, який дозволяє завантажувати переклади за запитом і адаптуватися до різних мовних налаштувань користувача, Polyglot.js є більш статичним рішенням, що підходить лише для базових сценаріїв.

- Відсутність розширюваності. Polyglot.js не передбачає можливості інтеграції з іншими інструментами або додавання нових функцій через плагіни. Це робить її менш універсальною, що обмежує її корисність у випадках, коли локалізація потребує спеціальних налаштувань або інтеграції з додатковими сервісами. Відсутність гнучкості робить її придатною лише для проєктів з невеликими вимогами до локалізації.

Polyglot.js є надто простим рішенням для розширених потреб локалізації веб-додатків, оскільки не надає ключових функцій, необхідних для повноцінної підтримки багатомовності у великих проєктах. Наприклад, відсутність можливості локально завантажувати файли JSON з перекладами значно обмежує процес оновлення і зберігання локалізованих даних. Крім того, Polyglot.js не підтримує вкладені ключі, що ускладнює роботу з великим обсягом тексту в складних проєктах, де кожен елемент може мати свої підкатегорії для різних локалей. Це рішення також не дозволяє перекладати весь проєкт на інші мови централізовано, що обмежує його функціональність у порівнянні з більш просунутими бібліотеками, такими як i18next.

Через ці обмеження Polyglot.js підходить лише для мінімальних потреб локалізації у проєктах з фіксованою структурою і невеликою кількістю текстових ресурсів. У випадках, коли проєкт вимагає комплексної системи для обробки динамічних перекладів та централізованого управління мовами, використання Polyglot.js стає недоцільним. Для забезпечення високої масштабованості, стабільності та гнучкості, у таких проєктах доцільніше обирати рішення, що підтримують локальне завантаження файлів, вкладені ключі, автоматичне визначення мов користувачів і можливість централізованого управління перекладами.

Висновок по розділу 2

Огляд бібліотек локалізації веб-додатків у цьому розділі показав їхні ключові особливості, переваги та обмеження, що впливають на вибір інструменту залежно від масштабів і потреб проекту.

Бібліотека i18next демонструє високий рівень функціональності та гнучкості, що робить її оптимальним вибором для великих і комплексних проектів. Її підтримка різних платформ, модульна структура, автоматизація мовного визначення та управління перекладами забезпечують повний набір інструментів для професійної локалізації. Водночас, висока складність документації та перенасиченість функціями створюють виклики для інтеграції та обслуговування, особливо в невеликих командах або проєктах із базовими потребами в локалізації.

На противагу цьому, Polyglot.js зосереджена на мінімалізмі та простоті. Її легкість інтеграції, швидкодія та базові можливості інтерполяції роблять її ідеальним рішенням для мало масштабних проектів. Однак обмежений функціонал, відсутність розширюваності та підтримки динамічного завантаження перекладів знижують її придатність для великих проектів або середовищ із високими вимогами до локалізації.

Таким чином, вибір між i18next та Polyglot.js залежить від потреб конкретного проекту: для великих і масштабованих систем рекомендовано використовувати i18next, тоді як Polyglot.js підходить для невеликих проектів із мінімальними вимогами. Обидві бібліотеки відіграють важливу роль у сфері локалізації, але їхні особливості визначають різний підхід до інтеграції та використання у веб-додатках.

РОЗДІЛ 3. РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ЛОКАЛІЗАЦІЇ WEB-ДОДАТКІВ

3.1. Розробка проекту ІС

Проектування та розробка системи локалізації web-додатків у є завданням, яке вимагає уважного вибору засобів та методів розробки для досягнення оптимальної функціональності, продуктивності та користувацької зручності. У цьому розділі роботи буде розглянуто важливий аспект – вибір технологічного стеку, який включає у себе Typescript, React, NodeJS та MongoDB.

React – це відкритий фреймворк JavaScript, створений компанією Facebook, який здобув велику популярність серед розробників. Використовується для ефективної розробки інтерфейсів користувача веб-додатків, надаючи високу продуктивність і масштабованість. Однією з ключових концепцій є компоненти, які є незалежними блоками коду для рендерингу частини інтерфейсу.

React Frontend спрямований на створення інтерактивних, динамічних та відзивчивих інтерфейсів, забезпечуючи швидкий рендеринг та перехід між сторінками. Він використовує віртуальний DOM та ефективний алгоритм оновлення для зменшення впливу на реальний DOM, що підвищує продуктивність.

Заснований на компонентній архітектурі, React спрощує розробку, тестування та підтримку коду, дозволяючи повторне використання та легку зміну компонентів. Використання одностороннього потоку даних сприяє простоті управління станом додатків.

Завдяки активній спільноті розробників і великій кількості ресурсів, React добре підходить для проектів різного масштабу. Підтримка серверного рендерингу поліпшує швидкість завантаження сторінок. Загалом, React дозволяє ефективно будувати потужні та швидкі інтерфейси,

полегшує роботу з компонентами та станом додатків і має широку підтримку спільноти розробників, що робить його ідеальним для командної розробки.

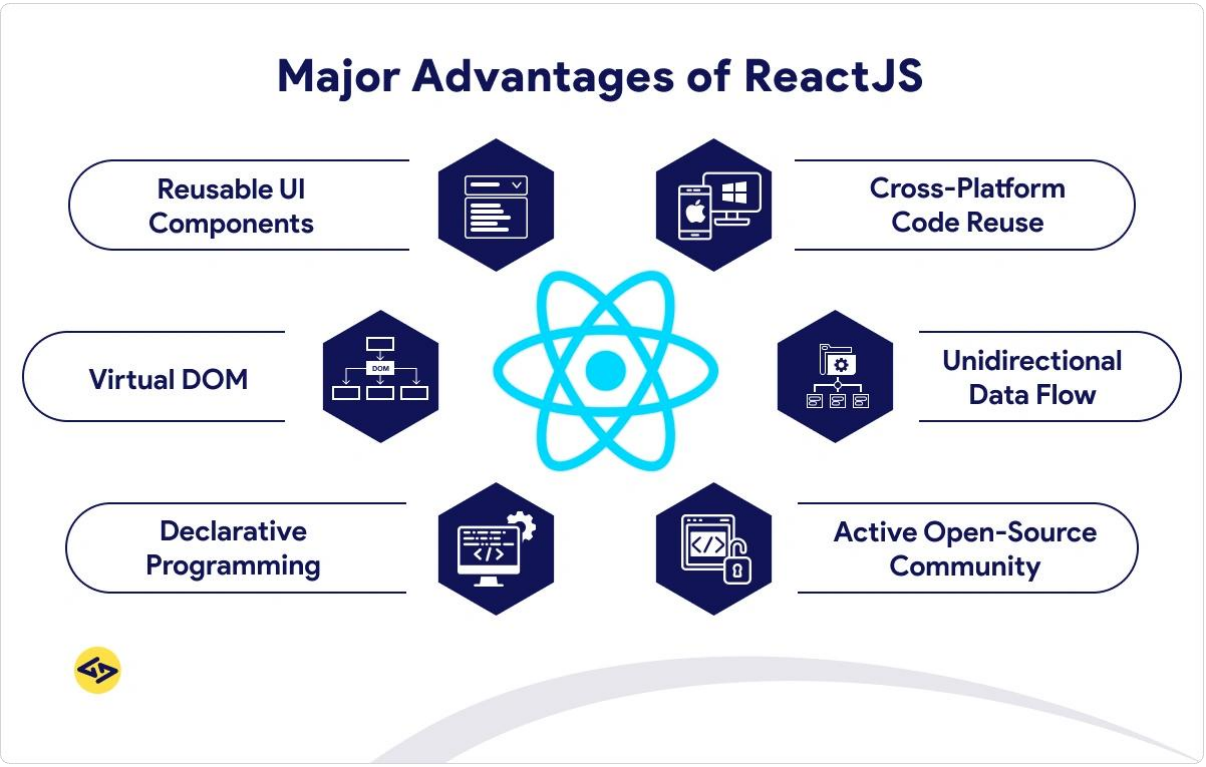

Рис 3.1 Переваги використання ReactJS

Node.js є відкритою платформою, створеною Раяном Далом, для виконання високопродуктивних мережевих застосунків, які написані мовою JavaScript. Перш ніж з'явився Node.js, JavaScript використовувався виключно для обробки даних в браузерах, але ця платформа розширила його можливості, дозволяючи виконувати JavaScript-скрипти на сервері.

Основні характеристики Node.js включають асинхронну однопоточну модель виконання запитів, неблокуючий ввід/вивід, використання системи модулів CommonJS та використання рушія JavaScript Google V8. Ці особливості сприяють високій продуктивності та швидкості виконання застосунків.

Node.js також використовує пакетний менеджер npm (Node Package Manager) для управління модулями. Завдяки цьому, розробники можуть

легко встановлювати, оновлювати та видаляти залежності в їхніх проектах, що сприяє ефективному розвитку та управлінню кодом.

Загалом, Node.js перетворив JavaScript на загальнопризначену мову програмування з великою та активною спільнотою розробників, відкривши нові можливості для розробки серверних застосунків та мережевих додатків.

MongoDB – це система керування базами даних (СКБД) з відкритим вихідним кодом, яка відрізняється документо-орієнтованим підходом та відсутністю потреби у попередньому описі схеми таблиць. Розроблена мовою C++ і поширюється під ліцензією AGPLv3.

Основні особливості MongoDB включають підтримку зберігання документів у форматі JSON, гнучку мову для формування запитів, можливість створення індексів для різних атрибутів, а також ефективне зберігання великих бінарних об'єктів. Система забезпечує журналювання операцій, підтримує реплікацію, може використовуватися за парадигмою Map/Reduce, і має вбудовані засоби для забезпечення шардінгу, що дозволяє побудувати горизонтально масштабований кластер зберігання.

MongoDB є крос-платформною і легко масштабується, складаючись з баз даних, які містять колекції. Кожна колекція містить документи, які, у свою чергу, складаються з полів у формі пар ключ-значення. Можливість індексації колекцій покращує швидкість вибірки та сортування даних. MongoDB також надає можливість розширення кластера та перетворення серверів без зупинки роботи бази даних.

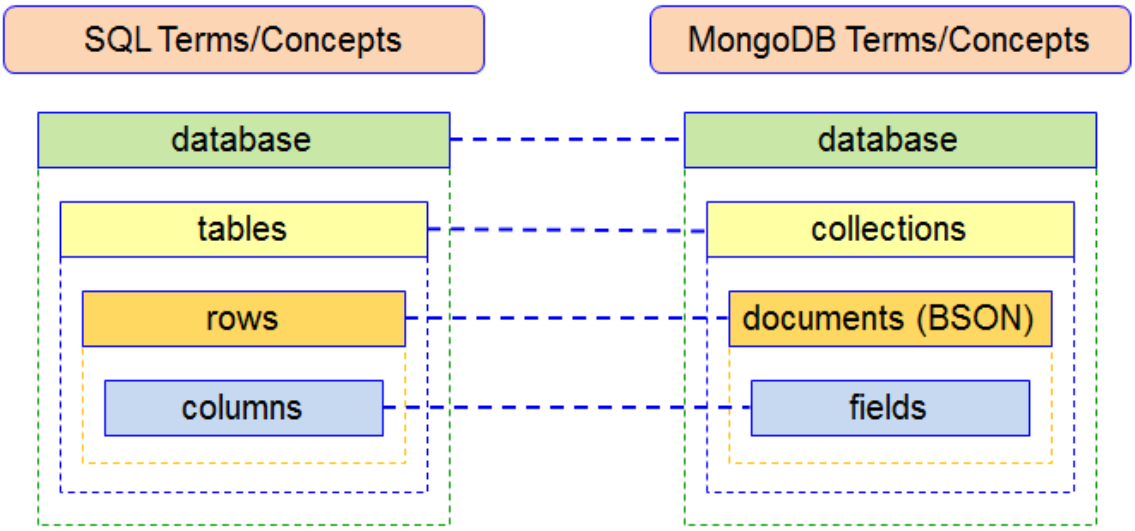

Рис 3.2. Порівняння MongoDB та SQL

Система локалізації веб додатків буде мати можливість можливість здійснити вхід для користувачів, які будуть мати розмежування доступу на основі їхньої ролі. Frontend буде оснащений інтерфейсом для перегляду інформації про проекти, а також для додавання та редагування інформації ключів локалізації.

З правами адміністратора передбачено можливість редагування та додавання нових записів в базу даних. У випадку змін чи додавання інформації на веб-сайті, ці зміни будуть автоматично фіксуватися в базі даних і відображатимуться для інших користувачів.

Основним завданням цього проекту є забезпечення миттєвого доступу користувачів до актуальної версії перекладу того чи іншого локалізаційного ключа. Для цього передбачено реалізацію зв'язків між таблицями в базі даних, щоб забезпечити коректне відображення та обробку даних щодо користувачів та дані локалізації.

Оскільки в якості бази даних використовується NoSQL база даних MongoDB, певної структури та зв’язків між таблицями не буде. Для стандартизації даних буде створена схема для кожного типу - будь то

користувач чи об’єкт з інформацією про бронювання

Для визначення схеми користувача був написаний наступний код:

```
const UserSchema = new Schema({
  id: { type: String, required: false },
  email: { type: String, required: true },
  password: { type: String, required: true },
  datetime: { type: Date, default: Date.now },
  role: { type: Number, required: true, default: 0 },
});
```

Дані про користувача містять поля ідентифікатору, електронної адреси, паролю, часової мітки та ролі. ID користувача генерується автоматично, тому потреби вказувати його немає.

В базі даних MongoDB це виглядає наступним чином:

Рис 3.3. База даних MongoDB, документ - users

Дані про проекти та локалізаційні ключі будуть знаходитись в іншому документі - *projects*. Схема даних елементу projects виглядає так:

```
const ProjectSchema = new Schema({
  id: { type: String, required: false },
  name: { type: String, required: true },
  datetime: { type: Date, default: Date.now },
  translations: [{
```

```
    id: { type: String, required: false },
    language: { type: String, required: false },
    text_keys: [{
      variant: { type: String, required: false },
      keys: { type: Array, required: false }
    }],
  }],
});
```

Схема має наступні поля - id прокету, назву, часову мітку, та вкладений об’єкт з перекладами який містить id кожної мови на яку перекладено текст, код мови та вкладений масив з ключами.

Рис 3.4. База даних MongoDB, документ - projects

Результатом аналізу технологічних засобів та методів, а також проектування логічної моделі даних, було обрано найбільш оптимальні інструменти для розробки системи локалізації веб - додатків, та обґрунтована доцільність їх використання.

3.2. Програмна реалізація системи з використанням генеративного ШІ

Вперше відкривши систему, користувачу потрібно буде авторизуватись. Логіка системи передбачає, що для користувача попередньо був створений аккаунт системним адміністратором, отже він може авторизуватись за наданим йому логіном та паролем.

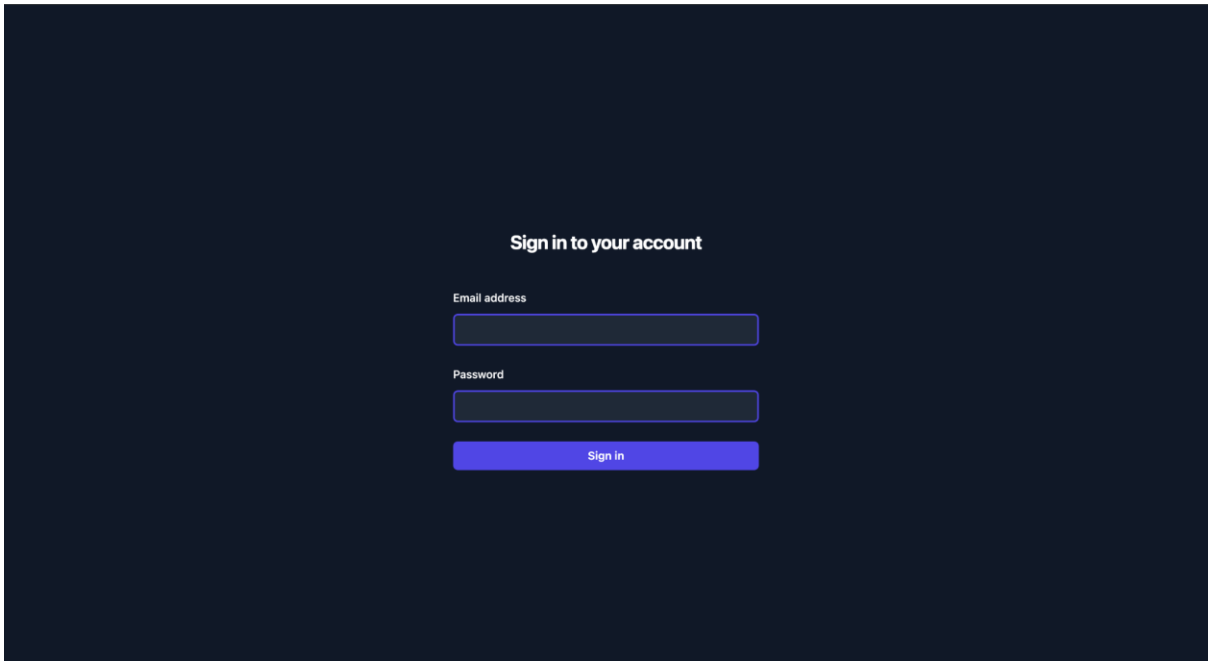

Рис 3.5. Екран авторизації додатку

Після успішної авторизації, користувачу відкриється доступ до сторінки Project, яка містить інформацію про всі активні наразі проекти, а також їх короткий опис та / або деталі.

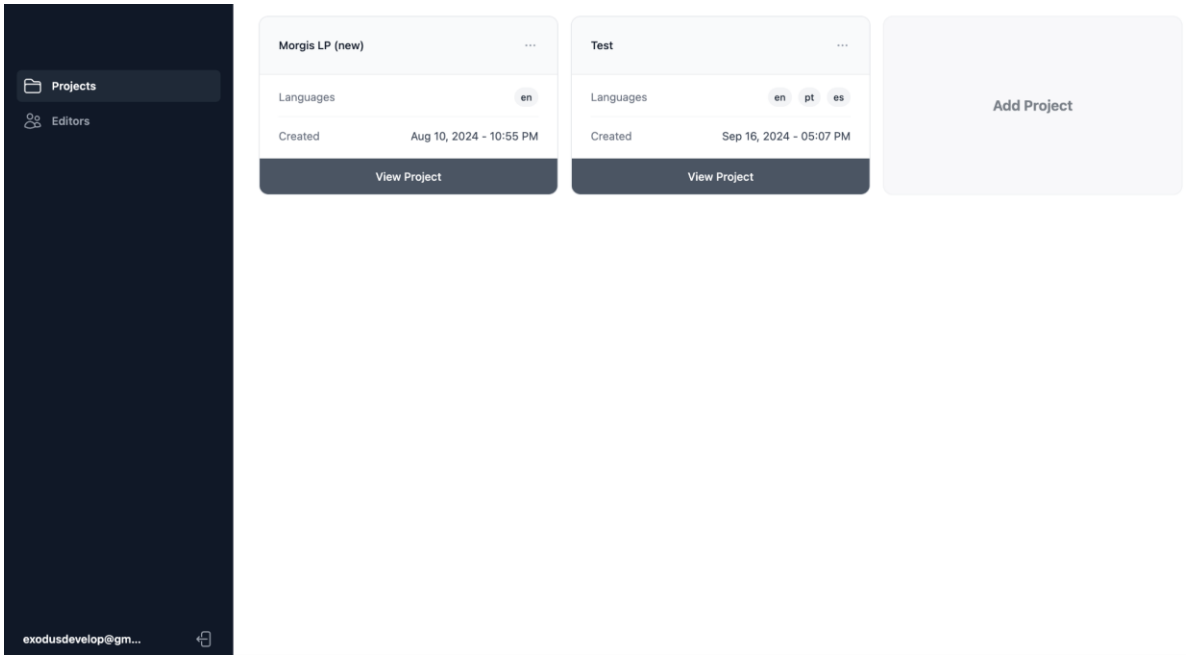

Рис 3.6. Сторінка проектів

Список проектів побудований на основі grid властивостей CSS. Це виглядає наступним чином:

```
<div className='grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4'>
        {
          projects.map((project) => (
            <ProjectCard key={project.id} project={project} />
          ))
        }
        {
          user?.role === 2 || user?.role === 3 ? <AddProject /> : null
        }
    </div>
```

Де *ProjectCard* та *AddProject* - компонент картки проекту та картки додавання проекту відповідно. Всі проекти завантажуються з API, та додаються у стан project, після чого за допомогою методу map - масив перебирається, та кожна картка рендериться з відповідною інформацією.

Приклад компоненту ProjectCard можна знайти в Додатку А.

Крім того, оскільки мати можливість додавати проекти мають лише користувачі з певними ролями (у нашому випадку це роль 2 та 3), додано перевірку на наявність у користувача відповідної ролі.

З боку бекенду GET запит на отримання списку проектів виглядає наступним чином

```
app.get('/project/all', checkAuth, async (req, res) => {
    await ProjectController.getProjects(req, res);
});
```

Рис 3.7. Логіка маршруту для отримання списку проектів

В свою чергу, після надсилання запиту за даним маршрутом з боку клієнту, запускається метод getProject, який обирає з бази даних MongoDB необхідні записи

```
async getProjects(req, res) {
    try {
        let projects = await this.project.find().select('-_id -__v');
        res.status(200).json(projects);

    } catch (error) {
        console.error('Error when getting projects', error);
        res.status(500).json({ message: 'Internal server error' });
    }
}
```

Рис 3.8. Логіка контроллеру для вибору проектів з БД

Якщо користувач має відповідні права доступу, він може додати новий проект, натиснувши кнопку «Add Project». Після цього, відкриється модальне вікно де користувачеві пропонується ввести ім’я проекту:

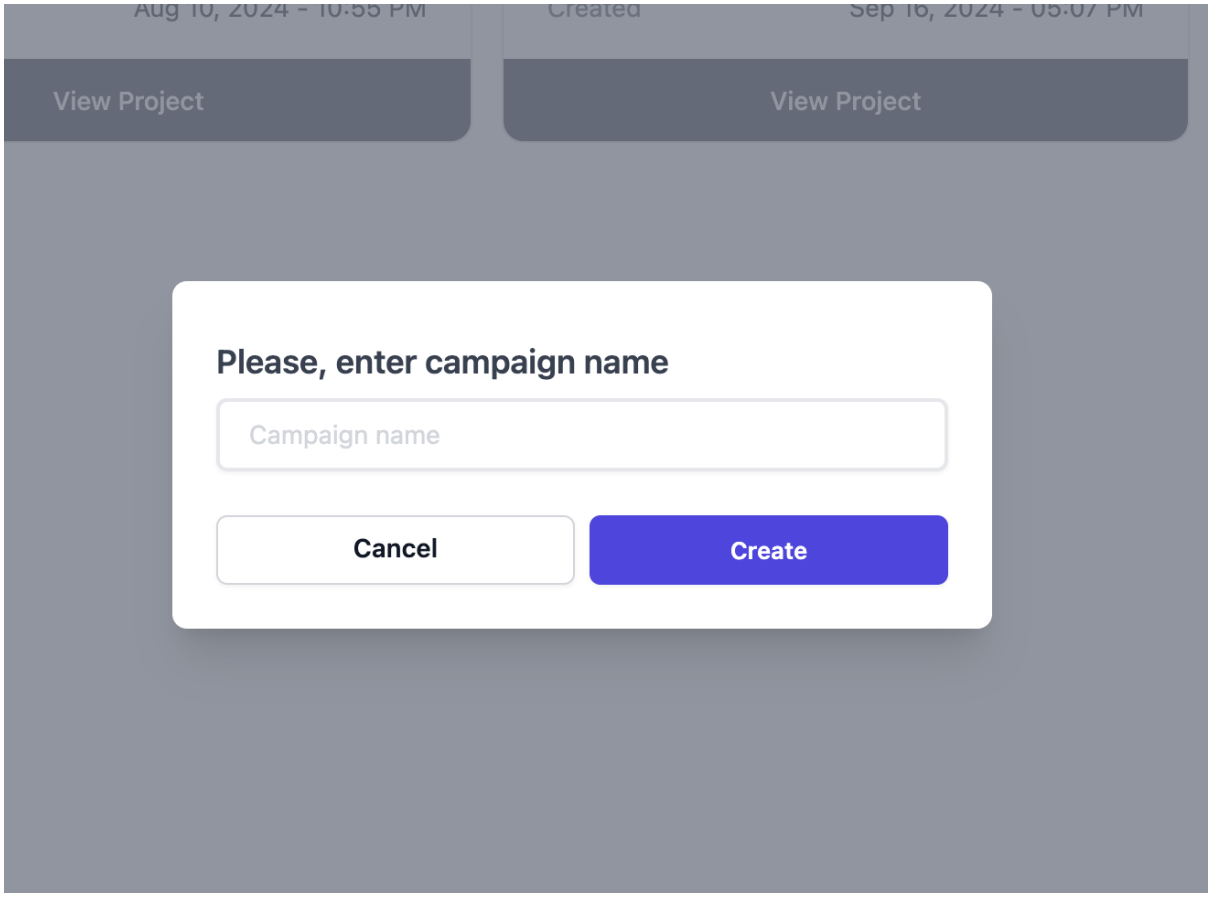

Рис 3.9. Модальне вікно додавання проекту

Після створення проекту та відповіді від бекенду яка містить id, користувач автоматично перенаправляється на динамічний маршрут {hostname}/projects/:id, на якому міститься основна логіка додатку.

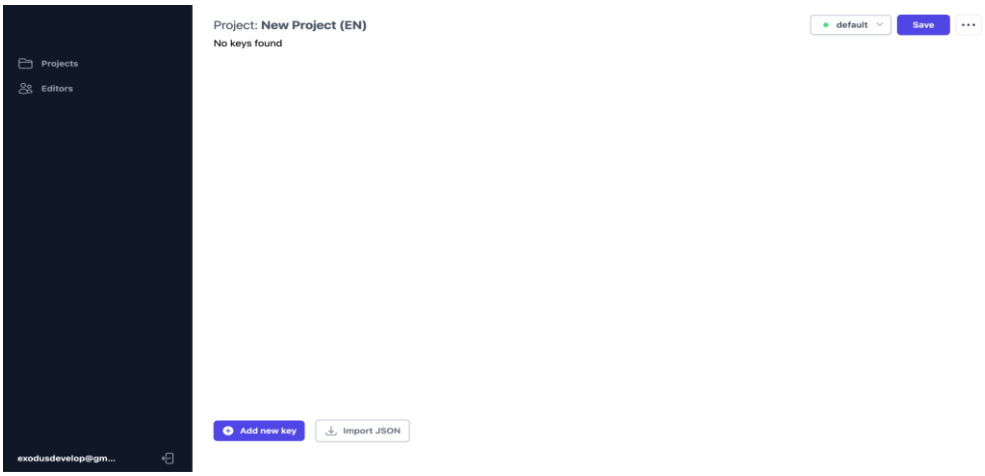

Рис 3.10. Основна логіка додатку

Тут користувач має можливість додати новий ключ з текстом, в тому числі ключ який має певну вкладеність. Крім того, користувач може змінити активну мову, створити новий переклад який автоматично перекладає весь наявний текст з англійської мови, на обрану. До прикладу, заповнимо ключі наступним чином:

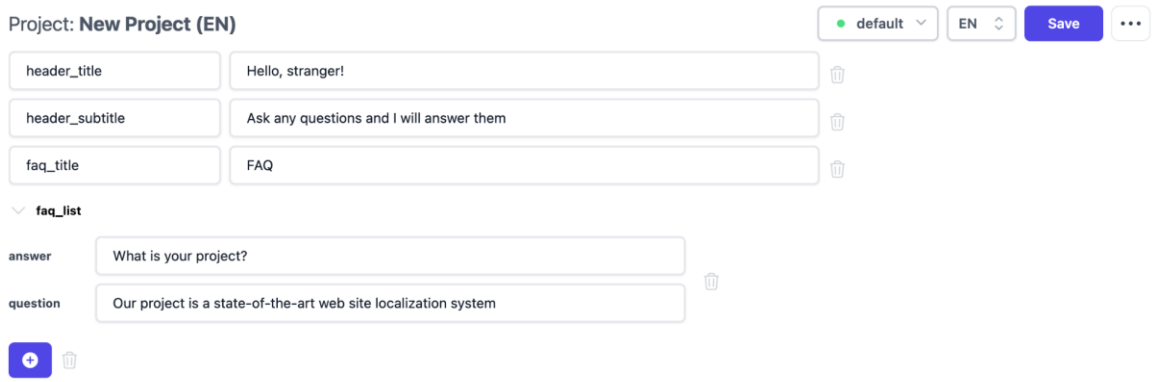

Рис 3.11. Заповнені дані проекту (ключі)

Після цього у користувача є можливість створити переклад для цього тексту на іншу мову. Потрібно відкрити меню, та обрати із випадаючого списку «Add new language». Після цього, нам запропонують деякі додаткові опції.

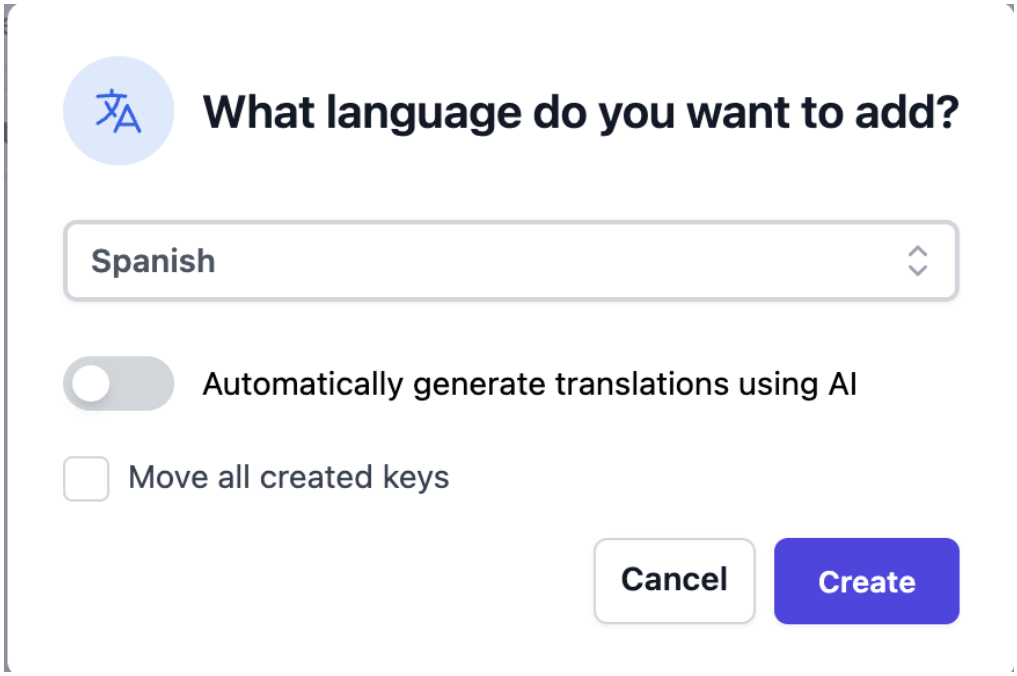

Рис 3.12. Модальне вікно створення перекладу

Якщо оберемо «Automatically generate translations using AI», локалізаційні ключі будуть перетворені у певний формат та відправлені до ChatGPT для перекладу на мову, вказану у prompt. Якщо ж вибрати «Move all created keys» (опція доступна для вибору лише якщо попередня - не активна) створюється нова мова з усіма ключами, але без тексту який відповідає кожному ключу.

Після завершення процесу перекладу який зайняв близько однієї секунди, ми отримаємо наступний результат:

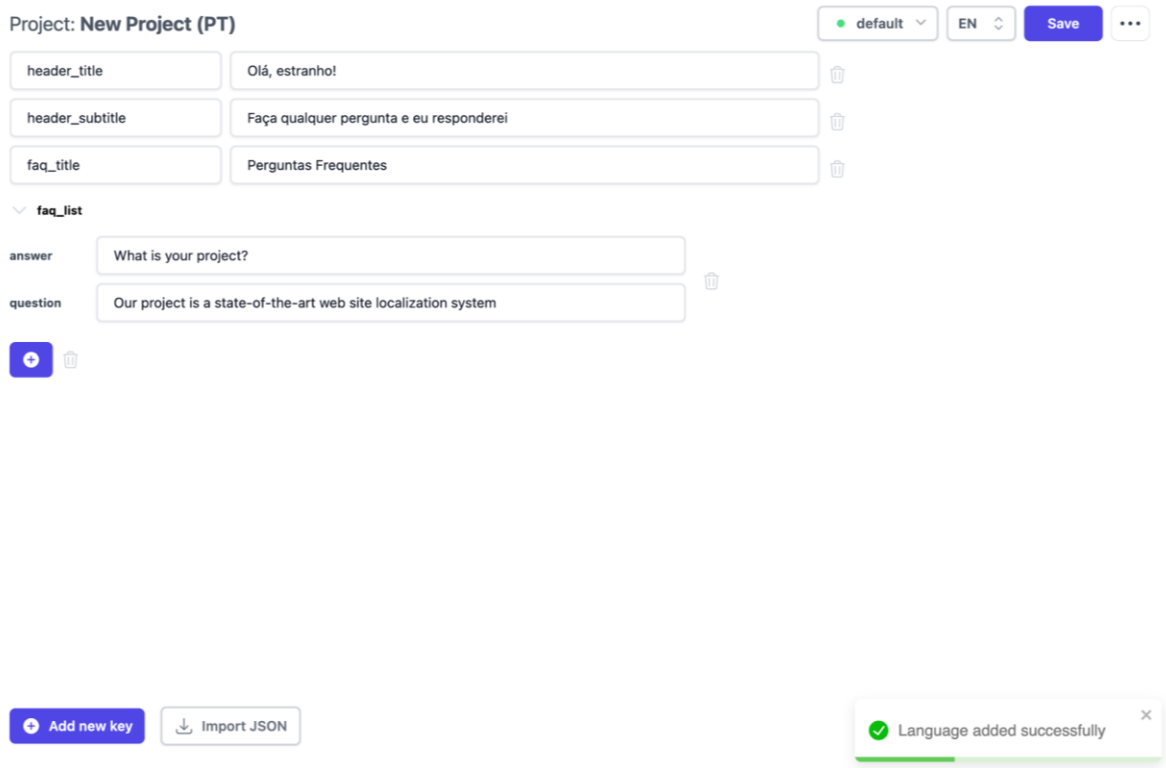

Рис 3.13. Результат перекладу тексту

Текст був перекладений на португальську мову, із збереженням оригінальних назв ключів для уникнення проблем із підключенням до frontend. Код сервісу для перекладу використовуючи генеративний ШІ можна знайти у Додатку Б.

Наступним кроком буде інтеграція перекладу у frontend. Це здійснюється завдяки заздалегідь створеному npm - пакету lokaliser. Після

його установки, все що потрібно це огорнути кореневий компонент в провайдер, наданий lokaliser. Це виглядає наступним чином:

```
return (
  <>
    <div className="App">
      <LokaliserProvider
        projectID={"047a7bf9-2b63-4896-8dfa-b570e32d50a69"}
        apiKey={"6b345684-5477-4452-b568-8f6fd0fe72547"}
        variant={variant || "default"}
        localJSON={test_lokaliser}
        countryDetect={true}
      >
        <BrowserRouter>
          <Routes>
            <Route path="/*" element={<RoutesList />} />
            <Route path="*" element={<NotFound />} />
          </Routes>
        </BrowserRouter>
      </LokaliserProvider>
    </div>
  </>
);
```

Рис 3.14. Використання lokaliser в frontend

Крім того, для використання на стороні Frontend, потрібно вказати деякі параметри. Це projectID та apiKey – ці параметри є обов'язковими. Всі інші, як от автовизначення країн чи використання локального файлу JSON для prod - середовища є опціональними.

Остання річ що залишилась – це використання правильного тексту безпосередньо в розмітці. Для цього потрібно імпортувати з lokaliser об'єкт languageKeys, та ввести назву ключа. Це виглядає наступним чином:

```
return (
    <div className='hostBg p-4 md:p-6'>
        <div className='max-w-[600px] md:max-w-[740px] m-auto'>
            <p className='text-[40px] md:text-[72px]'>{languageKeys?.hostNewTitle}</p>
            <p className='text-[40px] md:text-[72px]'>{languageKeys?.hostNewSubTitle}</p>
            <div className="flex bg-[#8649D1] rounded-2xl mt-4">
                <div className="w-full bg-purple-600 rounded-l-2xl">
                    {tabs.map((tab, index) => (
```

Рис 3.15. Використання lokaliser в розмітці

Висновки по розділу 3

У ході виконання роботи було розроблено інформаційну систему локалізації веб-додатків, яка забезпечує зручність управління локалізаційними ключами та перекладами в режимі реального часу. Використання сучасного технологічного стеку, який включає TypeScript, React, Node.js та MongoDB, дозволило створити потужний, масштабований та продуктивний інструмент для розробників і кінцевих користувачів.

Система надає функціонал для роботи з локалізаціями, включаючи авторизацію користувачів із розмежуванням доступу, додавання та редагування ключів, створення нових перекладів за допомогою генеративного штучного інтелекту, а також автоматичну синхронізацію змін із базою даних. Завдяки MongoDB як базі даних, вдалося досягти гнучкості у зберіганні даних та забезпечити їхню доступність.

Розроблений інтерфейс користувача спрямований на підвищення зручності роботи завдяки компонентній архітектурі React, що дозволяє легко розширювати функціонал системи. Водночас бекенд, побудований на Node.js, гарантує швидку обробку запитів і інтеграцію з базою даних.

Таким чином, розроблена система відповідає вимогам сучасних веб-додатків та може слугувати ефективним інструментом для управління локалізаціями, сприяючи покращенню досвіду користувачів і спрощенню роботи розробників.

ВИСНОВКИ

У межах дослідження проведено комплексний аналіз теоретичних аспектів локалізації веб-додатків у контексті сучасних тенденцій глобального ринку. Зокрема, досліджено основні вимоги до локалізації, що включають багатомовність, адаптацію до регіональних особливостей і зручність інтеграції в процес розробки.

Розглянуто взаємозв'язок понять «JSON» та «API» як ключових компонентів для організації локалізації, що дозволяють ефективно структурувати, зберігати та передавати текстові дані. Окремо описано та пояснено структуру «JSON» документів, і основну інформацію про «API» та найбільш поширені статус - коди.

Проведено порівняльний аналіз бібліотек для локалізації веб-додатків, що дозволило виділити їх переваги, недоліки та області застосування.

На основі проведеного аналізу розроблено архітектуру веб-додатку, яка забезпечує зручне та ефективне управління текстовими даними, а також підтримує динамічне оновлення контенту за допомогою API. Інтеграція локалізаційної системи з React/Next.js дозволила забезпечити автоматизоване оновлення локалізованого контенту без необхідності перезавантаження сторінок.

У результаті виконаної роботи створено локалізаційну систему, що відповідає сучасним вимогам масштабованості, зручності користування та швидкості оновлення даних. Інтеграція генеративного штучного інтелекту для автоматизованих перекладів підвищує ефективність і гнучкість системи, роблячи її придатною для широкого використання у реальних веб-додатках. Такий підхід дозволяє задовольняти потреби сучасних користувачів і забезпечує конкурентоспроможність на глобальному ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NodeJS – Wikipedia: веб сайт. URL: https://uk.wikipedia.org/wiki/Node.js (дата звернення: 15.09.2024).
2. Generative AI: веб сайт. URL: https://multilingual.com/issues/october-2023/generative-ai/ (дата звернення: 15.09.2024).
3. What is JSON: веб сайт. URL: https://www.mongodb.com/resources/languages/what-is-json (дата звернення: 29.09.2024)
4. What is an API: веб сайт. URL: https://www.postman.com/what-is-an-api/ (дата звернення 30.09.2024)
5. i18next documentation: веб сайт. URL: https://www.i18next.com/ (дата звернення 14.10.2024)
6. i18next: Comparison to others: веб сайт. URL: https://www.i18next.com/overview/comparison-to-others (дата звернення 25.10.2024)
7. airbnb/polyglot.js: Give your JavaScript the ability to speak many languages: веб сайт. URL: https://github.com/airbnb/polyglot.js (дата звернення 02.11.2024)
8. Основи MongoDB: веб сайт. URL: https://devzone.org.ua/post/osnovi-mongodb (дата звернення: 14.11.2024).
9. MongoDB – Wikipedia: веб сайт. URL: https://uk.wikipedia.org/wiki/MongoDB (дата звернення: 14.11.2024).
10. Getting started with React: веб сайт. URL:

https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started (дата звернення: 15.11.2024)

11. How to use Axios with React: веб сайт. URL: https://www.digitalocean.com/community/tutorials/react-axios-react (дата звернення 15.11.2024)
12. Що таке React JS і для чого він потрібен: веб сайт. URL: https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/ (дата звернення 17.11.2024)
13. The Best React Libraries for Internationalization: веб сайт. URL: https://phrase.com/blog/posts/react-i18n-best-libraries/ (дата звернення 18.11.2024)
14. What is MongoDB: веб сайт. URL: https://www.techtarget.com/searchdatamanagement/definition/MongoDB (дата звернення 18.11.2024)
15. MongoDB vs SQL: веб сайт. URL: https://www.knowi.com/blog/mongodb-vs-sql/ (дата звернення 18.11.2024)
16. MongoDB vs. MySQL Differences: веб сайт. URL: https://www.mongodb.com/resources/compare/mongodb-mysql (дата звернення 18.11.2024)
17. Client-server model: веб сайт. URL: https://en.wikipedia.org/wiki/Client%E2%80%93server_model (дата звернення 24.11.2024)
18. Node.js Overview: What is Node.js and Why It Matters: веб - сайт.

URL: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs (дата звернення 26.11.2024)

19. MERN Stack: An Overview: веб сайт. URL: https://www.coursera.org/articles/mern-stack (дата звернення 24.11.2024)
20. MERN Stack Explained: A Beginner's Guide [2025]: веб сайт. URL: https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mern-stack-introduction-and-examples (дата звернення 24.11.2024)

ДОДАТКИ

ДОДАТОК А

```
import React from 'react'
import { cn } from '../utills'
import { Menu, MenuButton, MenuItems, MenuItem, Transition } from
'@headlessui/react'
import { deleteProject } from '../api/projects'
import { EllipsisHorizontalIcon } from '@heroicons/react/24/outline'
import moment from 'moment'
import { Link } from 'react-router-dom'
import { useQueryClient } from '@tanstack/react-query'
import { toast } from 'react-toastify'
import { useUserStore } from '../../stores/userStore'

export default function ProjectCard({ project }) {

  const user = useUserStore(state => state.user)

  const queryClient = useQueryClient()

  const handleDeleteProject = async () => {
    try {
      const res = await deleteProject(project.id)
      toast.success(res.data.message)
      queryClient.invalidateQueries('projects')
    } catch (error) {
      toast.error(`Deleting error: ${error.response.data.message}`)
    }
  }

  return (
    <div className="overflow-hidden rounded-xl border border-gray-200">
      <div className="flex items-center gap-x-4 border-b border-gray-900/5
bg-gray-50 p-6">
```

```
          <div className="text-sm font-medium leading-6 text-gray-
900">{project.name}</div>
          {
            user?.role === 2 || user?.role === 3
              ? <Menu as="div" className="relative ml-auto">
                <MenuButton className="-m-2.5 block p-2.5 text-gray-400
hover:text-gray-500">
                  <span className="sr-only">Open options</span>
                  <EllipsisHorizontalIcon className="h-5 w-5" aria-
hidden="true" />
                </MenuButton>
                <Transition
                  enter="transition ease-out duration-100"
                  enterFrom="transform opacity-0 scale-95"
                  enterTo="transform opacity-100 scale-100"
                  leave="transition ease-in duration-75"
                  leaveFrom="transform opacity-100 scale-100"
                  leaveTo="transform opacity-0 scale-95"
                >
                  <MenuItems className="absolute right-0 z-10 mt-0.5 w-32
origin-top-right rounded-md bg-white py-2 shadow-lg ring-1 ring-gray-900/5
focus:outline-none">
                    <MenuItem>
                      {({ focus }) => (
                        <div
                          onClick={e => handleDeleteProject()}
                          className={cn(
                            focus ? 'bg-gray-50' : '',
                            'block px-3 py-1 text-sm leading-6 text-gray-900
cursor-pointer',
                          )}
                        >
                          Delete
                        </div>
                      )}
                    </MenuItem>
                  </MenuItems>
```

```
                    </Transition>
                </Menu>
                : null
            }
        </div>
        <dl className="-my-3 divide-y divide-gray-100 px-6 py-4 text-sm
leading-6">
            <div className="flex justify-between gap-x-4 py-3">
                <dt className="text-gray-500">Languages</dt>
                <dd className="flex gap-2">

                {
                    project.translations.length === 0
                        ? 'No languages'
                        : project.translations.map((translation) => (
                            <span key={translation.id} className="inline-block px-
2 py-1 text-xs font-medium bg-gray-100 text-gray-800 rounded-
full">{translation.language}</span>
                        ))
                }
                </dd>
            </div>
            <div className="flex justify-between gap-x-4 py-3">
                <dt className="text-gray-500">Created</dt>
                <dd className="text-gray-700">
                    {moment(project.datetime).format('MMM DD, YYYY - hh:mm
A')}
                </dd>
            </div>
        </dl>
        <Link to={`/projects/${project.id}`} className="block text-center text-
sm font-medium text-white bg-gray-600 py-3 border-t border-gray-900/5
hover:bg-gray-900 duration-300">View Project</Link>
    </div>
  )
}
```

ДОДАТОК Б

```
import { OpenAI } from "openai";

class gptService {
  constructor() {
    this.openai = new OpenAI({
      apiKey: process.env.OPENAI_API_KEY
    });
  }

  async translateText(text, targetLanguage) {

    const prompt = `Translate JSON (only value) to ${targetLanguage}:
"${text}". Send me only JSON object in {}`;

    const response = await this.openai.chat.completions.create({
      model: "gpt-4o",
      messages: [{ role: "user", content: prompt }],
      response_format: { "type": "json_object" },
      max_tokens: 4028,
    });
    return response.choices[0].message;
  }

  async translateObject(obj, targetLanguage) {
    const text = JSON.stringify(obj);
    try {
      const translatedText = await this.translateText(text, targetLanguage);

      try {
        let parsed = JSON.parse(translatedText.content);
        return { error: false, translated: parsed };
      } catch (error) {
        return { error: true, message: 'Invalid JSON object' };
      }
    } catch (error) {
      return { error: true, message: 'Internal server error' };
```

```
        }
    }
}

export default gptService;
```