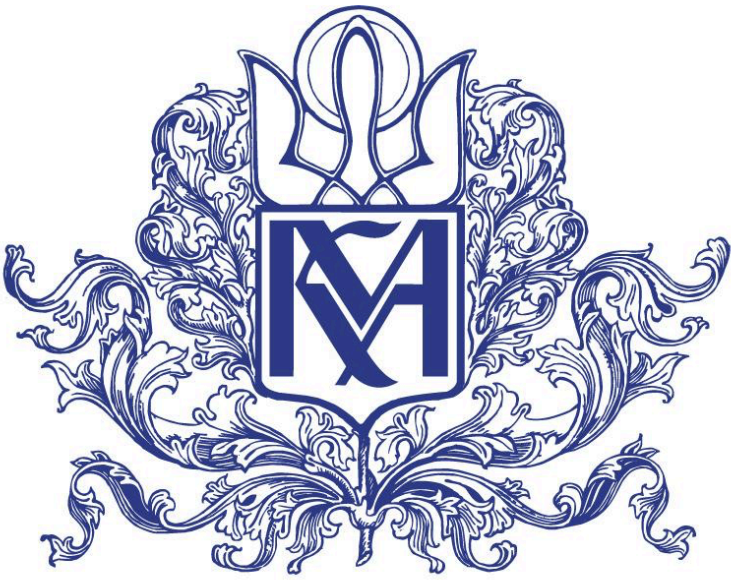
Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики



АВТОМАТИЗОВАНА ЛОКАЛІЗАЦІЯ ЗАСТОСУНКІВ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Текстова частина
магістерської роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник магістерської роботи
д.т.н., проф. Глибовець А.М.

_____
(підпис)
“____” _____ 2024 р.

Виконав студент
Верета В.В.
“____” _____ 2024 р.

Київ 2024

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

ЗАТВЕРДЖУЮ
Зав. кафедри інформатики
д.ф-м.н., проф. Малашонок Г.І
_____
(підпис)
“_____” _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу

студенту 2 р.н. магістерської програми Інженерія Програмного Забезпечення
Вереті Владиславу Віталійовичу

Дослідити Локалізацію застосунків в мікросервісній архітектурі

Зміст текстової частини до магістерської роботи:

Зміст
Анотація
Вступ
1 Теоретичні основи локалізації веб-застосунків
2 Аналіз існуючих підходів та інструментів для локалізації
3 Розробка веб-сервісу EchoLocal для оптимізації локалізації
Висновки по роботі та рекомендації для подальших досліджень
Проблематика локалізації у майбутньому
Список літератури

Дата видачі “_____” _____ 2024 р.

Керівник
А.М. Глибовець, доктор технічних наук, професор _____
(підпис)

Завдання отримав
В.В. Верета _____
(підпис)

Тема: Автоматизована локалізація застосунків в мікросервісній архітектурі

Календарний план виконання роботи:

№	Назва етапу дипломного проєкту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу	02.11.2023	
2.	Огляд літератури за тематикою роботи	16.11.2023	
3.	Виконання аналізу наявних рішень локалізації	30.12.2023	
4.	Порівняння програмних застосунків традиційного підходу до локалізації	01.01.2024 - 31.01.2024	
5.	Порівняння програмних застосунків напівавтоматичного підходу до локалізації	01.02.2024 - 29.02.2024	
6.	Порівняння програмних застосунків автоматичного підходу до локалізації	01.03.2024 - 10.03.2024	
7.	Огляд можливостей хмарних рішень Google Cloud	18.03.2024	
8.	Проєктування архітектури застосунку EchoLocal та налаштування розгортання	21.03.2024	
9.	Підготовка першого розділу роботи з оглядом необхідності підготовки інтернаціоналізації та локалізації	02.04.2024	
10.	Розробка програмного застосунку EchoLocal	18.04.2024	
11.	Підготовка другого розділу роботи з порівняльною характеристикою підходів до локалізації	27.04.2024	
12.	Написання третього розділу роботи з описом роботи застосунку EchoLocal	13.05.2024	
13.	Аналіз отриманих результатів з керівником, написання доповіді та попередній захист роботи	17.05.2024	
14.	Коригування роботи за результатами попереднього захисту	25.05.2024	
15.	Оформлення пояснювальної записки та слайдів	31.05.2024	
16.	Захист кваліфікаційної роботи	11.06.2024	

Студент Верета В.В

Керівник Глибовець А.М. “____”_____ 2024 р.

ЗМІСТ

Анотація

В рамках даної дипломної роботи був проведений аналіз різних інструментів та сервісів для здійснення локалізації *(перекладу)* веб-застосунків. В результаті було запропоновано новий сервіс, що покращує взаємодію учасників процесу локалізації *(перекладу)* веб-застосунків, а також дозволяє інтегрувати цей процес у розподілену мікросервісну архітектуру сучасних застосунків та розгортати у своїй закритій екосистемі.

Ключові слова: локалізація, інтернаціоналізація, глобалізація, переклад, i18n, l10n, мікросервісна архітектура, Kubernetes, Google Cloud Platform, GCP, CI/CD, localisation, internationalisation, globalisation.

ВСТУП

Актуальність. Локалізація або переклад веб-сервісів стає вирішальною задачею у процесі глобалізації продуктів та онлайн рішень. Наразі все більше і більше значення має своєчасність та чіткість адаптації контенту до потреб різномовних користувачів. Веб-сервіси, які підтримують більший перелік локалізацій, зазвичай здатні залучити ширшу аудиторію і забезпечити кращий, клієнтоорієнтований досвід. Однак, управління багатьма мовами може стати складною, ресурсоємною задачею без належних інструментів та підходів. Компанії, які інвестують у локалізацію своїх продуктів, можуть досягати значно кращих показників взаємодії з користувачами та збільшення доходів.

Незважаючи на це, багато організацій все ще не мають змоги швидко адаптувати свій контент під нові ринки через технічні обмеження своїх платформ, неправильно побудовану архітектуру, брак часу, брак ресурсів або просто не усвідомлення важливості цієї задачі. Часто компанії зосереджені на розширенні функціоналу своїх систем, не звертаючи належної уваги на оптимізацію процесів та користувацького досвіду. Водночас, потреба в своєчасній та точній локалізації лише зростає, а ефективність її реалізації може значно впливати на успіх продукту на міжнародному рівні.

Саме тому, необхідно підвищувати обізнаність бізнесу та продуктових рішень щодо локалізації та забезпечити розуміння не меншої важливості цього ніж конкретні особливості систем.

Мета дослідження. Розробка веб-сервісу **EchoLocal**, що надає можливість своїм користувачам інтегрувати та управляти локалізаціями своїх продуктів в одному місці.

Завдання дослідження. Аналіз існуючих рішень для управління локалізаціями веб-сервісів. Зрозуміти їх види та недоліки кожного. Розробка архітектури, яка дозволить швидко адаптувати сервіс під потреби різних

користувачів та різних архітектур проєктів. Вивчення потенціалу інтеграції з різними платформами та масштабування сервісу. Практична реалізація веб-сервісу **EchoLocal**.

Об'єкт дослідження. Веб-сервіси для локалізації контенту, методи та підходи до їх інтеграції та масштабування. Їх тонкощі імплементації, переваги та недоліки.

Предмет дослідження. Методи та інструменти для локалізації веб-сервісів, архітектурні рішення для забезпечення гнучкості та масштабованості робіт пов’язаних з локалізацією та управлінням.

Джерела дослідження. Аналіз літературних джерел, спеціалізованих веб-сайтів, форумів, відео-лекцій від експертів у галузі веб-розробки та локалізації.

Наукова новизна одержаних результатів полягає у створенні гнучкого веб-сервісу, який дозволяє користувачам самостійно управляти процесом локалізації своїх продуктів в реальному часі без залучення третіх сторін. А також розгортати сервіс у своїй інфраструктурі для своїх потреб безпеки та доступності.

Практичне значення одержаних результатів. Система **EchoLocal** дозволяє знизити витрати на локалізацію, спростити процес адаптації контенту під різні мовні ринки та підвищити ефективність комунікації між всіма учасниками процесу локалізації.

РОЗДІЛ 1: Теоретичні основи локалізації веб-застосунків

1.1 Основні поняття та визначення локалізації

Почати варто з того, що є декілька взаємопов’язаних понять, які безпосередньо стосуються приведення та перенесення онлайн-продукту на нові ринки, але не є чимось одним і тим же. Є поняття локалізації *(localisation, l10n)*, яке буде досліджене у даній роботі, а є поняття інтернаціоналізації *(internationalisation, i18n)*. Вони часто зустрічаються разом або використовуються які співзамінні, але іноді це не так. Перш ніж вдатись до деталей та їх відмінностей, давайте загалом розглянемо контекст та причини для їх появи.

Колись можна було просто відкрити продуктову лавку біля будинку і спокійно продавати товари своїм сусідам. Пізніше можливо було створити простий сайт-визитівку, який би запрошував людей у конкретне місце і так підприємці розширювали круг споживачів. Потім можливим стало втілити у життя продажі онлайн через інтернет-магазини. Але вони були локальні та працювали на локальний ринок *(мається на увазі країна або регіон)*, часто фізичними товарами. Сьогодні товар може являти собою не тільки щось фізичне, але і цифровий продукт, який можна придбати онлайн, про який можна дізнатись онлайн і який не вимагає якогось локального представництва. Тобто, тепер маючи ідею продукту для споживача, нам необов’зково створювати собі рамки і кордони - підприємець може, і насправді має, бути відкритими для всього світу. Ось тут нас і чекають ускладнення, адже що ми знаємо про весь світ? Ми знаємо і розуміємо мову нашого регіону, форматування дат, валют і чисел; знайомі нам образи і формати можуть бути незнайомі нашим покупцям на іншому кінці світу. Виходить, що ми маємо подумати про інтернаціоналізацію, вбудувати її та додати локалізацію для підтримки інших регіонів. Якраз час розібрати термінологію.

Інтернаціоналізація - узагальнене поняття, що являє собою принципи проектування та реалізацію програмного продукту або документацію таким чином, який максимально спростить подальшу локалізацію [1]. Можна сказати,

що це технічне вдосконалення продукту для того, щоб забезпечити, по-перше можливість його адаптації, а по-друге легкість локалізації для різних ринків без необхідності виконання дорогих змін. Це фундамент на якому будуються подальші налаштування відкритості світу, іншим цінностям та форматам.

Локалізація – процес адаптації програмного продукту до мови та культури клієнта [1]. Тобто саме локалізація стосується процесу зміни вже інтернаціоналізованого *(інколи, глобалізованого)* продукту для конкретного ринку. Це конкретні зміни виду, форматів та мови для споживачів конкретної країни або регіону.

Розглянемо приклади реалізації інтернаціоналізації, у чому вона полягає:

- Створення продукту з урахуванням можливості використання різних мов, які зберігаються в окремих джерелах і не вбудовані у саму систему;
- Проектування продукту з урахуванням нестандартного розміщення тексту справа-наліво, зверху-вниз. Проектування компонентів та використання вже перевірених компонентів, які вміють працювати з такими змінами;
- Використання у продукті локалізованих знаків валют, дат, чисел, тощо. Зазвичай досягається вбудованими утилітами мов програмування.

Тепер, розуміючи як і з чого складається інтернаціоналізація і перед тим як перейти саме до локалізації, пропоную подивитись порівняльну таблицю, яка наочно покаже відмінності між цими поняттями [2]:

ІНТЕРНАЦІОНАЛІЗАЦІЯ	**ЛОКАЛІЗАЦІЯ**
Процес архітектурного дизайну і розробки програми або продукту, який зможе бути адаптованим для різних мов і культур.	Процес адаптації продукту або контенту для конкретного ринку, регіону.
Включає в себе проектування продукту оцінюючи можливі мовні і	Включає в себе переклад, адаптацію та зміну продукту, щоб зустріти

культурні відмінності. Наприклад, використання Юнікоду, уникнення вбудованого тексту у застосунку, залишок достатнього місця для зміненого тексту.	конкретні лінгвістичні, культурні або регуляторні вимоги конкретної країни, регіону або ринку. Наприклад, використання локалізованого формату дат, валют та одиниць вимірювання.
Ідеально, якщо може бути виконана на етапі первинної розробки продукту, створити фундамент для локалізації.	Виконується по конкретному запиту, відповідає поточному етапу розширення та впливу.
Обидва процеси надзвичайно важливі для бізнесу або продукту, який хоче розширення та бути досяжним глобально. Ці інтегровані процеси дозволяють компаніям надавати більш персоналізований і відповідний досвід для своїх користувачів.	

Підсумовуючи, ще раз озвучу поняття самої локалізації та підкреслю, що це саме процес, процес адаптації продукту або контенту під конкретну культуру або мовну аудиторію. Локалізація забезпечує не тільки переклад тексту, але й адаптацію графічних елементів, валют, форматів дати та часу, використання правильних культурних референцій, а також адаптацію технічних вимог, таких як кодування символів. Процес локалізації необхідний для того, щоб продукт сприймався природно користувачем, підвищуючи тим самим його зручність та ефективність використання в різних регіонах та ринках. Така адаптація дозволяє компаніям ефективніше взаємодіяти з міжнародною аудиторією та розширювати свій вплив.

1.2 Проблематика та необхідність локалізації сучасних веб-рішень

Чому ж локалізація така важлива? Невже не можна обрати декілька поширених мов, форматів, одиниць вимірювання і просто використати їх? Насправді, все залежить від очікуваного результату. Адже, на початку, на етапі

становлення продукту ніхто особливо не вкладається у локалізацію, а чекають слушної нагоди, моменту розширення. Деякі бізнеси на цьому і зупиняються, порівнюючи вартість і необхідні ресурси та можливу вигоду від таких дій. В той же час, основи інтернаціоналізації мають закладатись саме в цей час. Пропоную в цьому підрозділі розглянути важливість локалізації, що це може дати сучасному онлайн-бізнесу.

Ми, як онлайн-бізнес, маємо говорити мовою клієнта. І це не просто вислів, а має пряме відношення до існуючих мов спілкування - будь-то звична нам Англійська, Іспанська або навіть Китайська. Якщо нам здається, що споживачі нашого продукту розуміють мову сайту достатньо добре і ми просто можемо пропустити етап перекладу, то скоріш за все це помилка. Факт, що існує надзвичайно міцний зв’язок між мовою контенту, мовою, якою ми пропонуємо купити щось та бажанням споживача зробити покупку.

Було проведено невелике дослідження у 8 країнах - Common Sense Advisory - з 2430 опитаних веб-споживачів [3]. Вони хотіли вивчити зв’язок як мова впливає на їх купівельну поведінку і зрозуміли, що:

- 72,1% споживачів проводять більшість або весь їхній час на сайтах їх рідною мовою;
- 72,4% споживачів сказали, що більш вірогідно купили б продукт з інформацією їх рідною мовою;
- 56,2% споживачів сказали, що можливість отримати інформацію їх рідною мовою набагато важливіше ніж ціна.

Тобто, більше половини респондентів не проти заплатити за товар більше, якщо зможуть отримувати інформацію рідною мовою.

У 2011 році вчення від European Commission [3], яке базувалось на Gallup survey серед користувачів інтернету в 23 країнах Євросоюзу відкрило декільки декілька схожих тез:

- Дев’ять з 10 інтернет користувачів сказали, коли наявний вибір, то вони завжди оберуть сайт на їх рідній мові;

- Приблизно один з 5 Європейців сказав, що ніколи не шукав нічого в інтернеті іншою ніж його рідна мова;
- 42% сказали, що ніколи не купують продукти або сервіси іншими мовами.

Знову-ж-таки, тут треба вважати мультикультурне підгрунтя Європейців. Багато з них знають декілька мов та вільно ними володіють, але в той же час надають сильну перевагу купівлі продуктів і сервісів з веб сайтів, які говорять їх рідною мовою. Звичайно, щоб досягти успіху, мало просто перекласти сайт і зробити його доступним з будь-якого куточку світу. Але це одна із тих речей, про яку часто забувають і яку буде вже не так просто зробити на пізніших етапах. Саме тому важливо думати про інтернаціоналізацію на початку шляху.

Підсумовуючи, що можна тут сказати. Дослідження доводять, що навіть, якщо людина володіє багатьма мовами і вільно їх вживає, то скоріш за все вона все одно буде надавати перевагу контенту власною, рідною мовою. В якійсь мірі це логічно, адже наш мозок завжди буде шукати найлегший шлях, а рідна мова, якою ми спілкуємось з дитинства, напевно, завжди буде “на поверхні”. Цікаво було б ще дослідити, що в даному контексті з довірою. Чи дійсно люди більше довіряють сайтам і контенту рідною мовою чи тільки сприймати легше. Не впевнений, щодо переплати, але отримати інформацію про продукт рідною мовою. Тут, мені здається неоднозначно, адже ніхто не забороняє отримати всю інформацію про продукт, відгуки та інше рідною мовою, а купити там, де дешевше, краще, зручніше. В той же час, впевнений, що у процесі підготовки локалізації на якийсь певний ринок, важливо трохи дізнатись про соціокультурне оточення, яка характеристика цього суспільства і чого важливо дотримуватись при виході на цей новий ринок. Типовий приклад із життя, багато іноземних сайтів не сильно переймаються соціокультурною ситуацією наших східних регіонів і просто йдуть найлегшим шляхом у процесі локалізації виставляючи російську мову за замовчуванням для України. У нас є власна мова і культура, але чомусь це не є очевидним для деяких надавачів послуг та продуктів, які всіх рахують як одне. Ще один приклад із-за кордону - корпорація McDonald’s. Вони не просто перекладають сайт, а пішли далі - вони змінюють

своє меню, щоб підкреслити ці культурні відмінності та запропонувати споживачам страви ближчі і зрозуміліші, інтерпретовані під національні. В Індії – котлету з картоплі та гороху, у Малайзії – десерт зі смаком дуріану, а в Україні найчастішим гостем у бургерах буде сало *(бекон)* [4].

Локалізація включає не лише переклад вмісту на мову місцевих користувачів, але й враховує культурні, юридичні та технічні аспекти, такі як формат дати та часу, національні символи, способи оплати, закони та правила, адаптацію дизайну тощо. Вона важлива для збільшення привабливості бренду для місцевої аудиторії, збільшення конверсії та розширення географії бізнесу.

1.3 Виклики при здійсненні локалізації

У попередніх підрозділах ми розглянули, що таке локалізація, навіщо вона потрібна та як до неї підготуватись, починаючи з інтернаціоналізації. Як нам стало відомо, локалізація, має декілька складових, але предметом дослідження цієї роботи буде лише одна, а саме - переклад. Переклад, як один з найважливіших, на мою думку, та найбільш помітних одразу елементів локалізації і як наслідок, який має найбільший вплив на користувача. Я впевнений, що мова сайту одразу кинеться в очі та буде оцінена користувачем ніж валюта у якій може проходити покупка на даному веб-сайті, або формат дати. Тому, я сконцентрувався на цій частині і далі наведена інформація, аналіз та висновки будуть націлені на розкриття саме цього аспекту.

Тож, локалізація важлива, потрібна і сближує нас і потенційним користувачем нашого продукту або сервісу. Але чи все завжди є і буде гладко? Які можуть виникати виклики, складнощі і проблеми у процесі впровадження локалізації? Пропоную це і розглянути у даному підрозділі.

Не завжди процес адаптації та локалізації сайтів відбувається гладко, а помилки, в свою чергу, можуть призвести до серйозних негативних наслідків. Неточний, неправильний переклад може відштовхнути потенційних клієнтів, викликати невдоволення у існуючих та інколи зашкодити репутації компанії. Серед поширених помилок, які допускаються при локалізації веб сайтів є такі:

- Пізній початок впровадження. Як уже говорилось раніше, що починати потрібно з впровадження інтернаціоналізаційної підтримки якомога раніше, щоб бути готовим до швидкої і безпроблемної адаптації контенту до нового регіону або ринку;
- Помилки у дизайні. Зазвичай про це не думають, але дизайн сайту має бути універсальним та зрозумілим, а можливо іноді навіть гнучким з можливістю його зміни. Через тонкощі сприйняття кольорів, розстановки елементів, шрифтів та інших символів у різних країнах та культурах, можлива підміна понять та втрата сенсу;
- Неточності у назві, слоган або інші заклики, які мають неоднозначний переклад, розуміння та можуть викликати різні асоціації у різних культурах. Важливо розуміти, що те, що сприймається одним чином у нас, може мати абсолютно інше значення у інших;
- Використання зображень із текстом. Зараз це рідко практикується саме через незручності у зміні і адаптації текстового контенту на веб сторінці, але раніше часто можна було зустріти, коли деякі частини сторінок із текстом замінювали зображеннями. Це дозволяло підкреслити, виділити якусь частину змісту, застосувати дизайнерський шрифт або застосувати нестандартне форматування і розташування елементів. Очевидний недолік, що ця картинка переїзжала разом із перекладом, все перекладалось, а вона - ні;
- Ну і найбільш очевидна можливість помилитись - неправильний переклад або переклад поза контекстом. Тут можуть бути як технічні причини - неправильно застосований переклад через, що перекладений текст з’явився не на тому місці, а проконтролювати це не завжди можливо, бо не всі володіють мовою, якою здійснено переклад. Такі помилки можливі і відсутність контролю також можлива, все залижить від методології та процесу локалізації, які ми розглянемо у наступному розділі. А може бути і не технічні причини, такі як людський фактор і невдало підібрані слова,

або переклад без контексту, коли при перекладі втрачається суть адже перекладач не розуміє контексту в якому використаний цей текст.

І ось, здавалось би така проста задача перекладу тексту, може викликати складнощі, якщо вчасно не задуматись про таку роботу. Наслідками поганої локалізації можуть стати:

1. Труднощі у розумінні контенту сайту та використання його функціональності, що може призвести до незадоволенності користувачів та їх відтоку;
2. Втрата потенційних клієнтів;
3. Проблеми з місцевим законодавством;
4. Негативний вплив на репутацію компанії.

Як наслідок перерахованих проблем - обмеження доступу до потенційних ринків, які можна було б охопити, якби все було зроблено вчасно і правильно.

Як висновок, я хотів би ще додати такий момент, що проблематика та перспективність локалізації на цьому не обмежується. Ще є куди рости та куди розвивати локалізацію, для розширення впливу та збільшення доступності для потенційних споживачів. Можна тестувати різні кольорові гами, різні тексти та підходи до закликів, адже всі ми різні та по-різному сприймаємо інформацію і це дуже залежить від нашого культурного підґрунтя.

РОЗДІЛ 2: Аналіз існуючих підходів та інструментів для локалізації

2.1 Сучасні підходи до локалізації веб-застосунків

Вибір підходу до локалізації або перекладу веб-застосунків може бути непростою задачею. Ми вже визначили навіщо нам робити наш сайт доступним для більшої аудиторії, але як це зробити нам ще тільки потрібно розглянути.

Звичайно, кожен розробник, менеджер або власник проєкту обирає сам як буде проходити процес локалізації, наскільки це критично та хто буде за це відповідати. Та кількість інформації з цього приводу, а також інструментів теж не мала, тож розглянемо загальні підходи і конкретно запропоную рішення на стику цих підходів у наступному розділі.

Починаючи з вбудованих, зав’язаних на коді підходів до повноцінних сервісів, які повністю перепишуть ваш контент немає єдиного правильного рішення, яке буде підходити у кожному випадку. В залежності від загального підходу до бізнесу, налаштованих бізнес-процесів та навіть орієнтації самої компанії буде залежати, які підхід до локалізації буде служити краще. Наприклад, є повноцінно комп’ютерно налаштовані організації, де все йде від розробників, де всі говорять зрозумілою лише технічним спеціалістам мовою. Тут мабуть не буде абсолютною проблемою, якщо процес налаштування і зміни локалізованого контенту буде проходити через розробника із фактичними змінами у кодову базу. Але ж є і такі компанії, які все ще потребують послуг розробників, але не є чисто комп’ютерно налаштованими, де міркують більше самою ідеєю, заробітком та маркетингом. Напевно тут було б краще обрати підхід, який вимагає або менше участі розробника або ж взагалі все можна зробити без нього. Тут ми наочно бачимо, що можуть бути різні підходи і які по-різному будуть працювати у різних компаніях, але варто пам’ятати і про недоліки. Адже не тільки варто думати про сам процес та переваги, кожен із підходів матиме певні нюанси про які варто пам’ятати, які мають вплив.

Обираючи один з підходів, які ми розглянемо вже незабаром, я перелічу орієнтовні питання, які варто поставити перед самим вибором [5]:

- **Хто створює та коректує контент на веб сайті?** Від того, звідки береться сам текст і хто його модерує може сильно залежати вибір. Чи самі розробники підготовлюють текст, пропонують його для відображення на сайті або це вже затверджений керівництвом маркетинговий матеріал, який пишеться не розробниками? У багатьох випадках розробники визначають, який текст з'являється в інтерфейсі, але маркетингові тексти можуть вимагати участі спеціалістів з інших відділів.
- **Як часто плануються зміни у вже готовий текст?** Який розмір та формат тексту найчастіше використовується на вашому веб сайті? Є велика різниця між часто повторюваними короткими фразами і фрагментами та великими, осмисленими шматками, які потребують періодичних змін. Часті зміни можуть вимагати гнучкої системи локалізації, здатної швидко адаптуватися до нових вимог.
- **Хто відповідає за переклад?** Чи планується залучати зовнішні мовні агентства для перекладу текстів або планується особисте керування процесом? Від цього залежить, наскільки гнучкою та керованою буде ваша система локалізації.
- **Наскільки ІТ орієнтована компанія та скільки розробників можуть бути задіяні?** Чи буде постійна підтримка з боку інженерів у майбутньому, або ж вона буде доступна лише на етапі програмування та налаштування? Важливо врахувати, чи зможе бути забезпечена постійна технічна підтримка для системи локалізації.
- **Чи важлива продуктивність додатку?** Оптимізація продуктивності веб-сайту може вимагати додаткових витрат на реалізацію у порівнянні із автоматизованими рішеннями.
- **Чи є інші системи, з якими потрібно інтегруватися?** Наприклад, системи управління контентом *(CMS)*, бази даних або платформи веб-хостингу. Інтеграція з існуючими системами може бути важливою частиною стратегії локалізації.

- **Чи є вимоги до безпеки та системи авторизації?** Великі компанії звертають на це увагу і для них дійсно важливий контроль того, хто матиме доступ до їх ресурсів та матеріалів. Часто використовуються системи єдиної авторизації через сервіси SSO *(Single Sign On, наприклад Okta)* і їх підтримка також важлива складова забезпечення безпеки.

Розглянувши питання, можна перейти до переліку самих підходів, яких насправді всього три:

1. Локалізація вбудована у код *(традиційна, ручна)*;
2. Локалізація винесена в окрему CMS *(напівавтоматична)*;
3. Проксі-локалізація *(автоматична)*.

Щоб спростити вибір та для більшої наочності, пропоную розглянути порівняльну таблицю по критеріям:

Критерій	**1. Ручна**	**2. Напівавтоматична**	**3. Проксі**
Складність налаштування	Висока	Висока	Низька
Необхідність участь розробника для завантаження нового перекладу	Так	Ні	Ні
Продуктивність	Висока	Висока	Різниться

У наступних підрозділах я пропоную пройтись по всім озвученим підходам, розглянути їх особливості, перелічити недоліки та переваги.

2.2 Традиційний підхід або ручна локалізація

В залежності від технологічного стеку самого веб-застосунку можуть бути різні “природні” рішення саме для нього. Можна вважати їх базовими, з яких все починається і які підтримані розробниками. Підійдуть ці варіанти чи ні у кожному окремому випадку використання необхідно дивитись на конкретних прикладах. Як уже було описано у попередньому підрозділі є багато питань, які необхідно проаналізувати, щоб зрозуміти наскільки має бути ефективним той чи

інший підхід. Але коли немає чіткої впевненості, такі базові підходи, які закладені у технологічний стек можуть слугувати тим мінімумом з якого можна почати. Я зі своєї сторони пропоную розглянути такий базовий підхід до традиційної локалізації властивий React-екосистемі з якою я працюю та маю певний досвід.

Природним вибором для React-коду є використання бібліотек i18n, таких як react-i18next, react-intl або lingui. Існують також спеціалізовані бібліотеки для конкретних фреймворків, наприклад, gatsby-plugin-i18n для Gatsby та next-intl для Next.js. На високому рівні архітектура більшості цих бібліотек схожа, тому я використаю приклади коду react-i18next для ілюстрації основних концепцій.

Як це працює?

1. **Файли перекладу.** В основі процесу локалізації знаходяться файли перекладів. Їх можна зберігати як статичні файли або імпортувати безпосередньо в код. Хоча кожна бібліотека має свої специфікації для цих файлів, на високому рівні файли перекладу просто складаються з рядків, які використовуються в додатку, ідентифікованих за допомогою унікальних ключів *(ключ-значення записи)*. Формат файлу зазвичай *json*.

   {

   “title”: “Hello, World”,

   “subtitle”: “What a great day to say Hello!”

   }

   Подібні записи будуть міститись у всіх файлах, по одному для кожної мови, яка підтримується застосунком.

2. **API бібліотека.** Далі необхідно налаштувати роботу із файлами перекладу у самому застосунку. Для саме цього використовуються бібліотеки, які дозволяють це зробити. У наступному прикладі показано як можна завантажити файли перекладу через HTTP і автоматично визначити та встановити локаль зважаючи на *Accept-Language* HTTP заголовок. Використана бібліотека i18n з пакету i18next, що є доволі типовим вибором для React застосунку.

```
import i18n from 'i18next'
import { initReactI18next } from 'react-i18next'
import Backend from 'i18next-http-backend'
import LanguageDetector from 'i18next-browser-languagedetector'

i18n
    .use(Backend)
    .use(LanguageDetector)
    .use(initReactI18next)
    .init({я
        fallbackLng: 'en',
        debug: true,
        interpolation: {
            escapeValue: false,
        },
    });
```

Цей приклад коду налаштовує бібліотеку i18next для використання в React-додатку з різними плагінами та конфігураціями. Давайте детально розглянемо кожен крок і параметр налаштування.

1. Підключення плагінів:
    a. *.use(Backend)*: Підключає плагін i18next-http-backend, який дозволяє завантажувати файли перекладу через HTTP. Це означає, що файли перекладу можуть зберігатися на сервері або CDN і завантажуватися за потреби;
    b. *.use(LanguageDetector)*: Підключає плагін i18next-browser-languagedetector, який автоматично визначає мову користувача на основі заголовка HTTP *Accept-Language*. Це допомагає автоматично встановлювати локаль для користувача відповідно до його мовних налаштувань браузера;

c. *.use(initReactI18next)*: Інтегрує i18next з React через плагін react-i18next, що дозволяє використовувати можливості i18next у React-компонентах.

2. Ініціалізація i18next:
   a. *.init({ ... })*: Метод init приймає об'єкт конфігурації, який задає параметри для налаштування i18next.
3. Параметри конфігурації:
   a. *fallbackLng: 'en'*: Визначає мову за замовчуванням, яка буде використовуватися, якщо мова користувача не підтримується або не визначена. У цьому прикладі мова за замовчуванням – англійська (en);
   b. *debug: true*: Включає режим налагодження, що дозволяє бачити додаткову інформацію в консолі браузера. Це корисно для розробки та відлагодження;
   c. *interpolation: { escapeValue: false }*: Налаштовує інтерполяцію рядків. Параметр *escapeValue: false* вказує на те, що значення не потрібно екранувати, оскільки React автоматично виконує екранування для запобігання XSS-атак.

Виконавши подібне налаштування та подбавши про файли перекладів тепер можливо використовувати ключі у самому коді. Наприклад:

```
import { useTranslation } from 'react-i18next';

const NewComponent = () => {
    const { t } = useTranslation();
    return <h1>{t('title')}</h1>;
};
```

Також, конкретно ця бібліотека для React включає окремий компонент - *Trans*, який ще більше розширює можливості підстановки тексту перекладу, ускладнює її. Наприклад, тепер не тільки текстові рядки можуть бути замінені

по ключу, а налаштовані окремі теги-компоненти, які будуть використані у тексті:

```
import { Trans } from ‘react-i18next’;

const NewComponent = () => {
    return (
        <Trans
            i18nKey="key_with_substitution"
            values={{ world: 'world'}}
            components={{ strong: <strong /> }}
        />
    );
};
```

Компонент вище має взяти ключ *key_with_substitution* та з використанням значень підстановки повернути форматований рядок.

```
{
    “title”: “<strong>Hello, {world}</strong>”,
}
```

Бібліотека також має утиліту для статичного аналізу коду та виявлення нових або невикористовуваних більше ключів. Це допомагає актуалізувати список ключів, але все ж далі має бути ручна праця по наповненню файлів перекладу актуальними значеннями - текстом відповідною мовою.

На цьому моменті можемо підвести певні підсумки та визначити переваги та недоліки цього підходу.

Перевагами традиційної локалізації є:

1. **Налаштування** - розробники мають повний контроль і відповідають за реалізацію локалізації у повному об’ємі. В придачу, це робиться лише один раз при конфігурації самого проєкту;

2. **Оптимізація** - переклад береться із файлу статичного, який лежить разом із кодовою базою або завантажується із стороннього ресурсу. Це значить, що після того як дані файлу доступні, переклад можливий моментально.

Недоліками ж можна виділити наступні пункти:

1. **Підтримка розробників** - потрібна постійна участь розробників у налаштуванні, перевірці та корекції контенту. Всі залежності та налаштування живуть у кодовій базі тож будь-яка зміна - зміна до кодової бази;
2. **Необхідність тримати файли перекладу поряд** - компроміс між місцем та додатковими залежностями у кодовій базі або час на завантаження певного мовного файлу. Тут я одразу можу сказати, що подібний компроміс властивий багатьом рішенням, які приймаються у програмуванні тож можливо це більше особливість, а не недолік.

Ми розглянули підхід та його імплементацію зі сторони розробників проєкту, як він влаштований та як його підтримують. Але в той же час, сама локалізація - переклад текстів відбувається в іншому місці. Для невеликих проєктів з малою кількістю значень це може виконуватись вручну перекладачами і потім знову збиратись у файли, які повернуться у код. Більші ж проєкти вже навряд чи обійдуться без профільних платформ TMS *(Translation Management System)* для управління локалізаціями. Це сервіси, які зазвичай відірвані від розробки і більше налаштовані на роботу і управління роботою команд перекладачів адже вони також мають свої процеси, які мають бути під контролем. Далі я пропоную розглянути одну із таких систем - Localazy [6]. Сервіс має багату функціональність та насичену документацію, але в даний момент я пропоную розглянути ту його частину, яка саме відповідає за закриття тієї прогалини у локалізації, яка у нас утворилась - переклад контенту.

Для того, щоб почати роботу потрібно зареєструватись на їх сайті та створити проєкт. Для цього необхідно надати:

- Назву проєкту;

- Тип проєкту - платформа підтримує переклад будь-яким зареєстрованим користувачем або приватними перекладачами, доступ яких регулюється;
- Мова проєкту - базова мова проєкту, яка буде основою для перекладів;
- Участь у спільних перекладах - використання загальних перекладів та дозвіл на використання.

Після створення проєкту є можливість завантажити файл перекладу, додати мови на які має бути переклад. Далі відбувається сам переклад, його перевірка та завантаження готових результатів. В той же час, сервіс пропонує широкий вибір ролей, на які можуть залучатись члени організації - перекладач, довірений перекладач, рев'ювер, менеджер, власник; можливість контролювати активність та інші більш просунуті функції.

Таким чином, сервіс Localazy спрощує та організовує процес перекладу, допомагає ефективно розподілити та проконтролювати роботу, щоб забезпечити якісний результат. Подібних сервісів багато, всі вони пропонують як додаткові, просунуті функції по типу рішень на основі штучного інтелекту та різного роду автоматизації. Обираючи подібний сервіс є із чого обирати, потрібні тільки час і наснага проаналізувати особливості кожного.

2.3 Напівавтоматична локалізація винесена в окрему CMS

Наступним пропоную розглянути підхід до локалізації, який мається на увазі під напівавтоматичною. У контексті цієї роботи він є таким, який відбувається за допомогою стороннього сервісу - CMS *(Content Management System)*. Такий підхід підтримує написання контенту кількома мовами, які має підтримувати веб-застосунок та спрощує редагування контенту для нетехнічних спеціалістів і перекладачів, які можуть керувати контентом через цей окремий сервіс.

Як це працює?

CMS-системи мають бути глибоко інтегровані у саму кодову базу застосунку. В залежності від самого сервісу буде різнитись ця перша конфігурація, але потім нетехнічні спеціалісти зможуть створювати новий

контент або редагувати існуючий безпосередньо через CMS. На платформі можна керувати локалями або мовами, які доступні у веб-застосунку, можна додати новий переклад для існуючої структури. Розробники потім отримують локалізований контент через вказану локаль. Але є певна унікальність у цього підходу.

Чим відрізняється традиційний підхід від CMS?

Ці два підходи мають багато спільного. Обидва зберігають перекладений контент як структуровані дані, які застосунок може отримати за локалю. Однак вони відрізняються підходом до програмування та необхідною архітектурою кодової бази адже мають різні API для взаємодії:

- Для реалізації традиційного підходу потрібна реалізація і робота з контентом на рівні рядків. Тобто кожну частину тексту, яку необхідно локалізувати потрібно обгорнути у функцію *(наприклад, t)*. Так налаштована система буде знати, що наведений ключ потрібно шукати у файлах перекладів та підставити відповідний локалізований текст;
- На відміну від цього підходу, системи CMS дозволяють визначати типи самого контенту, групуючи текстові дані у так звані компоненти - публікація у блозі, картка товару, тощо. Тобто, коли застосунок отримує дані з CMS - це результат у вигляді об’єкту, який представляє собою перекладений контент компоненту, що безпосередньо розміщений на сторінці. Такий підхід вимагає завчасно продуманої архітектури застосунку та не може бути просто заміненим іншим підходом.

Переваги напівавтоматичної локалізації на базі CMS:

1. Перший пункт аналогічний традиційному підходу - **налаштування** - розробники мають повний контроль над реалізацією перекладу та як відображається локалізований контент;
2. **Легке редагування** - оскільки дані зберігаються в CMS, всі зміни легко можуть бути внесені нетехнічними спеціалістами - маркетологами,

контент-мейкерами, перекладачами. Таким чином можуть оновлюватись дані або навіть створюватись нові елементи або локалі для застосунку.

Недолік же на мою думку один і значний - складність стартового налаштування та відповідний дизайн архітектури, який заточений під роботу з конкретною CMS та запрограмованими компонентами.

Щоб продемонструвати принцип роботи подібних систем, давайте розглянемо сервіс Contentful [7]. Цей сервіс являє собою повноцінну CMS з безліччю функцій для управління даними, ролями, перекладами, тощо.

Щоб почати працювати із системою, традиційно, потрібно зареєструватись. Далі потрібно створити так званий space *(робочий простір, скорочено від workspace)*, вказавши назву та шаблон. Наступним елементом, який задіяний на платформі є content model - це система, яка вибудується, коли ми додамо записи. Кожен запис у свою чергу, компонент, який будується на описаній у підрозділі логіці - групує більш атомарні рядки тексту або інших даних, створюючи поєднання залежних по змісту типів контенту у вигляді об’єкту. На платформі Contentful ці компоненти мають назву content types - типи контенту, які представляють собою одиницю контенту, об’єднавший малі складові частини. Щоб створити новий тип потрібно вказати назву та додати ідентифікатор. Опціонально можна додати опис. Наступним кроком буде наповнити цей тип полями з типом даних цих полів, які потрібно буде наповнювати при створенні нового компоненту на його основі. Наприклад для типу пост у блозі можна створити поля:

- Заголовок як текстове поле;
- Пост як текстове поле;
- Дата публікація як поле для дат.

Маючи подібний тип, його можна використати для створення контенту, наповнивши його саме даними. Таким чином наповнюється content model

ресурсу і потім буде доступна для завантаження такими об’єктами *(пост у блозі)* або переліком таких об’єктів *(всі пости у блозі)* у заданій локалі.

2.4 Автоматична локалізація

Цей підхід найцікавіший і майже нічого не вимагає зі сторони розробників застосунку! Але в той же час він і більш ризиковий і не дає повної гарантії на повноцінну локалізацію адже для ідеального виконання цього підходу мають бути виконані певні умови. Для досягнення так званої автоматичної локалізації багато сервісів зробили крок вперед та автоматизували збирання тексту із живого сайту, який повністю у продакшені і навіть динамічну заміну текстів на сторінці на локалізовані.

Як це працює?

Ці автоматизовані сервіси зазвичай мають свої бібліотеки, які мають бути включені напряму у кодову базу застосунку. Потім, при завантаженні ресурсу, включений скрипт знайде всі видимі тексти на сторінках та завантажить їх у дашборд. Сервіс покаже список та зможе запропонувати машинний переклад або дозволить додати його вручну. Як тільки контент буде локалізовано, майбутні завантаження веб-застосунку за допомогою встроенного скрипту буде знаходити відповідний до локалі користувача переклад та на льоту замінювати DOM елементи щоб відобразити локалізацію.

На перший погляд цей підхід здається такою собі срібною кулею, яка вирішує всі питання перекладу для веб-сайтів, майже все автоматизовано і з мінімумом налаштувань. Потрібно тільки перекладати автоматично знайдений текст та й усе, але на практиці все може бути не так райдужно через обмежену кастомізацію, неможливість зберігати та перевикористовувати частки контенту, відсутність можливості передати контекст, тощо.

Підсумовуючи, можна виділити такі **переваги** підходу:

1. **Легке налаштування** - підключити бібліотеку або скрипт напряму у застосунок, зазвичай на рівні базового html;

2. **Легке використання** - текст автоматично знайдений, вивантажений на платформу та показаний користувачу після перекладу. Оскільки все робиться через окремий сервіс, то в основному застосунку ніякі зміни не потрібні і можна слідувати своєму поточному процесу без змін.

Недоліки очевидні та випливають із переваг:

1. **Оптимізація застосунку** - оскільки текст замінюється напряму у DOM після завантаження перекладу це відбувається з певною затримкою та може бути явно бути поміченим користувачем;
2. **Обмежена кастомізація** - оскільки сервіс автоматично зчитує написані текстові рядки у застосунку, він чітко має “розуміти” структуру кодової бази, щоб їх знайти. Також, така автоматизація значно зменшує можливості індивідуальних проєктних налаштувань, адже має бути універсальною для всього застосунку.

Як ілюстрацію підхід автоматичного перекладу розглянемо сервіс Gtranslate [8]. Це сервіс просто знахідка для тих випадків, коли потрібна не обов’язково дуже точна і якісна локалізація, але її наявність може відіграти важливу роль. Робота із сервісом майже нічого не вимагає від сторони розробника, ніякі зміни до кодової бази вносити не потрібно. Є певні налаштування DNS *(Domain Name System)*, які потрібно зробити щоб пропустити сайт через сервіс та виконати переклад. Основним є налаштування CNAME *(Canonical Name)* яке буде відповідати за перенаправлення вашого сайту на сервіс локалізації, а також перенаправлення локалізованих версій.

Сервіс пропонує безкоштовну версію із машинним перекладом, а також декілька варіантів із розміщенням локалізованих версій сайту:

- Використовуючи суб-домени - https://uk.example.com
- Використовуючи суб-директорії - https://example.com/uk

Дуже зручно, що переклад який був створений за допомогою query параметрів *language_edit=1* доступний до редагування прямо на сторінці. Також

сервіс надає мінімальні можливості для встановлення селектору мов та має плагіни для налаштувань для певних популярних систем, наприклад, Wordpress.

Підсумовуючи інформацію щодо різних підходів по локалізації веб-застосунків можна сказати, що немає єдиного рішення для всіх випадків. Часто рішення залежить від того наскільки тісно з цим працювати може і зможе у майбутньому сам розробник, чи можлива його підтримка. Автоматизована локалізація може бути гарним варіантом, коли потрібна найменша підтримка зі сторони розробників, але в той же час має свої недоліки. Який би підхід не був би обраний, зазначу, що перехід між ними теж не є простою задачею. Для кожного з них кодова база має мати певну структуру та по-різному викликати локалізовані елементи, а під автоматизовану - не всі технології підійдуть.

Також важливо виділити окремо ситуацію, коли можлива інтеграція у вже існуючий проєкт без локалізації взагалі. В такому разі важливо зорієнтуватись якого типу застосунок, його архітектура та кількість тексту для локалізації. На противагу потрібно розуміти, коли і наскільки змінна локалізація потрібна. Вибір підходу до локалізації може бути складним рішенням, але якби воно не далось бізнесу, це гарне рішення, яке в перспективі здатно значно впливати на розвиток.

РОЗДІЛ 3: Розробка веб-сервісу EchoLocal для оптимізації локалізації

3.1 Концепція платформи EchoLocal

Враховуючи, наведені у попередньому розділі можливі підходи до локалізації *(тут і далі, здебільшого мається на увазі переклад)* та особистий досвід роботи, у мене виникла ідея розробити застосунок, який покращив би, оптимізував цей процес у загальному.

В чому головна складність перекладу? На мою думку, складність тут у передачі інформації у зв’язку замовник - дизайнер - розробник - перекладач. Те, що здається на перший погляд логічним і очевидним, може значно змінити своє значення у процесі цієї передачі адже учасники цього ряду можуть абсолютно не спілкуватись між собою, але лише у порядку цього зв’язку. Тобто те, що замовник придумав і полюбив, переказав у найменших деталях дизайнеру до перекладача прийде у сухому тексті і часто без контексту взагалі. Також, складності до цього всього додає те, що цей процес асинхронний і потребує певного часу на виконання своєї роботи кожним учасником. Замовник сьогодні замовляє і лише через певний час - дизайну, розробки та перекладу - побачить результат. Потрібна певна база знань, де буде збережений контекст, щоб максимально зручно і ефективно передати значення кожного тексту перекладачу.

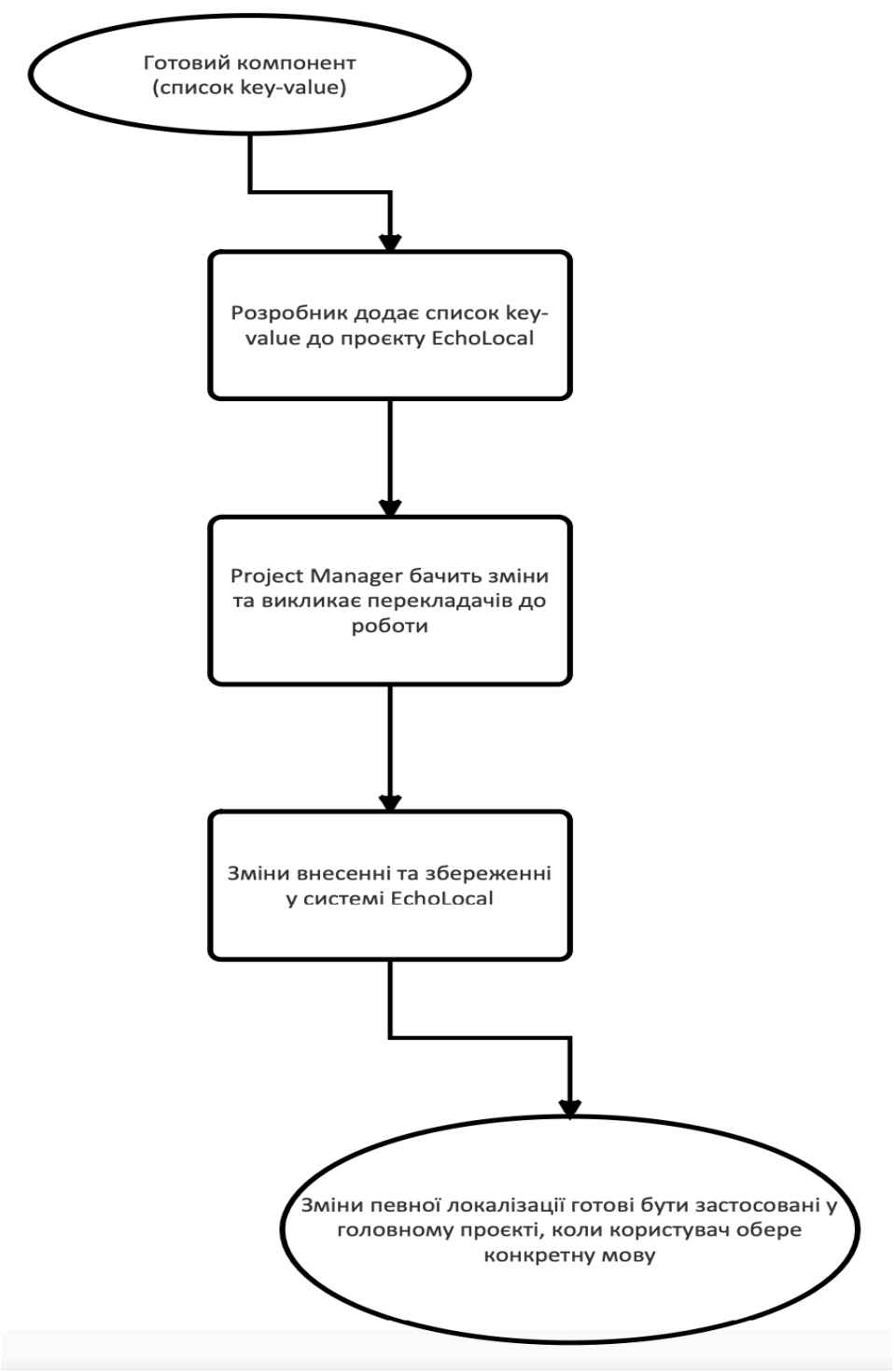
На ринку існують з однієї сторони інтегровані рішення технологічної сторони, бібліотеки, які реалізують сам функціонал перекладу у веб-застосунку, а з іншої - гарні веб-рішення для перекладачів, які спрощують роботу саме для них. Моя ж ідея полягає в тому, щоб почати реалізовувати щось середнє між цим, даючи можливість ефективно співпрацювати розробникам та людям, які відповідають за контент не вимагаючи додаткових, непотрібних дій.

Як ми уже знаємо, один з підходів до перекладу контенту - це групування ключів та їх значення у файл. Цей файл лежить на сервері разом із проєктом і завантажується разом із ним у браузер користувача. Це найшвидший та найлегший спосіб для проєкту отримати нові значення, але процес перекладу ключів, знання контексту цих ключів залишається абсолютно не зручним та не

оптимізованим. Моя ідея проєкту вирішує цю проблему. Тепер цей файл ключів має зберігатись не у проєкті всередині, а на окремому ресурсі, який бере на себе всю роботу по організації перекладу та наповнення веб-сторінки контентом. Самому проєкту залишається лише підвантажити перелік необхідних ключів з перекладами та й усе, це можуть бути всі або лише частина. Також, при такому підході може бути доступна певна метаінформація про переклад такі як дата створення, дата останнього оновлення і тд.

Продовжуючи тему зручностей, я абсолютно не зменшую зручність для самих перекладачів, адже саме для них на ринку дуже багато платформ, так званих менеджерів перекладів. Створюючи нову платформу для менеджменту перекладів і допускаючи на неї самих розробників тих проєктів, які підключаються за перекладом надає певні переваги та поліпшення. Наприклад, тепер ключі можуть створюватись людьми, які фактично їх додають у проєкт, може бути завантажена картинка для контексту та інша додаткова інформація, яка допоможе перекладачу краще зорієнтуватись.

Тож, як це має працювати *(дивись блок-схему нижче)*. Уявімо, що є дизайн і розробник вже спроєктував та вніс необхідні зміни у код-базу. Маємо робочий функціонал на веб-сторінці з текстом англійською мовою. Текст не є вбудованим, а підв’язаний до ключів, які в свою чергу перетворюються у реальний текст необхідною мовою. Якраз ці ключі і текст необхідною мовою будуть завантажуватись у проєкт. У самому проєкті є налаштований функціонал для роботи із сервісом перекладів де витягуються ключі, а також 1 файл перекладів основною мовою *(так званий fallback переклади англійською мовою згідно цього прикладу)*. Що потрібно зробити далі? Розробник самостійно вносить цей ключ та переклад у запрограмовану систему **EchoLocal** і прикріплює необхідні допоміжні елементи, щоб передати контекст. На цьому робота замовника закінчилась. Далі штат перекладачів вносить зміни, додаючи переклади на інші мови, а застосунок вже сам підтягне ці зміни в залежності від обраної мови користувачем.



У чому виграш платформи **EchoLocal**? На мою думку, у рішенні проблеми ефективності на прибиранні зайвої, непрофільної роботи з розробника. Тепер кожен займається своєю роботою та не залежить від часу і роботи іншого. Тепер ти зробив свою роботу *(запрограмував)*, підготував роботу для іншого *(завантажив ключі)* і пішов, далі все автоматично запрацює

коли буде готово *(після оновлення контенту перекладачами)*. Також є неочевидна, але перевага у контролі, адже тепер не потрібно шукати текст на веб-сайті, або перевіряючи по якомусь критерію мати доступ до усіх частин сайту. Можна просто переглянути список усіх ключів та їх перекладів на зручному веб-ресурсі та зробити необхідні зміни без будь-якої зовнішньої допомоги.

Підсумовуючи головну концепцію застосунку **EchoLocal**, дозвольте перелічити основні можливості:

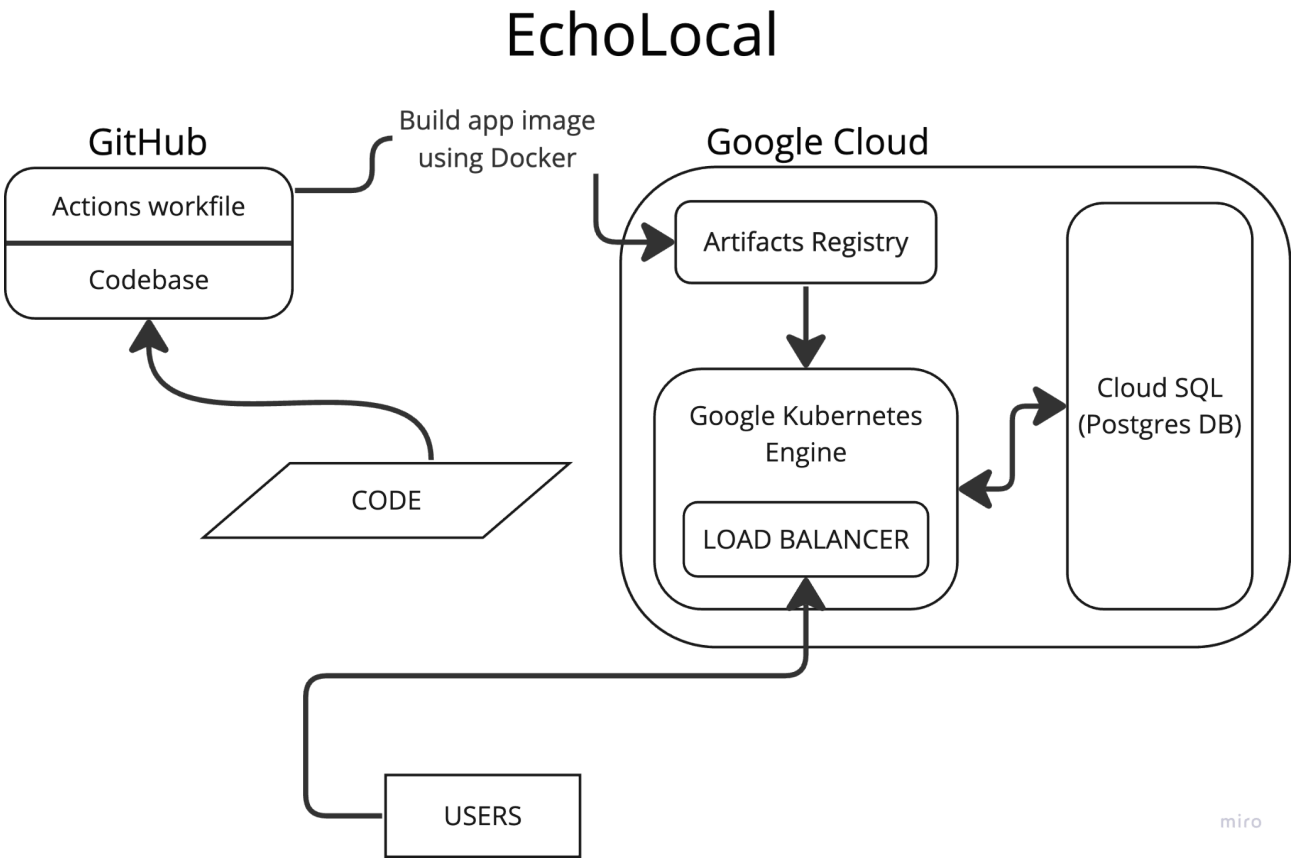
- Зберігання даних проєкту для якого робиться переклад та список усіх доступних перекладів;
- Зберігання листів перекладів на різні мови для зручної роботи перекладачів та менеджерів проєкту;
- Можливість оновлення контенту у бекграунді і автоматичного оновлення на сайті;
- Незалежність всіх учасників процесу у часі;
- Ефективність роботи всіх учасників, кожен займається своєю справою.

3.2 Основні компоненти архітектури

Починаючи мову про основні компоненти та функціональні можливості, я спочатку пропоную розглянути загальну архітектуру проєкту, схему бази даних та підхід до розгортання проєкту. У другій частині даного підрозділу на основі з скріншотів самого застосунку буде висвітлено з яких компонентів складається візуальна частина платформи та її функціональні можливості.

Загальна архітектура проєкту **EchoLocal** складається із окремих сервісів для клієнта та сервера, які співпрацюють один з одним за допомогою обміну HTTP запитами. Клієнт та сервер представляють собою два окремих застосунків розміщених у одному репозиторії *(монорепозиторій)* та слугують незалежними частинами, які в той же час співпрацюють. Оскільки однією із цілей застосування є підтримка роботи із мікросервісами, то такий розподіл є необхідним адже API має бути доступним для зовнішніх джерел - запити від

зовнішніх проєктів мають йти напряму до серверної частини застосунку і завантажувати переклад або його частину. Таким чином користувачі не прив’язані до фронтенд частини і мають можливість користуватись API напряму.



Проєкт **EchoLocal** запущений “у хмарі” на потужностях Google Cloud. Що таке “хмара” і її особливості будуть розглянуті далі. Складовими частинами проєкту є:

1. **Google Kubernetes Engine.** Частина хмарного рішення, сервіс по роботі із Kubernetes від Google Cloud. Головний тримач застосунку, автоматизований Kubernetes кластер, який має декілька реплікацій та доступ ззовні. Може гнучко налаштовуватись під потреби застосунку в залежності від навантаження.
2. **Artifacts Registry.** Частина хмарного рішення, сервіс для зберігання та управління образами застосунків для подальшого використання у Kubernetes.

3. **Cloud SQL.** Частина хмарного рішення, сервіс для баз даних від Google Cloud. Для цього застосунку була обрана база даних - Postgres. Детальна схема бази буде показана далі.
4. **GitHub.** Рішення для збереження кодової бази та подальшого розгортання проєкту. Репозиторій містить весь код проєкту та дозволяє зручно його переглядати та оновлювати. Для опersорсних проєктів *(ті, які мають відкритий код, який доступний будь-кому для перегляду або зміни)* дозволяє зручно перевірити надійність зовнішнім спеціалістам або запропонувати зміни, покращення.
5. **Допоміжні інструменти - Docker, GitHub Actions.** Інструменти, які використані для зручнішого розгортання застосунку на платформі Google Cloud. Docker - інструмент контейнеризації, який я розгляну далі дозволяє “упакувати” застосунок у контейнер із заданими налаштуваннями оточення так, що будь-який інший розробник зможе відтворити це оточення, якщо у нього встановлений Docker. Також був налаштований CI/CD процес, який автоматично оновлює застосунок до останньої версії у Google Cloud за допомогою GitHub Actions.

Хмарне рішення.

Загалом, рішення розмістити проєкт у хмарі було обрано через гнучкість і легко налаштовувану необхідну потужність рішення. Відносно просто можна налаштувати оновлення застосунку за допомогою Continuous Integration та Continuous Delivery процесів. А також, не на останньому місці, доступність застосунку, зручно показати демо-версію та налаштувати необхідні доступи. Google надає певний кредит для навчальних цілей, тож дуже вчасно і ефективно можна використати його для тестування власного рішення.

Пропоную розглянути загальну таблицю порівняння хмарного рішення та фізичного сервера як єдину опцію, яку ми мали раніше. Розумію, що для демонстрації можна було розгортути і локальне середовище, але я хотів максимально наблизитись до реального прототипу.

Критерій	**Хмарне рішення**	**Фізичний сервер**
Масштабованість	Можливість автомасштабування, просте додавання нових сервісів	Залежність від фізичних серверів та замовленої потужності
Управління інфраструктурою	Автоматичне управління, допомога з оновленнями та підтримка	Самостійне управління і налаштування
Вартість	Лише за використанні ресурси	Початкові капіталовкладення для закупівлі обладнання
Безпека	Відповідальність провайдера послуг	Власна відповідальність
Доступність та резервне копіювання	Автоматизована система контролю	Необхідність власної організації

Як і у всьому є свої переваги і недоліки у кожної із опцій, щоб зробити правильний вибір для конкретної ситуації потрібно проаналізувати і застосунок, і його архітектуру, і спосіб розгортання, і вимоги до продуктивності, тощо. Зі своєї сторони та для свого проєкту я скористався доступними мені можливостями і особливості хмарного рішення мене повністю задовільнили:

1. Зі сторони бюджету - тестовий період та бюджет, якого мені вистачить для реалізації проєкту з використанням зазначених сервісів від Google Cloud;
2. Зі сторони можливостей масштабування - не передбачено, оскільки це тільки MVP, навчальний проєкт;
3. Зі сторони управління - зручність з цієї точки зору я ставив на перше місце. Застосунок **EchoLocal** налаштований на автоматичне розгортання і оновлення згідно оновлень у репозиторії. Є доступна ip адреса за якою можна контролювати процес та провести демо;
4. Сторона безпеки та резервного копіювання сильно не досліджувалась, були прийняті всі налаштування за замовчуванням, оскільки це навчальний проєкт.

Kubernetes.

Kubernetes - платформа з відкритим кодом для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Розроблена та активно підтримується компанією Google. Цей інструмент був обраний, щоб забезпечити гнучкість у розгортанні та можливість масштабування при необхідності. Система автоматично зможе підлаштовуватись під навантаження, а також забезпечує необхідні оновлення і резервні копії станів застосунку. Загалом Kubernetes має такі основні функції, які в свою чергу стали певним стандартом у веб-розробці і дуже часто використовуються при налаштуванні деплою додатків:

- Управління контейнерами: Kubernetes автоматично розподіляє контейнери між вузлами *(nodes)* кластеру, забезпечуючи ефективне використання ресурсів;
- Масштабування: платформа дозволяє автоматично і вручну масштабувати додатки, додаючи або видаляючи контейнери залежно від навантаження;
- Управління життєвим циклом додатків: Kubernetes автоматизує розгортання та оновлення додатків, забезпечуючи безперервність бізнес-процесів;
- Відновлення після помилок: у разі помилки, контейнери будуть автоматично перезавантажені та розподілені між вузлами кластера;
- Баланс навантаження *(Load Balancing)*: Kubernetes забезпечує автоматичний розподіл навантаження між контейнерами;
- Управління конфігурацією та змінними: Kubernetes дозволяє зберігати та керувати конфігураціями додатків і змінними *(паролі, ключі)*, забезпечуючи безпеку і зручне управління.

Kubernetes є потужним інструментом для управління контейнеризованими додатками і як вже було сказано раніше, є сьогоднішнім стандартом розгортання. Для повноцінної роботи із Kubernetes потрібен наступний інструмент, який підготувати образ застосунку для роботи. Оновлення цього образу у сховищі дасть команду кластеру на перезбір.

Docker.

Docker був обраний як базова складова для контейнеризації застосунку і подальшого розгортання. Наразі це вже стандарт, навіть без використання Kubernetes, які для своєї роботи використовують контейнеризовані застосунки або образи, а навіть просто для зручності розробки та розгортання - оточення завжди буде однаковим. Саме таким, який його запрограмували. Контейнери, створенні за допомогою Docker, дозволяють ізолювати додатки з усіма їх залежностями, забезпечуючи стабільність роботи в будь-якому середовищі, від розробки до випуску. Основні функції Docker:

1. Контейнеризація. Docker дозволяє упаковувати додатки та їх залежності в так звані контейнери, забезпечуючи їх ізольоване та стабільне оточення;
2. Легкість і швидкість. Контейнери як програмований засіб легші за віртуальні машини і запускаються швидше, оскільки вони не налаштовують нову операційну систему поверх хосту, а використовують її напряму;
3. Портативність. Контейнери можуть бути запущені на будь якій операційній системі та комп’ютері, головне щоб був встановлений Docker;
4. Швидкість розгортання. Docker дозволяє швидко розгортати і масштабувати додатки, заощаджуючи час на конфігурацію та налаштування.

Docker значно спрощує процес розробки, тестування та розгортання застосунків, забезпечуючи стабільність та ефективність на всіх етапах життєвого циклу додатка.

PostgreSQL.

Postgres як реляційна база даних була обрана через свою впевнену роботу, підтримку шардінгу та реплікацій у разі необхідності. Також, у майбутньому можливе використання особливостей цієї бази даних, зокрема підтримку enum. Ця функціональність може бути використання у розподілі ролей або у підтримці

певного переліку мов, доступних до перекладу. Реляційна ж база даних у свою чергу була обрана через нормалізовану структуру самих даних, які будуть зберігатись у ній. Всі записи, а також зв’язки між ними досить очевидні та прямолінійні, як наслідок легко лягають у структуру реляційної бази даних. Варіант NoSQL бази даних також розглядався і мав би певну перевагу при зчитуванні, адже можна було простіше створити запит на отримання даних для проєкту у заданій мові, але мені здалось процес запису і налаштування зв’язків потребував би більше дій і контролю.

Пропоную розглянути ключові відмінності реляційної та NoSQL баз даних на прикладі PostgreSQL та MongoDB:

PostgreSQL	**MongoDB**
Реляційна	Документо-орієнтована
Містить таблиці, рядки та стовпці	Містить колекції та документи
Схема даних статична та визначена наперед	Схема даних гнучка та може змінюватись
Використовується для зберігання цілісності даних та зв’язків	Використовується для аналізу великих даних з динамічними схемами
Нормалізована структура - реляційні зв’язки між таблицями та записами	Денормалізована структура - документи із вбудованими документами
Висока продуктивність для складних запитів з JOIN	Висока продуктивність для швидких вставок/запитів документів
Складність масштабування горизонтально, статична та незмінна схема вимагає ретельного планування	Відсутність строгої схеми може призвести до непослідовності даних, складні транзакції можуть бути менш ефективні

Виходячи із таких вхідних даних я обрав використовувати саме реляційну базу даних адже вона закриває всі потреби застосунку **EchoLocal** та має забезпечити достатню стабільність та швидкодію.

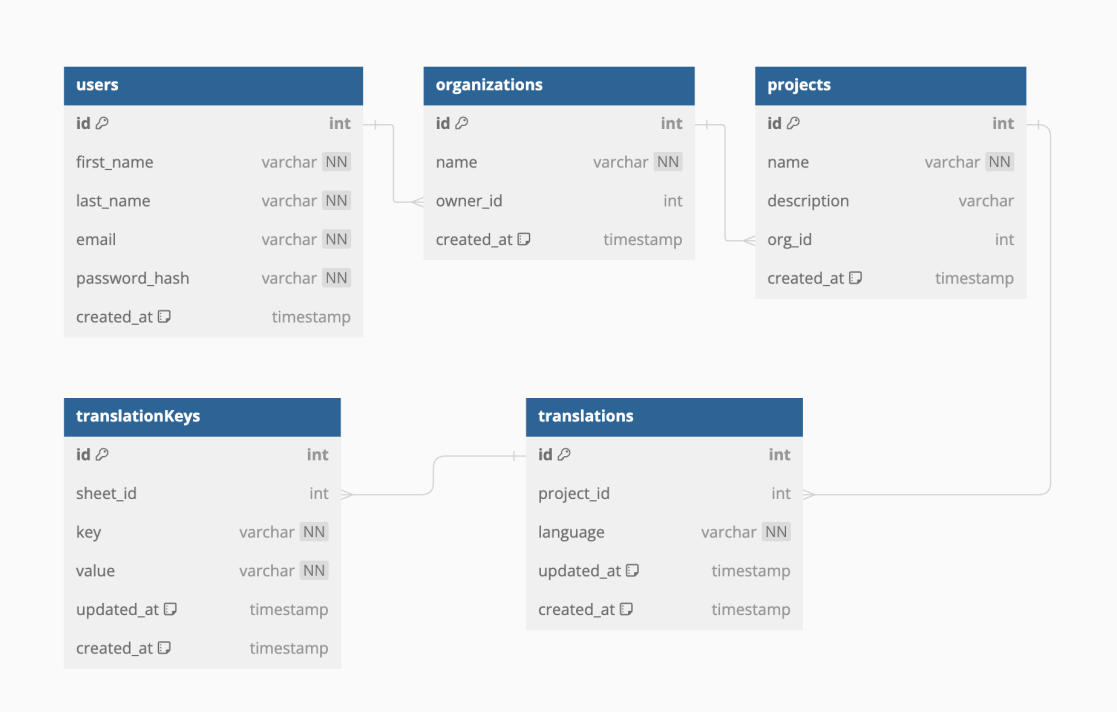
Таблиці бази даних

На даному етапі імплементації застосунку база даних має наступну таблиці як структурні елементи:

- Таблиця **users** містить дані про зареєстрованих користувачів системи - *first_name*, *last_name*, *email*, *password_hash*, *created_at*.
- Таблиця **organizations** містить дані про створені користувачами організації - *name*, *owner_id*, *created_at*.
- Таблиця **projects** містить дані про створені користувачами проєкти, які належать конкретній організації - *name*, *description*, *org_id*, *created_at*.
- Таблиця **translations** містить дані про листи перекладів створені та доступні конкретному проєкту - *project_id*, *language*, *updated_at*, *created_at*.
- Таблиця **translationKeys** містить дані про перелік ключів та їх перекладів для конкретного листа перекладу - *sheet_id*, *key*, *value*, *updated_at*, *created_at*.

Наразі це всі таблиці, які використовуються. В той же час у процесі розвитку проєкту передбачаються певні зміни у структурі та зв’язках між таблицями.

Схема бази даних виглядає так:



GitHub, GitHub Actions.

GitHub сервіс був обраний як найпопулярніший для зберігання вихідного коду. Тут просто і відкрито можна зберігати код, доповнювати його, а також налаштувати процесу деплою у GitHub Actions. Адреса репозиторію проєкту: https://github.com/vlvereta/EchoLocal

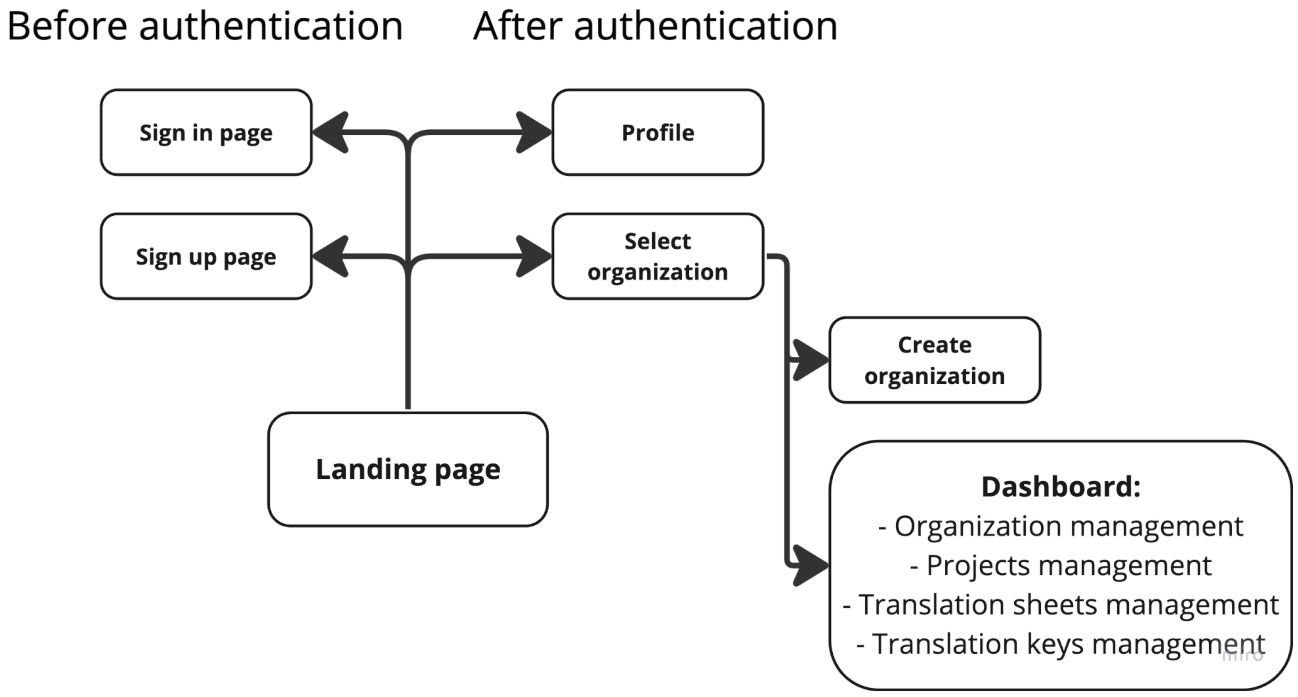
GitHub Actions налаштовані наступним чином *(https://github.com/vlvereta/EchoLocal/blob/main/.github/workflows/main.yml)*:

1. Після налаштування часу застосування - на оновленні головної гілки репозиторію *(main)* та змінних оточення, запускається перший процес - ***test***. Копіюється код в оточення, встановлюються залежності та запускається команда *yarn test* для прогону тестів.
2. Далі йде головний процес - ***build-push-deploy-artifact***. Він в свою чергу спочатку створює файл змінних оточення *(.env)*, далі авторизується у Google Cloud, налаштовує Cloud SDK та gcloud CLI. Далі створюється Docker образ застосунку та зберігається у Google Artifact Registry. Після цього у справу вступає Kubernetes - налаштовується доступ та запускаються команди на оновлення застосунку згідно з новою версією образу аплікації.

3.3 Функціональні можливості EchoLocal

Щоб розглянути функціональні можливості саме веб-застосунку **EchoLocal**, давайте пройдемось по основним сторінкам інтерфейсу. Тут наочно я зможу описати можливості та функціонал доступний користувачу системи. На момент написання роботи, доступна лише одна роль користувача, який реєструється і має доступ до всього функціоналу. У подальшому планується введення більш гранулярного доступу.

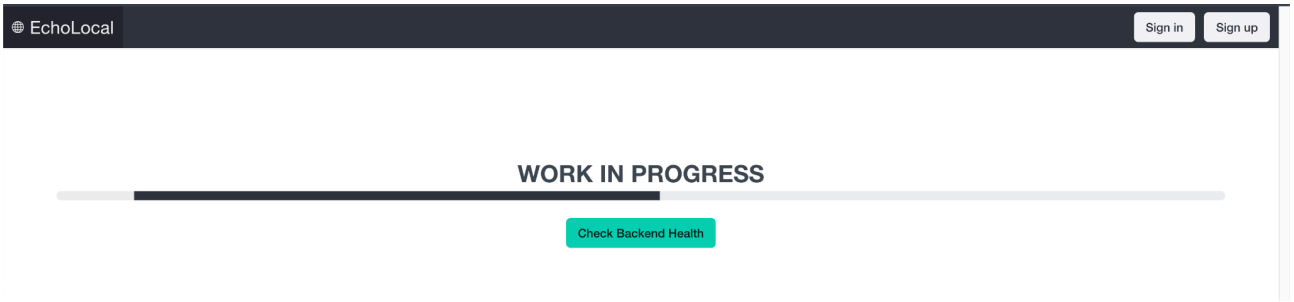
Портал **EchoLocal** умовно розділений на 2 частини в залежності від статусу користувача - авторизований він чи ні. Головна сторінка *(Landing page)* доступна для всіх та має містити загальну інформацію про сервіс. Для неавторизованого користувача доступні сторінки входу та реєстрації.



Якщо користувач авторизований, то він має повний доступ до системи - сторінки профілю *(Profile)*, вибору та створення організації *(Select organization, Create organization)* та головного дашборду *(Dashboard)* - сторінка, які містить усю корисну взаємодію із сервісом. Далі пропоную розглянути кожну сторінку та її функціональність окремо для більшого розуміння загальної функціональності та можливостей системи.

Доступні сторінки та їх функціональність у системі

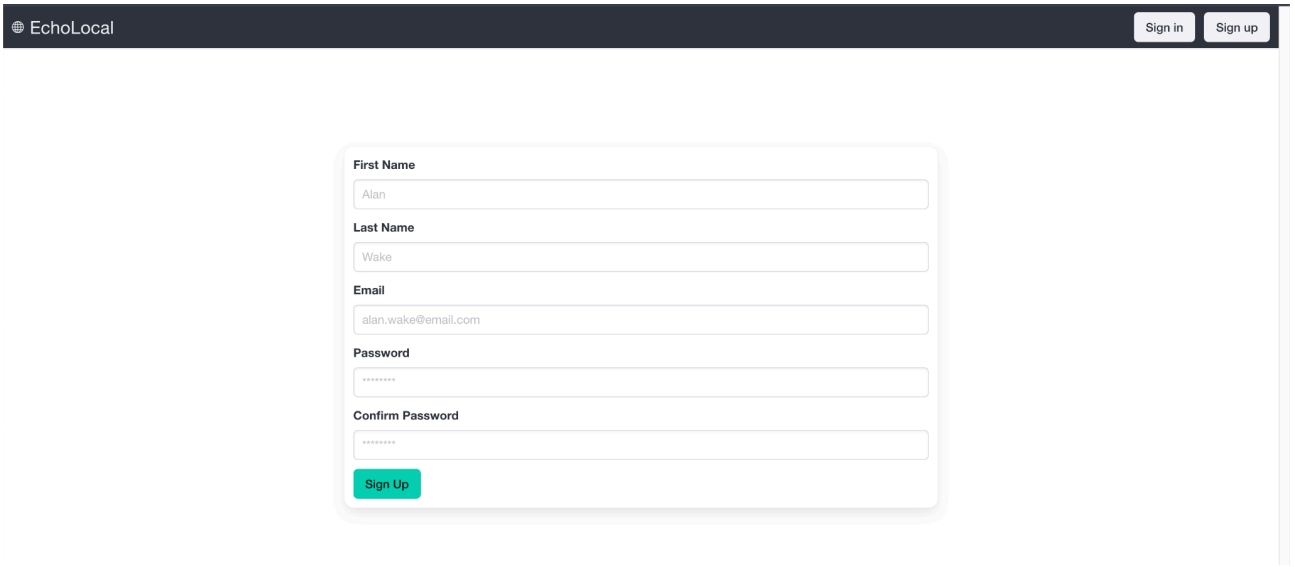
- Головна сторінка *(Landing page)*: */*



На головній сторінці відображається простий інтерфейс з логотипом **EchoLocal** та меню навігації, яке дозволяє неавторизованому користувачу зареєструватись або увійти до системи; а авторизованому - переходити між

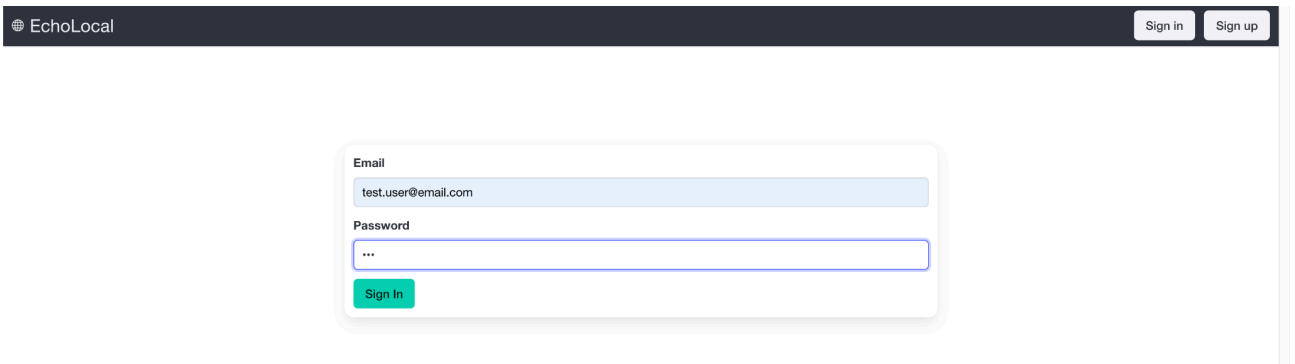
сторінками системи. Також варто варто зазначити, що візуальна частина сторінки не готова і показує лише те, що сторінка у процесі розробки та кнопку перевірку статусу сервера.

- Сторінка реєстрації *(Sign up page)*: */signup*



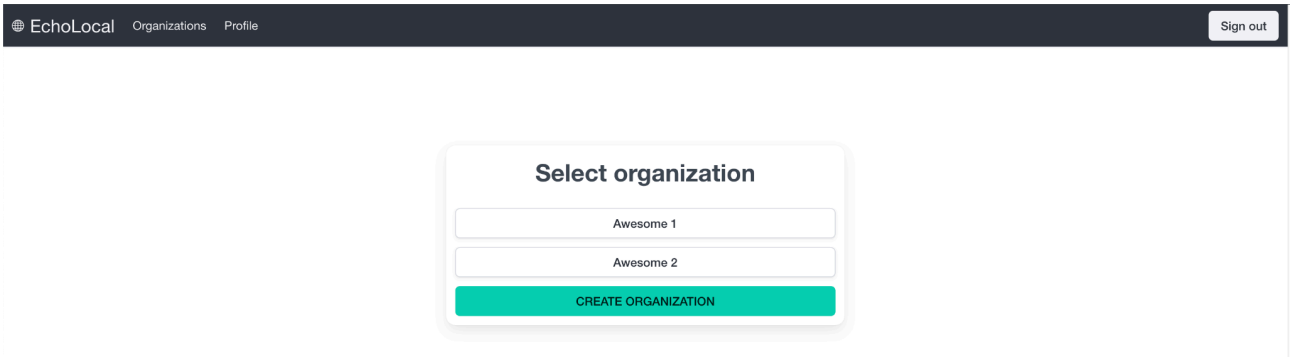
Сторінка реєстрації дозволяє новим користувачам створити обліковий запис, ввівши необхідні дані: ім'я, прізвище, електронну пошту, пароль і підтвердження пароля. Після введення даних користувач натискає кнопку *Sign Up* для створення облікового запису. При виникненні помилки чи неспівпадінні паролів буде показана помилка. Тут планується додавання нових полів, щоб зазначити хто саме реєструється, розробник, перекладач або власник, який зможе керувати.

- Сторінка входу *(Sign in page)*: */signin*



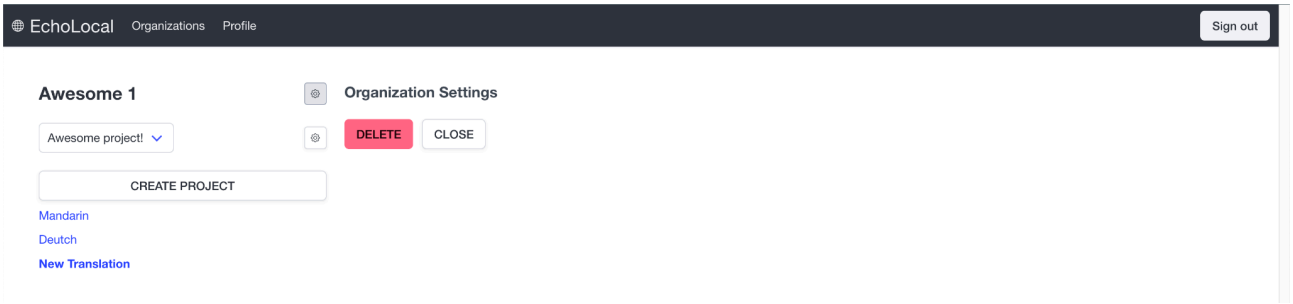
Сторінка входу дозволяє існуючим користувачам увійти в систему, ввівши електронну пошту та пароль. Після введення даних користувач натискає кнопку *Sign In* для автентифікації.

- Сторінка вибору організації *(Select organization)*: */organizations*



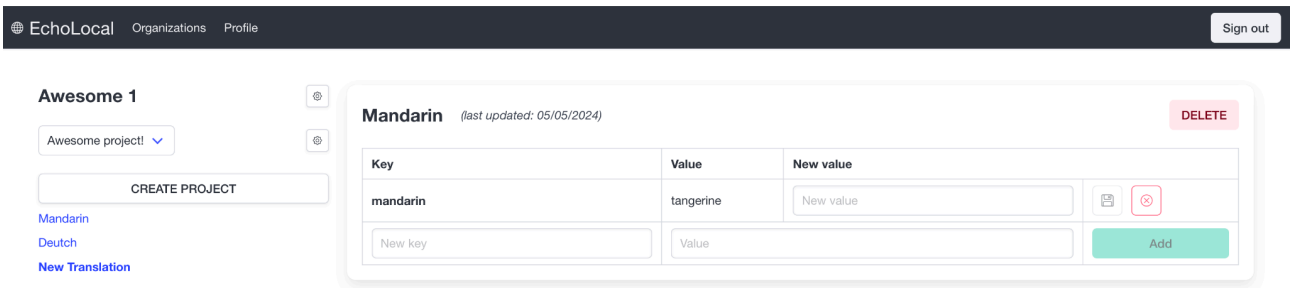
Після входу в систему користувач потрапляє на сторінку вибору організації. Тут можна вибрати існуючу організацію або створити нову через кнопку *CREATE ORGANIZATION*.

- Сторінка налаштувань організації *(Organization settings)*: */organizations/6*



Сторінка налаштувань організації дозволяє користувачу керувати організацією, включаючи її видалення. Сторінка у процесі розробки, тут будуть доступна зміна назва або додавання інших даних до організації, управління працівниками. Наразі доступне тільки видалення через кнопку *DELETE*.

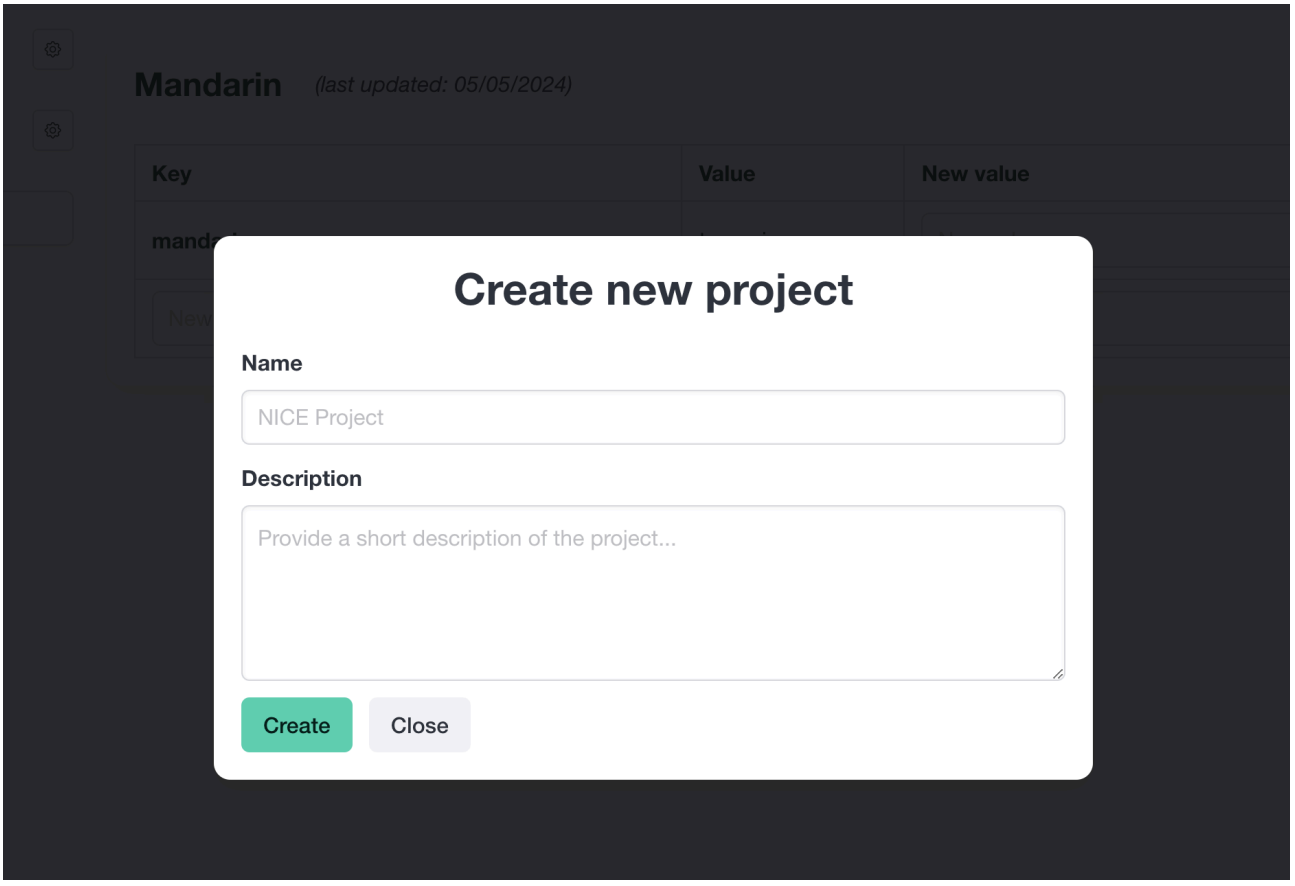
- Панель управління або дашборд *(Dashboard)*: */organizations/6*



Панель управління надає користувачу доступ до управління проєктами та перекладами. Це головна сторінка сервісу, де відбувається основна взаємодія і тут доступний наступний функціонал:

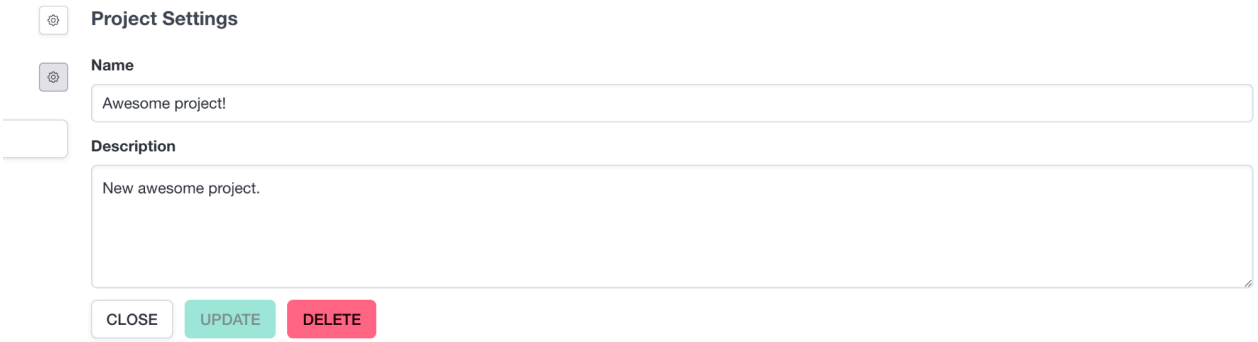
- **Вибір проекту:** можливість вибрати проект з випадаючого списку;
- **Список перекладів:** перегляд існуючих перекладів для обраного проекту;
- **Вибір листа перекладу:** вибір конкретного листа перекладу для детальної роботи із ним;
- **Додавання нового перекладу:** користувач може додати новий ключ перекладу та його значення;
- **Зміна та видалення:** можливість зміни або видалення існуючого перекладу.

- Сторінка створення проєкту *(Create project)*: */organizations/6*



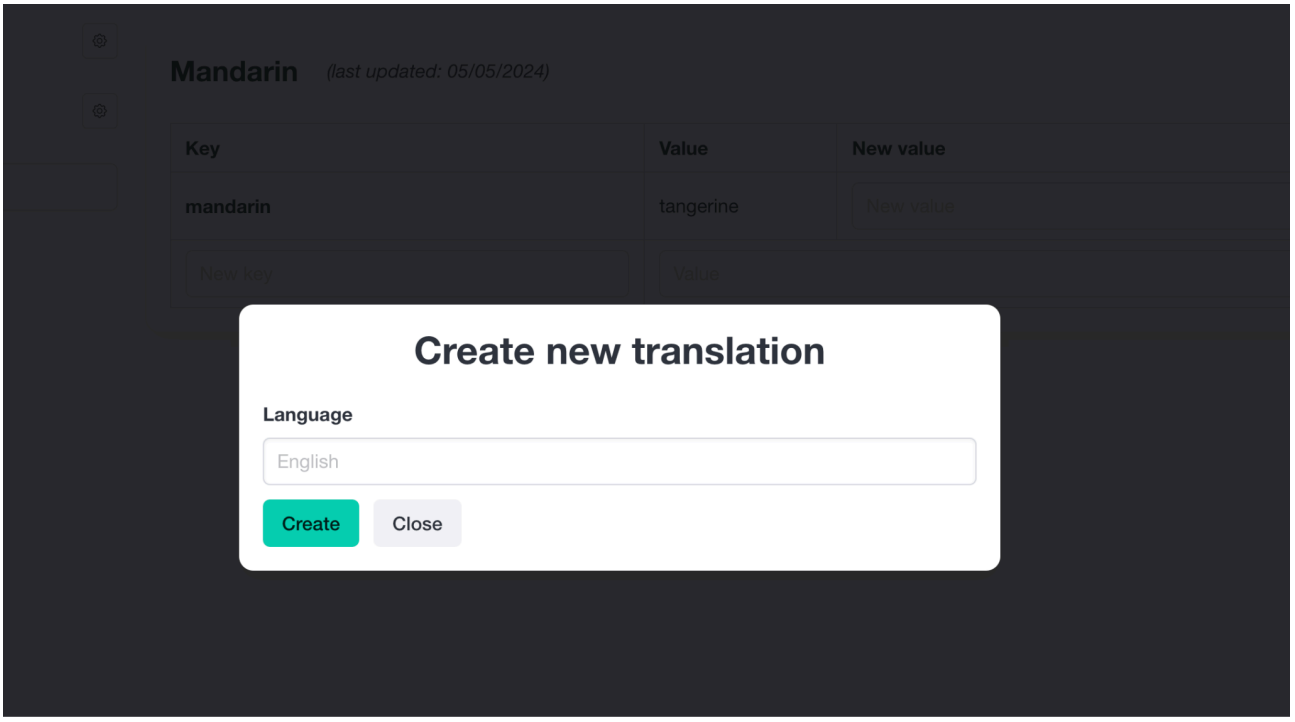
Користувач може створити новий проект, ввівши його назву та опис. Після введення даних користувач натискає кнопку "Create" для створення проекту. На даний момент це всі дані, які обов’язкові при створенні і допускається додавання нових деталей на сторінці налаштувань. Там же має бути введений менеджмент учасників проєкту, закріплення ролей та проєктного менеджера.

- Сторінка налаштувань проєкту *(Project settings)*: */organizations/6*



Сторінка налаштувань проекту дозволяє користувачу змінювати назву та опис проекту, а також видаляти проект.

- Сторінка створення листа перекладу *(Create translation sheet)*: */organizations/6*



Користувач може створити новий переклад, вибравши мову для перекладу. Назва мови може бути будь-якою, навіть унікальною і відомою малому колу людей - вигадана або із кінофільму. Принцип роботи залишається таким самим.

Цей детальний огляд функціональних можливостей **EchoLocal** дозволяє зрозуміти, як саме користувач взаємодіє із системою та які можливості вона

надає для управління процесом локалізації. Детальний шлях користувача і практичне застосування системи буде розглянуто у наступних підрозділах.

3.4 Практичне застосування та переваги використання сервісу

Для того щоб висвітлити практичне застосування та прогнозовані переваги використання саме сервісу **EchoLocal** давайте розглянемо шлях потенційного користувача сервісу - власника бізнесу, стартапу або іншого сервісу, та його очікування. Варто зазначити, що наступний перелік містить деякі елементи, які не доступні та нереалізовані у поточному стані системи, але планується їх реалізація у процесі розвитку.

Шлях користувача EchoLocal

1. Реєстрація та налаштування облікового запису:
    a. Користувач *(власник проєкту)* реєструється в сервісі **EchoLocal**, вводячи основні дані, такі як ім'я, прізвище, електронну адресу та пароль;
    b. Після підтвердження електронної адреси користувач створює організацію та проєкт, що потребує локалізації.
2. Додавання команди:
    a. Власник проєкту запрошує інших учасників до своєї організації: розробників, перекладачів, менеджерів проєкту;
    b. Ролі розподіляються таким чином, щоб кожен член команди мав доступ до відповідних функцій сервісу: розробники можуть додавати нові ключі, перекладачі – перекладати тексти, менеджери – контролювати процес.
3. Створення та управління ключами локалізації:
    a. Розробники додають ключі для локалізації в сервіс, визначаючи текстові елементи, які потребують перекладу;
    b. Кожен ключ включає текст - значення, що потребує перекладу;
    c. Додаються коментарі для перекладачів та інша необхідну інформація для розуміння контексту.

4. Переклад контенту:
    a. Перекладачі отримують доступ до ключів локалізації та починають переклад текстів на необхідні мови;
    b. Інтерфейс **EchoLocal** дозволяє перекладачам бачити контекст кожного ключа та отримувати інструкції від розробників.
5. Перевірка та затвердження перекладів:
    a. Менеджери проєкту перевіряють переклади та затверджують їх для подальшого використання;
    b. Власник проєкту може також перевіряти статистику перекладів, щоб оцінити прогрес локалізації.
6. Інтеграція локалізованих даних:
    a. Розробники інтегрують локалізовані тексти в проєкт, використовуючи API сервісу **EchoLocal**, який дозволяє автоматично завантажувати перекладені ключі;
    b. Всі зміни синхронізуються в реальному часі, забезпечуючи актуальність контенту.

Переваги використання EchoLocal

Гнучкість та масштабованість	- **EchoLocal** підтримує різні технології та платформи, що дозволяє легко інтегрувати сервіс у будь-який проєкт; - Завдяки можливостям масштабування, **EchoLocal** підходить як для малих стартапів, так і для великих компаній з багатьма проєктами та мовами.
Централізоване управління локалізацією	- Всі ключі локалізації та переклади зберігаються в одному місці, що забезпечує легкий доступ та управління;

	- Централізований контроль дозволяє уникнути дублювання зусиль та помилок, забезпечуючи єдність контенту.
Реальний час та автоматизація	- Завдяки можливостям синхронізації в реальному часі, всі зміни в локалізації миттєво відображаються в проєкті; - API сервісу дозволяє автоматизувати процес завантаження та оновлення перекладених текстів, знижуючи потребу у ручній праці.
Ефективне управління командою	- Власник проєкту може легко додавати та керувати ролями членів команди, забезпечуючи ефективну співпрацю між розробниками, перекладачами та менеджерами; - Інтуїтивно зрозумілий інтерфейс дозволяє кожному члену команди швидко зрозуміти свої задачі та розпочати роботу.
Висока якість перекладів	- Наявність коментарів та контексту для перекладачів значно підвищує якість перекладів, зменшуючи кількість помилок та непорозумінь; - Можливість затвердження перекладів менеджерами забезпечує додатковий контроль якості.

Сервіс **EchoLocal** – це потужний інструмент для управління локалізацією, який забезпечує гнучкість, ефективність та високу якість перекладів. Він дозволяє знизити витрати та час на локалізацію, підвищуючи при цьому загальну ефективність роботи команди. Використовуючи його, компанії можуть

легко адаптувати свої продукти під нові ринки, розширюючи свою аудиторію та збільшуючи доходи.

3.5 Майбутні напрямки розвитку та удосконалення EchoLocal

Звичайно, у рамках цієї роботи не були досягнуті усі бажання та ідеї стосовно тематики дослідження. Більший упор був на дослідження проблеми та методи можливого покращення та підвищення ефективності крос-командної співпраці у роботі над локалізацією веб-рішень. А тому, практична частина роботи - веб-застосунок **EchoLocal** демонструє базові можливості роботи та підхід до роботи на основі якого може рости і вдосконалюватись цілковито ринкове рішення, яке змогло б зайняти свою нішу. Серед можливих ідей покращень сервісу я бачу такі:

Вдосконалення одночасної роботи. Наразі програма працює таким чином, що чиї оновлення будуть записуватись пізніше того і правда. Це звучить логічно, коли працює лише один перекладач на одному проєкті як зараз, але із ростом популярності і складності проєктів для перекладу, можливі накладки і перезапис даних над якими ведеться робота. Як рішення розглядаються варіанти:

- Версіонування листів перекладу - при такому підході кожна робота буде збережена під своєю версію і у випадку накладок, потрібно буде змінити версію та переглянути зміни. При такому підході можна буде легко навіть відновити дані.
- Живе підлаштування. Мається на увазі перевірка даних, які перезаписуються. У випадку, якщо перекладач працює над тією частиною, яка щойно змінилась, то будуть запропоновані зміни і потрібно буде одразу обрати який варіант записати у базу.

Додавання контексту. Як було згадано у роботі, перекладачам дуже сильно допомагає контекст у використаний той або інший текст. Це дуже позитивно впливає на загальну локалізацію, тримає правильний тон та веде погоджений діалог із користувачами. Розглядається можливість додавання

картинок, фотоматеріалів або нотаток для додаткового роз’яснення, щодо використання конкретних частин тексту.

Введення ролей та покращення контролю за доступом і можливостями. Мається на увазі введення ролей користувачів та закріплення за ними певних можливостей, які доступні на порталі. Це стане більш актуально із збільшенням аудиторії та розмірів проєктів, які будуть на порталі. Наприклад, надання можливості редагування лише певних частин контенту або обмеження доступу до конфіденційної інформації може стати важливою частиною регуляції та контролю доступу.

Фіксування переліку мов для перекладачів. Необхідно додати фіксацію мов, які будуть доступні перекладачам для роботи. Наразі це не реалізовано, тож будь-який перекладач умовно може перекласти або підтвердити переклад будь-якого контенту. Тут однозначно потрібна регуляція і чіткий розподіл можливостей.

Живий переклад. Дуже сильною перевагою серед подібних сервісів було б додати переклад у реальному часі. Тобто всі ключі завантаженні та сервіс підключений для проєкту, а при зміні самого перекладу вони одразу відобразяться у проєкті. Ніяких перезавантажень сторінки не потрібно. Це можливо зробити із ускладненням підключення сервісу та проєкту за допомогою WebSockets або long-polling підходів. Як рішення, я б розглянув створення спеціально налаштованих бібліотек для популярних технологій які вже мали такий функціонал під капотом.

Розробка інструментів тестування локалізації. Нерідко бувають ситуації, коли дизайн веб-сторінок однією мовою абсолютно не включає можливі зміни або порушення структури іншими текстами. Було б цікаво реалізувати інструмент, який допоміг би розробникам та перекладачам перевіряти, як виглядає локалізований контент у різних мовних версіях. Це допомогло б виявити проблеми зі структурою сторінки, довжиною тексту та інші можливі невідповідності.

Трохи машинного навчання та перевикористання перекладів. Також було б цікаво додати чорнові переклади, які б створювались би автоматично в залежності від мови самого листа перекладу та базового тексту. Таким чином перекладач вже бачив би машинний переклад і лише вдосконалював би сам переклад, підганяв би його під контекст. Також розглядається перевикористання подібним чином повторюваних фраз та висловів, для збереження ресурсу.

Інтеграція з програмами менеджменту проєктів. Подібна інтеграція із Jira, Trello або подібними дозволить менеджерам проєктів відстежувати процес локалізації, призначати завдання та координувати роботу.

І це лише перелік тих покращень, які на мою думку на поверхні. На додачу можна додати ще підтримку локалізацію самим сервісом **EchoLocal**, підтримку мобільних платформ або мобільний застосунок, автооновлення контенту при зміні і так далі. Із розвитком сервісу точно будуть виникати нові ідеї та кращі способи рішення вже відомих проблем. Головне тримати темп та бути на зв’язку із самими користувачами, адже їх фідбек - це найголовніше, що ми можемо отримати у наш сервісний час.

Висновки по роботі та рекомендації для подальших досліджень

У даній роботі було детально розглянуті різні підходи до локалізації сучасних веб-застосунків, їхні переваги та недоліки, а також була розроблена концепція нового сервісу для локалізації - **EchoLocal**. Сервіс покликаний значно покращити процес та зайняти свою нішу серед подібних рішень на ринку, був частково реалізований у практичній частині роботи.

Сервіс **EchoLocal** був розроблений з урахуванням основних недоліків існуючих підходів і надає наступні переваги:

- **Гнучкість та масштабованість.** Підтримує різні технології та платформи, що дозволяє легко інтегрувати сервіс у будь-який проєкт. Точніше сказати, що він незалежний від технологій та платформ, адже надає API для роботи, яка у свою чергу може бути побудована різним чином - вручну налаштована або із використанням сторонніх бібліотек. Завдяки можливостям масштабування, **EchoLocal** підходить як для малих стартапів, так і для великих компаній з багатьма проєктами та мовами.
- **Централізоване управління локалізацією.** Всі ключі локалізації та переклади зберігаються в одному місці, що забезпечує легкий доступ та управління. Це дозволяє уникнути повторення помилок та неефективної співпраці, забезпечуючи цілісність контенту.
- **Ефективне управління командою.** Менеджер проєкту або власник організації може легко додавати та керувати ролями членів команди, забезпечуючи ефективну співпрацю між розробниками, перекладачами та іншими менеджерами. Інтуїтивно зрозумілий інтерфейс дозволяє кожному члену команди швидко зрозуміти свої задачі та розпочати роботу. На даному етапі реалізації ролі не підтримуються, але запланованими до реалізації у наступних ітераціях.
- **Розуміння контексту.** Наявність нотаток та контексту для перекладачів значно підвищує якість перекладів, зменшуючи кількість помилок та недорозумінь. Можливість затвердження перекладів менеджерами забезпечує додатковий контроль якості.

- **Реальний час та автоматизація.** Завдяки можливостям синхронізації в реальному часі, всі зміни в локалізації миттєво відображаються в проєкті. API сервісу дозволяє автоматизувати процес завантаження та оновлення перекладених текстів, знижуючи потребу у ручній праці. Дана особливість працюватиме при правильному першому налаштуванні як було згадано у пункті 3.5 цієї роботи.

Проблематика локалізації у майбутньому

З розвитком технологій і збільшенням кількості цифрового контенту локалізація стає ще більш важливим аспектом для компаній, які прагнуть виходити на міжнародні ринки.

Основними викликами в майбутньому на мою думку стануть:

- Автоматизація та штучний інтелект - подальший розвиток технологій машинного перекладу та штучного інтелекту дозволить значно покращити якість автоматичних перекладів, знижуючи потребу у ручному редагуванні. Однак, це вимагатиме розробки нових алгоритмів та методик для забезпечення точності і контексту перекладів;
- Інтеграція з новими технологіями - з появою нових технологій і платформ, локалізаційні сервіси повинні будуть адаптуватися до змін, забезпечуючи підтримку різноманітних форматів даних та методів інтеграції;
- Глобалізація та персоналізація - збільшення кількості глобальних сервісів, які будуть доступні у всьому світі потребуватиме більш персоналізованого підходу до локалізації, до спілкування з користувачем, враховуючи культурні особливості та специфіку кожного регіону. Це створить нові виклики для забезпечення відповідності контенту очікуванням користувачів.
- Забезпечення безпеки даних - в умовах зростаючої кількості кіберзагроз, локалізаційні сервіси повинні забезпечувати високий рівень безпеки даних, включаючи захист конфіденційної та маркетингової інформації.

EchoLocal, як сучасний інструмент для управління локалізацією, забезпечує гнучкість, масштабованість та високу якість перекладів, що робить його ідеальним рішенням для компаній, які прагнуть ефективно локалізувати свій контент і вийти на нові ринки. Однак, подальші дослідження та розвиток технологій залишаються важливими для подолання майбутніх викликів у сфері.

Список літератури

1. Інтернаціоналізація та локалізація. [Електронний ресурс] – Режим доступу: https://qalight.ua/baza-znaniy/internatsionalizatsiya-ta-lokalizatsiya/
2. Ilya Krukowski. Internationalization vs. localization (i18n vs l10n): What's the difference? [Електронний ресурс] – 2023 – Режим доступу: https://lokalise.com/blog/internationalization-vs-localization/
3. Nataly Kelly. Speak to Global Customers in Their Own Language [Електронний ресурс] – 2012 – Режим доступу: https://hbr.org/2012/08/speak-to-global-customers-in-t
4. Які проблеми може спричинити неправильна локалізація сайту. [Електронний ресурс] – 2023 – Режим доступу: https://philin.com.ua/ua/blog/2023/09/25/problemy-nepravilnoj-lokalizacii-sajta/
5. Raymond Cheng. How to choose a localization approach for your React application. [Електронний ресурс] – 2022 – Режим доступу: https://www.plasmic.app/blog/react-localization
6. Документація сервісу Localazy: https://localazy.com/docs/general/getting-started-with-localazy
7. Документація сервісу Contentful: https://www.contentful.com/help/contentful-101/
8. Документація сервісу Gtranslate: https://gtranslate.io/docs/58-gtranslate-tdn-documentation
9. Репозиторій проєкту: https://github.com/vlvereta/EchoLocal
10. Офіційний сайт GitHub: https://github.com/
11. Документація GitHub Actions: https://docs.github.com/en/actions
12. Офіційний сайт React: https://react.dev/
13. Офіційний сайт Preact: https://preactjs.com/
14. Офіційний сайт менеджера пакетів npm: https://www.npmjs.com/
15. Офіційний сайт менеджера пакетів yarn: https://yarnpkg.com/
16. Офіційний сайт Bulma: https://bulma.io/

17. Офіційний сайт Docker: https://www.docker.com/
18. Офіційний сайт Google Cloud Platform: https://cloud.google.com/
19. Офіційний сайт Kubernetes: https://kubernetes.io/
20. Офіційний сайт інструменту конфігурації Kustomize: https://kustomize.io/
21. Сервіс для візуалізації схеми бази даних DBML: https://dbdiagram.io/home/
22. Donald A. DePalma, Benjamin B. Sargent, and Renato S. Beninatto. Can’t Read, Won’t Buy: Why Language Matters on Global Websites. [Електронний ресурс] – 2006 – Режим доступу: https://motsdici.be/wp-content/uploads/2019/04/Article-cant-read-wont-buy.pdf