МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

На правах рукопису

АНДРУЩУК ПАВЛО ЄВГЕНОВИЧ

ВИКОРИСТАННЯ ТЕХНОЛОГІЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗРОБКИ СЕРСІВУ АВТОМАТИЗОВАНОГО ПЕРЕКЛАДУ ТЕКСТІВ

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
ГРИШАНОВИЧ ТЕТЯНА ОЛЕКСАНДРІВНА
кандидат фіз.-мат наук, доцент кафедри
комп’ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ

Протокол №__
засідання кафедри комп’ютерних наук
та кібербезпеки від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т. О.

ЛУЦЬК-2025

ЗМІСТ

ВСТУП

У сучасному світі, ознаменованому цифровою революцією та доступом до глобального культурного простору, особливо до музичного контенту, проблема розуміння іншомовних текстів залишається актуальною. Традиційні підходи до перекладу, часто вимагають значних людських ресурсів для передачі не лише буквального змісту, а й стилістичних особливостей, метафор та контексту. Це створює бар'єр для багатьох слухачів, обмежуючи їхнє розуміння зарубіжних пісень.

Однак стрімкий розвиток технологій штучного інтелекту (ШІ) відкриває принципово нові способи вирішення цієї проблеми. Саме можливість використання алгоритмів машинного навчання та нейронних мереж для автоматизованої обробки та перекладу текстів становить актуальність даної роботи. ШІ-технології здатні значно прискорити процес створення перекладів та обробляти великі масиви даних. Також, при додатковому тренеруванні моделей, можна добавити контекст перекладених тексті, покращуючи оброблення стилістичних особливостей штучним інтелектом. Це не лише спрощує процес створення перекладів, дозволяючи ширшому колу ентузіастів долучатися до нього, але й додає професійним перекладачам зручний інструмент для перекладу складних моментів.

Незважаючи на потужність сучасних ШІ-моделей, потреба у спеціалізованих вебдодатках, що інтегрують ці технології, залишається високою. Розробка сучасних вебрішень, що поєднують можливості ШІ з інтуїтивно зрозумілим функціоналом для користувачів, дозволяє ефективно використовувати потенціал новітніх технологій для автоматизації ресурсоємних задач.

Мета роботи – розробити вебдодаток для створення перекладів, використовуючи технології штучного інтелекту, що дозволить доповнити існуючу базу новими перекладами, та забезпечити звичайних користувачів матеріалом для перегляду.

Для досягнення поставленої мети було сформульовано наступні **завдання:**

- ознайомитися з сучасними тенденціями веброзробки;
- розглянути використання технологій штучного інтелекту у перекладі текстів;
- проаналізувати аналогічні розробки, визначити їхні ключові особливості та наявні обмеження;
- обрати модель розробки програмного забезпечення;
- визначити інструменти та засоби для вирішення проблеми;
- розробити всі необхідні модулі для коректної роботи додатку, такі як модуль музики, перекладів, користувачів;
- розробити інтерфейс відповідно до сучасних стандартів дизайну;
- провести тестування усіх ключових функціоналів вебдодатку, виправити наявні недоліки;
- сформувати документацію, рекомендації щодо використання програмного продукту.

Об’єктом дослідження є вебдодатки для перегляду та створення перекладів текстів пісень.

Предметом дослідження є процес проєктування та розробки платформи для перегляду й створення перекладів, за допомогою технологій штучного інтелекту.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Сучасні технології веброзробки

Сучасна веброзробка – це динамічна сфера, що безперервно еволюціонує, впроваджуючи нові технології, інструменти та методології розробки. Цей стрімкий розвиток зумовлений зростаючими вимогами до функціональності, продуктивності, безпеки та користувацького досвіду вебзастосунків. Головні тенденції спрямовані на створення більш інтерактивних, швидких та масштабованих рішень.

Однією з ключових тенденцій є популярність JavaScript-фреймворків та бібліотек для розробки фронтенду. Серед найпопулярніших інструментів виділяються React, Vue.js та Angular. [1] React відомий своєю гнучкістю, компонентним підходом та великою екосистемою, що дозволяє створювати складні інтерфейси. Vue.js приваблює розробників своєю простотою розуміння, прогресивністю та високою продуктивністю. Angular, є повноцінним фреймворком, пропонує комплексне рішення для великих корпоративних застосунків, включаючи такі інструменти, як TypeScript для статичної типізації та RxJS для реактивного програмування. Всі ці інструменти дозволяють створювати динамічні користувацькі інтерфейси (UI), часто у вигляді односторінкових застосунків, які забезпечують позитивний користувацький досвід, оскільки оновлення контенту відбувається без перезавантаження всієї сторінки. Це робить фронтенд більш незалежним від серверної частини, взаємодіючи з нею переважно через API (програмний інтерфейс).

У сфері серверної розробки (бекенду) також відбуваються значні зміни. PHP-базовані серверні розробки, хоча й мають багату історію та все ще використовуються в багатьох проєктах, особливо у старих, для яких міграція на нову систему не є вигіднішим, ніж продовження розробки на PHP, поступово поступаються місцем новішим технологіям у контексті розробки складних, високопродуктивних систем. Це пов’язано з архітектурними особливостями,

залученістю спільноти розробників та стрімким розвитком альтернатив. Натомість, технології на основі Python (з фреймворками як Django, FastAPI, Flask) або Node.js (з фреймворками Express.js, Nest.js) все частіше стають основою для сучасних вебдодатків. [2]

Python приваблює своєю простотою синтаксису, великою кількістю бібліотек (зокрема для аналізу даних та машинного навчання) та потужними фреймворками. Django пропонує великий набір вбудованого функціоналу, що прискорює розробку типових функцій, таких як ORM, адмін-панель, система автентифікації. FastAPI стрімко набирає популярність завдяки своїй високій продуктивності (заснованій на асинхронності з asyncio та uvicorn), автоматичній генерації документації API (OpenAPI) та використанню типізації Python. Таким чином, дозволяє розробляти швидкі мікросервіси.

Node.js, що дозволяє виконувати JavaScript на сервері, став популярним завдяки можливості використання однієї мови для фронтенду та бекенду (full-stack JavaScript), а також своїй неблокуючій, event-driven архітектурі, що ідеально підходить для застосунків реального часу та високонавантажених систем. Express.js є стандартом для Node.js, мінімалістичним та гнучким фреймворком, що надає міцну основу для створення API та вебзастосунків.

Завдяки простоті їхнього розгортання (часто з використанням контейнеризації, наприклад, Docker) та подальшої розробки, гнучкості в реалізації різноманітних архітектурних патернів, а також високій продуктивності та підтримці сучасних підходів до розробки (таких як CI/CD, тестування, моніторинг), ці технології забезпечують швидку інтеграцію з хмарними сервісами, ефективну реалізацію мікросервісної архітектури та роботу з масштабованими базами даних (як реляційними, так і не реляційними). Це робить їх ідеальним вибором для сучасних вебзастосунків, які мають бути надійними, гнучкими та здатними витримувати великі навантаження.

Оскільки кожен сервіс, чи то вебсайт, API або складний корпоративний портал, потребує розгортання для загального чи корпоративного доступу, хмарні сервіси стали невід’ємною частиною сучасної інфраструктури. Вони

пропонують масштабованість, надійність, глобальну доступність та широкий спектр керованих послуг, що спрощують розгортання та управління застосунками. Лідерами хмарних сервісів на сьогоднішній день є Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure. [3] Кожна з цих платформ надає великий набір інструментів для обчислень (віртуальні машини, контейнерні сервіси, безсерверні функції), зберігання даних (об’єктні сховища, реляційні та NoSQL бази даних), мережевих рішень, інструментів для машинного навчання, аналітики та багато іншого, що дозволяє компаніям будувати та підтримувати інфраструктуру будь-якої складності без необхідності інвестувати у власне фізичне обладнання.

1.2 Архітектура вебзастосунків

Розробка сучасних вебзастосунків нерозривно пов’язана з ретельним вибором архітектурного підходу та відповідного технологічного стека. Це рішення є одним з найважливіших на початкових етапах проєктування, оскільки воно фундаментально впливає на весь життєвий цикл продукту: від швидкості розробки та розгортання до можливостей масштабування, надійності, зручності підтримки та подальшого розвитку. Кожен архітектурний підхід має свої унікальні переваги та неминучі недоліки, і оптимальний вибір залежить від специфічних вимог проєкту, таких як очікувана продуктивність, необхідність горизонтального чи вертикального масштабування, складність бізнес-логіки, розмір та досвід команди розробників, а також бюджетні обмеження.

Архітектура вебзастосунків визначає фундаментальну структуру системи: як організовані її ключові компоненти (наприклад, користувацький інтерфейс, бізнес-логіка, доступ до даних), як ці компоненти взаємодіють між собою, та якими є принципи їхнього розгортання та експлуатації. Розглянемо детальніше основні архітектурні підходи, що використовуються при розробці вебзастосунків. [4]

Монолітна архітектура – традиційний підхід до побудови застосунків, де

всі функціональні компоненти – користувацький інтерфейс (фронтенд), серверна логіка (бекенд), модулі взаємодії з базою даних та інші додаткові сервісні модулі – об’єднані в єдину, цілісну кодову базу та розгортаються як єдина програмна розробка.

Характеристика:

- усі компоненти застосунку (бекенд, фронтенд, база даних) розташовані в єдиній кодовій базі;
- використовується у випадках, коли важливими є швидкість розробки, оскільки немає складнощів з міжсервісною взаємодією;
- завдяки присутності всіх компонентів у одному контексті, полегшується End-to-end тестування;
- простота у розгортанні є менш складним, ніж управління багатьма окремими мікросервісами.

Недоліки:

- із зростанням кодової бази та масштабами проекту стає важко орієнтуватися, вносити зміни та розібратися у системі;
- важко інтегрувати нові технології, оскільки весь моноліт зазвичай побудований на одному стеку;
- обмежені можливості з масштабування, яке зазвичай запускає нові екземпляри всього застосунку, навіть коли проблема лише з одним конкретним модулем;
- будь-яка незначна зміна вимагає перезапуску усього застосунку, та збільшує ризики регресії (коли протестований код містить помилки у зв’язку з невірною поведінкою залежних частин).

Мікросервісна архітектура – підхід, при якому застосунок структурується як набір невеликих, незалежних та слабко зв’язаних сервісів. Кожен сервіс відповідає за конкретну бізнес-функцію, має власну кодову базу, може розроблятися, розгортатися та масштабуватися незалежно від інших.

Характеристики:

- система поділена на незалежні сервіси, тобто децентралізація;

- кожен сервіс виконує специфічну, характерну йому функцію;
- розгортання чи оновлення одного сервісу не впливає на інші;
- можливість вибору різних технологій для кожного сервісу, забезпечуючи найкращу взаємодію та продуктивність;
- незалежне масштабування дозволяє розширювати тільки ті сервіси, які отримують найбільше навантаження;
- збій одного сервісу не обов’язково призведе до відмови інших – вони можуть продовжити свою роботу.

Недоліки:

- необхідність реалізації надійної комунікації сервісів, через синхронні REST API чи gRPC або асинхронні Kafka чи RabbitMQ, є зазвичай складним завданням;
- складність End-to-end тестування, через розподілення багатьох сервісів які взаємодіють окремо;
- розгортання, моніторинг та управління цією системою вимагає застосування складніших інструментів та інфраструктури (контейнеризація за допомогою Docker, управління за допомогою Kubernetes, системи моніторингу та логування, наприклад Grafana;
- необхідність застосування комплексніших інструментів взаємодії з транзакціями.

1.3 Використання технологій штучного інтелекту для автоматизації перекладів

Штучний інтелект (ШІ) відіграє ключову роль в автоматизованому перекладі текстів, значно покращуючи якість та швидкість обробки мовних даних. Сучасні нейромережеві моделі, такі як трансформери, здатні аналізувати контекст та адаптувати переклад відповідно до стилю тексту. [5] Завдяки цьому якість машинного перекладу поступово наближається до рівня професійних

лінгвістів.

Одним із найпотужніших інструментів для автоматизованого перекладу є ChatGPT – нейромережа, заснована на архітектурі GPT (Generative Pre-trained Transformer), розроблена компанією OpenAI. [6] Вона може виконувати такі завдання, як автоматичний переклад текстів різними мовами, аналіз контексту та стилю мовлення та безліч інших дій в усіх сферах.

OpenAI надає API для роботи з ChatGPT, який можна інтегрувати у вебзастосунок. Для цього використовується REST-запит до API OpenAI з передачею вхідного тексту та отриманням перекладеного результату у відповідь.

Fine-tuning (донавчання моделі) – це процес адаптації нейромережевої моделі до специфічних завдань шляхом її тренування на спеціалізованих даних[7]. У контексті автоматизованого перекладу fine-tuning дозволяє покращити точність перекладу за рахунок навчання на великій базі текстів певного жанру або тематики, пристосувати модель до стилю мови.

API OpenAI надає засоби для навчання моделей, що можна робити за допомогою власної бази паралельних текстів (оригінал та переклад).

1.4 Огляд аналогічних програмних розробок

1.4.1 Genius

Genius.com – американський вебсайт створений для поширення музичної інформації мережею інтернет. [8]

Сайт пропонує широкий набір функціональних можливостей:

- база більшості існуючих пісень;
- пошук по платформі по усім критеріям, такі як текст пісні, її виконавець чи альбом;
- реєстрація користувачів та доступ до особистого кабінету;
- можливість створення перекладів текстів пісень;
- написання коментарів під текстами пісень;

- додавання анотацій під оригінальний чи перекладений текст пісні.

Дана розробка є успішною та зручною у використанні. Величезний набір даних та активна аудиторія дозволяє безперестанку находити потрібні пісні, а в деяких випадках, переклад до них на потрібну мову. Хоча це не так популярно, так як, як правило, основним завдання платформи є поширення оригінального тексту й супутньої інформації.

Проглянувши їхній вебсайт, структуру вебсторінок та заголовки запитів, які повертає серверна частина та сторінку з робочими вакансіями, з великою ймовірністю можна дізнатися про засоби, використані у розробці. Основною особливістю, є використання Ruby on Rails – веб-орієнтований фреймворк для мови програмування Ruby. Він, як й Django, дозволяє створювати сучасні системи, працюючи з базою даних, шаблонами сторінок та асинхронністю [9].

Інтерфейс сторінки пісні (рис. 1.1) містить оригінальний текст, зображення обкладинки пісні та супутню інформацію, таку як виконавець, дата випуску та інше. При натисканні кнопки з перекладами, рідко можна знайти українську мову. Відповідно, мета роботи вирішує цю проблему, а саме, ставить цільовою аудиторією саме україномовних користувачів.

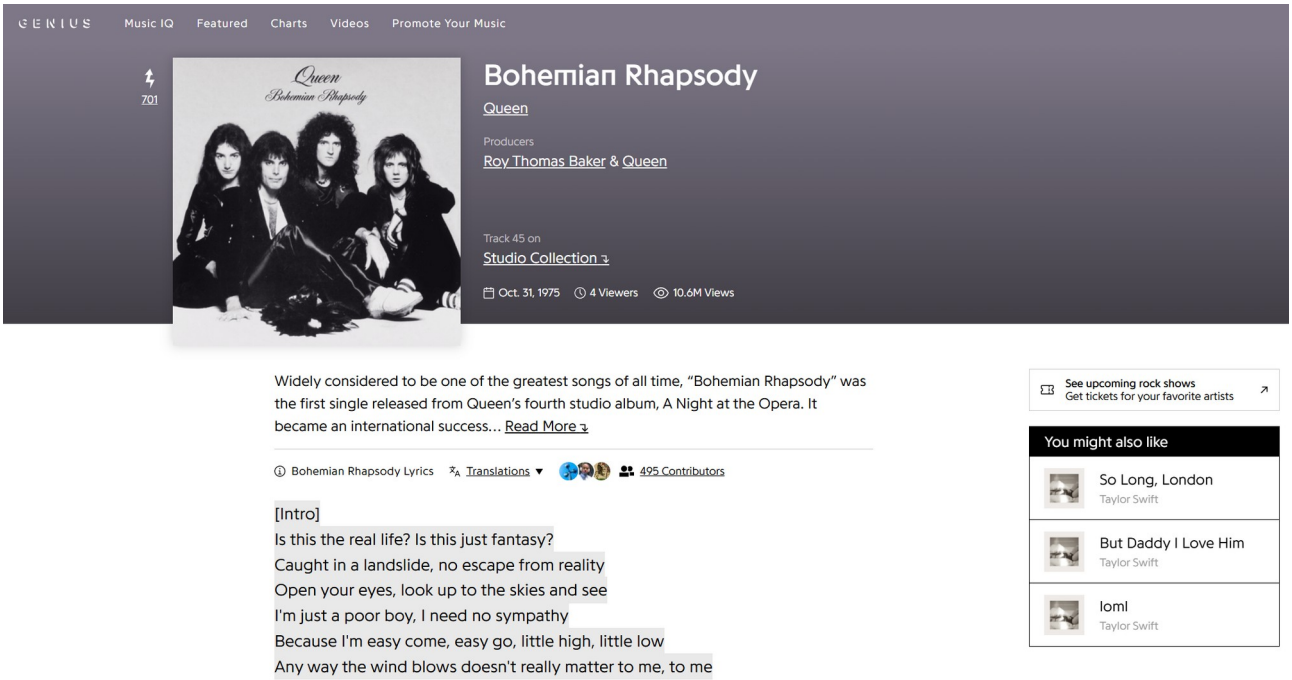


Рисунок 1.1 – Сторінка тексту пісні на платформі Genius.com

Розглянемо також інтерфейс створення перекладу. Він доступний будь-якому зареєстрованому користувачеві. Початково, це інтерфейс додавання оригінальної пісні та її тексту, але ця ж сторінка використовується для створення перекладів. Доступні для користувача кілька полів для введення, такі як назва, виконавець, жанр, текст та ще декілька додаткових, не обов’язкових полів. На рисунку 1.2 зображено сторінку створення. Як можна помітити, при створенні перекладу, не має змоги паралельно переглядати оригінальний текст. Також платформа не пропонує ніяких інструментів для допомоги у перекладі, що є суттєвим мінусом.

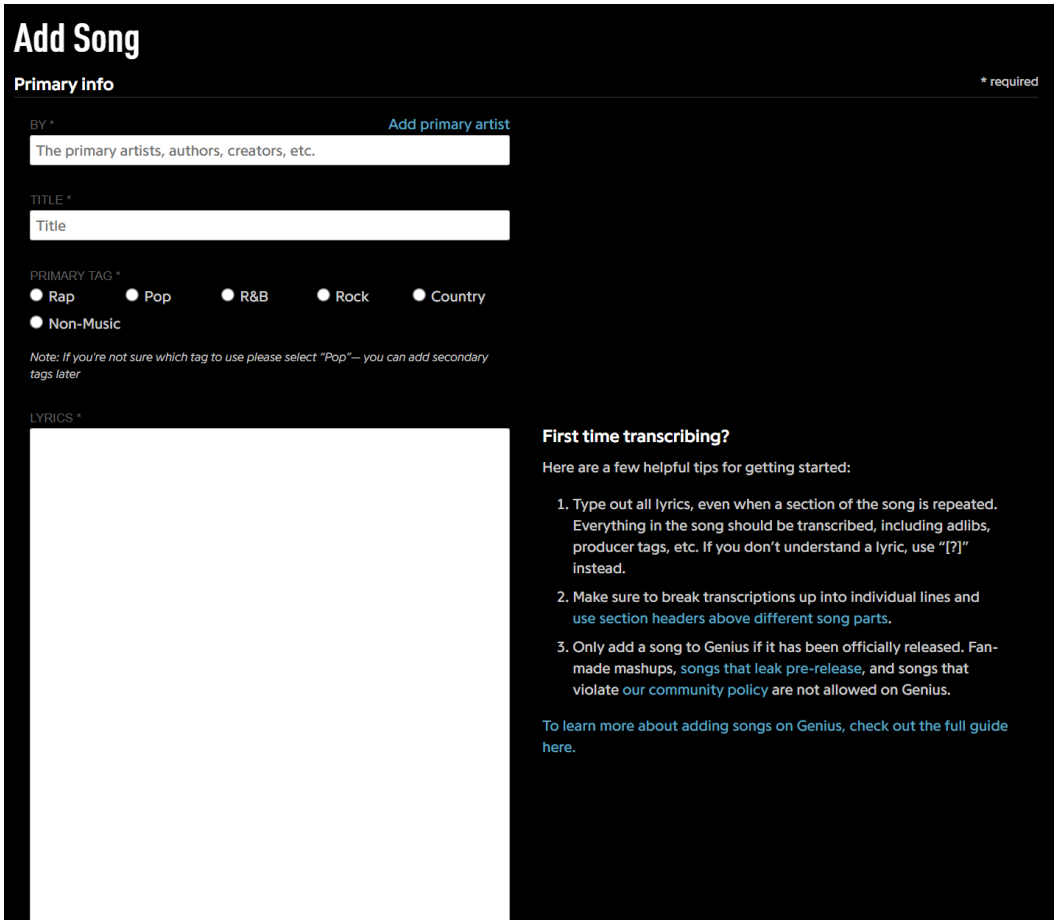


Рисунок 1.2 – Форма створення тексту/перекладу пісні

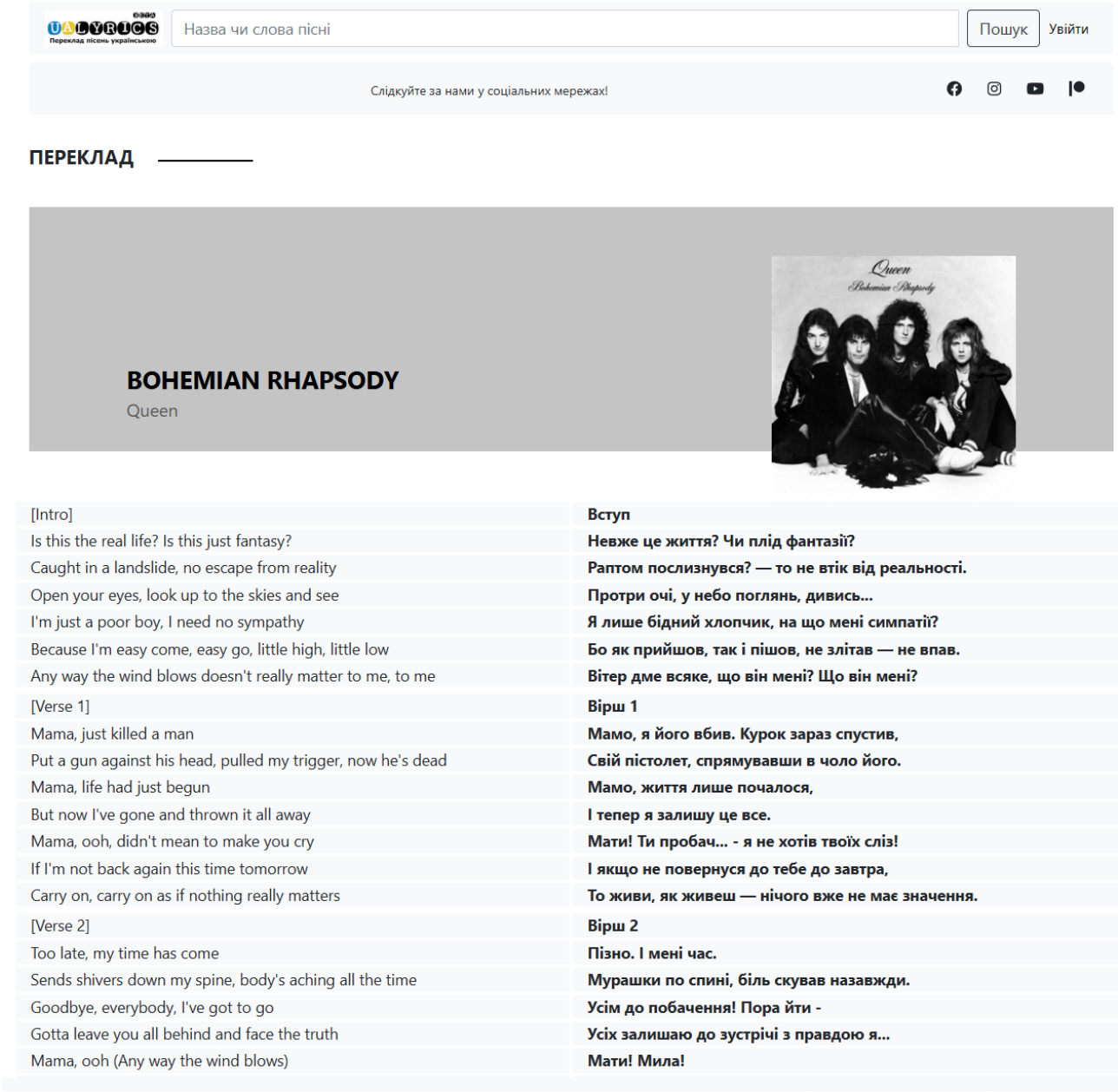
1.4.2 Ualyrics

Ще однією розробкою є ualyrics.com. Українська платформа, яка позиціонує себе як найамбітніший проект з перекладу текстів пісень українською. Судячи із заголовків запитів, які надсилаються сервером у відповідь на запити клієнтської частини сайту, сайт розроблений з використанням GraphQL стандарту, та використовує відповідний сервіс AWS AppSync для роботи з стандартом від провайдера хмарних послуг Amazon.

GraphQL – це мова запитів й маніпулювання даними. Розроблена у 2012 році тогочасною компанією Facebook, вона протягом наступних років швидко інтегрувалася у наявні системи, та стала основою для нових. Використовуючи підхід, наданий GraphQL можна гнучко вибирати структуру й набір даних, які отримуються з серверу, що є корисним для великих проекті, але не застосовується для простих API [10].

А, аналізуючи структуру HTML-сторінок які вертаються, можна зрозуміти що фронтенд написаний на React. Це дає нам знати, що технологічна складова сайту сучасна, з використанням поточних тенденцій у веброзробці.

Візуальна складова перекладів відрізняється суттєво від попередника. Мінімально подана інформація про пісню, весь акцент спрямовано на самих текстах оригіналу та перекладу (рис. 1.3). Сам дизайн сторінки виконаний доволі просто, що може здатися трохи застарілим. А у цілому, сайт працює швидко, без перешкоди. Також велика база готових перекладів покриває більшість найвідоміших пісень.



[Intro]	**Вступ**
Is this the real life? Is this just fantasy?	**Невже це життя? Чи плід фантазії?**
Caught in a landslide, no escape from reality	**Раптом послизнувся? — то не втік від реальності.**
Open your eyes, look up to the skies and see	**Протри очі, у небо поглянь, дивись...**
I'm just a poor boy, I need no sympathy	**Я лише бідний хлопчик, на що мені симпатії?**
Because I'm easy come, easy go, little high, little low	**Бо як прийшов, так і пішов, не злітав — не впав.**
Any way the wind blows doesn't really matter to me, to me	**Вітер дме всяке, що він мені? Що він мені?**
[Verse 1]	**Вірш 1**
Mama, just killed a man	**Мамо, я його вбив. Курок зараз спустив,**
Put a gun against his head, pulled my trigger, now he's dead	**Свій пістолет, спрямувавши в чоло його.**
Mama, life had just begun	**Мамо, життя лише почалося,**
But now I've gone and thrown it all away	**І тепер я залишу це все.**
Mama, ooh, didn't mean to make you cry	**Мати! Ти пробач... - я не хотів твоїх сліз!**
If I'm not back again this time tomorrow	**І якщо не повернуся до тебе до завтра,**
Carry on, carry on as if nothing really matters	**То живи, як живеш — нічого вже не має значення.**
[Verse 2]	**Вірш 2**
Too late, my time has come	**Пізно. І мені час.**
Sends shivers down my spine, body's aching all the time	**Мурашки по спині, біль скував назавжди.**
Goodbye, everybody, I've got to go	**Усім до побачення! Пора йти -**
Gotta leave you all behind and face the truth	**Усіх залишаю до зустрічі з правдою я...**
Mama, ooh (Any way the wind blows)	**Мати! Мила!**

Рисунок 1.3 – Сторінка перекладу пісні на платформі ualyrics.com

Порівняємо також сторінки створення перекладів. На цьому сайті також, як й на попередньому, створити переклад може будь-який авторизований користувач. При огляді досвіду створення перекладу, було помічено два значних недоліки. Немає можливості добавити переклад пісні виконавця, якого не має у системі. Другий – натискаючи на пісні в системі, які не перекладені, відбувається пошук оригінального тексту, проте не успішно. Система постійно видає помилку. Також, було визначено що ця система також використовує Genius API як провайдер даних пісень та артистів.

Сторінка перекладу, дозволяє працювати у режимі рядок за рядком, де видно оригінальний текст (рис. 1.4). Дизайн даної форми для введення тексту не найкращий, проте функціональний. Перекладачу не представлений жоден додатковий інструмент для допомоги у перекладі.

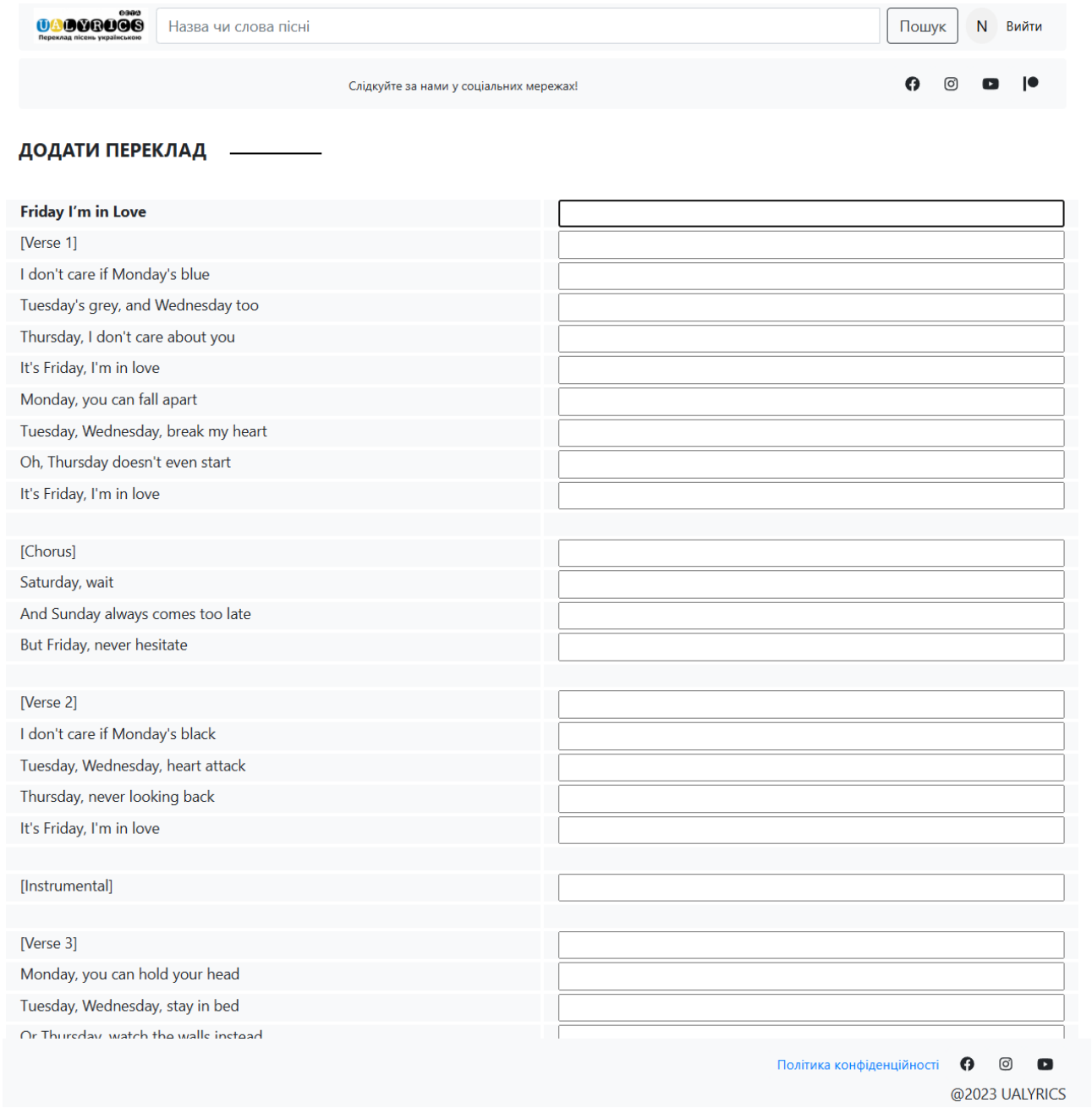


Рисунок 1.4 – Сторінка створення перекладу

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА СЕРВІСУ ДЛЯ ПЕРЕКЛАДУ ТЕКСТІВ

2.1. Постановка задачі, призначення та вимоги до програмного засобу

Метою дослідження є розробка вебсайту для автоматизованого перекладу текстів пісень із використанням технологій штучного інтелекту. Для реалізації цього завдання, потрібно не тільки забезпечити інтерфейс, але й організувати доступ до матеріалів, які стануть основою перекладів та забезпечити безперебійний, швидкий та зручний доступ до інтерфейсів технології штучного інтелекту. Ключовими елементами такого вебсайту є:

- велика база музичної інформації, що включає у себе усі потрібні зв’язки;
- можливість створення перекладів як вручну, так й використовуючи допомогу інструментів штучного інтелекту;
- реєстрація користувачів для розширеного доступу взаємодії з платформою;
- інтерактивний та зрозумілий інтерфейс.

Інтерфейс платформи розроблений таким чином, щоб убезпечити інтуїтивність користування та зручність, дотримуючись сучасних стандартів дизайну. Він має просту та зручну структуру, багато спільних елементів, що спрощує взаємодію та розуміння вебсайту. Основні елементи інтерфейсу включають:

1. Головна сторінка, яка містить блоки з різними типами ресурсів системи, тобто піснями, виконавцями та альбомами. Зверху є посилання на автентифікацію користувача, реєстрацію, створення нових перекладів.
2. Сторінка перекладу. Зображено всю інформацію про пісню, а саме назва, виконавець, альбом, дата випуску. Також у двох колонках, паралельно зображені оригінальний текст пісні та переклад.
3. Сторінки виконавця, альбому, де відображена інформація, відповідна для них, та пісні, які прямо відносяться до вибраного виконавця чи

альбому.

4. Сторінка створення перекладу. Зображено основну інформацію, та поля для введення перекладу кожної стрічки окремо. Також є бічні кнопки для застосування штучного інтелекту на вибраний рядок. Це все супроводжується кнопкою для подальшого створення перекладу у системі.
5. Форма реєстрації або автентифікації, де користувач вводить дані.

Основні функціональні можливості:

- каталог пісень, їхніх перекладів, виконавців та альбомів;
- пошук пісень, виконавців за назвою;
- використання технологій штучного інтелекту для допомоги у перекладі;
- наявність користувача (реєстрація та автентифікація);
- створення перекладу (введення текстів перекладу, збереження у системі).

Режими роботи:

1. Користувач, який може переглядати пісні, переклади.
2. Перекладач, який імпортує нові пісні у систему та перекладає їхній текст.

Основні програмні модулі:

1. Модуль музики, що відповідає за взаємодію з основними компонентами платформи, такі як виконавець чи пісня, за допомогою моделей Django, які роблять запити до бази даних.
2. Модуль перекладів для їхнього створення та показу користувачам.
3. Модуль користувача що відповідає за його реєстрацію, автентифікацію.
4. Модуль штучного інтелекту для забезпечення комунікації між внутрішньою системою та системою провайдера технологій штучного інтелекту.
5. Модуль фронтенду, який відповідає за відображення усіх даних, переданих з серверної частини, та забезпечує зворотнє передавання

цих даних до серверу. Написаний з використанням фреймворку React та використовуючи мови JSX та CSS.

2.2. Вибір моделі розробки програмного засобу

Для реалізації даного програмного засобу було обрано ітеративну модель розробки. Ітеративна модель передбачає поетапне створення системи шляхом розробки окремих функціональних частин (ітерацій), кожна з яких проходить повний цикл життєвого циклу: аналіз вимог, проєктування, реалізацію, тестування та вдосконалення [11].

Ця модель була враховуючи такі переваги:

- гнучкість у розробці – дає змогу вносити зміни до функціоналу на ранніх етапах без значного ризику для всієї системи;

- можливість раннього отримання результату – вже після першої ітерації можна протестувати базову версію продукту;

- зручність у тестуванні та виправленні помилок – тестування кожної ітерації окремо дозволяє швидко виявляти та усувати недоліки;

- поступове розширення функціоналу – основні можливості реалізуються на ранніх етапах, а менш критичні – додаються у наступних ітераціях;

- покращення тестування – можна враховувати працездатність системи після кожної ітерації.

В умовах змінюваних вимог до користувацького інтерфейсу та розвитку інструментів машинного перекладу, ітеративна модель є найбільш доцільною. Вона дозволяє адаптувати архітектуру та функціонал системи відповідно до поточних потреб, забезпечуючи швидке впровадження змін.

2.3. Загальний опис проєкту

Проєкт полягає у розробці вебзастосунку для перегляду та перекладу текстів пісень з використанням сучасних технологій штучного інтелекту.

Основна мета вебдодатку – забезпечити зручний інтерфейс для користувачів, які бажають читати тексти пісень в оригіналі та в перекладі, а також розширити доступ до музичного контенту для неангломовної аудиторії.

Застосунок реалізовано за клієнт-серверною архітектурою. У якості бекенду використано мову програмування Python з фреймворком Django та Django REST Framework, що забезпечує реалізацію REST API для обміну даними між клієнтом та сервером [12, 13]. Для клієнтської сторони обрано JavaScript бібліотеку React, яка відповідає за динамічне відображення контенту та взаємодію з користувачем.

Додатково реалізовано імпорт пісень через API платформи Genius, що дає змогу наповнювати базу даних актуальним контентом. На рисунку 2.1 схематично зображено використання цього API у системі та алгоритм взаємодії з ним.

Рисунок 2.1 – Алгоритм взаємодії з Genius API

Застосунок розгорнутий у середовищі Docker, що забезпечує портативність, масштабованість та зручність у супроводі [14]. У перспективі система може бути розширена функціоналом особистих кабінетів, збереженням улюблених пісень, редагуванням перекладів вручну та підтримкою декількох мов перекладу.

База даних проєкту реалізована на основі SQLite3, що є базою на основі файлу, та дозволяє оперувати контекстом за допомогою спеціального, спрощеного діалекту мови SQL [15]. Це робить SQLite3 зручною базою даних для етапу розробки та тестування. Основні модулі системи включають роботу з артистами, альбомами, піснями, текстами оригіналів та їх перекладами. Переклади надаються автоматично за допомогою інтеграції з API OpenAI, що дозволяє отримувати якісні машинні переклади на українську мову.

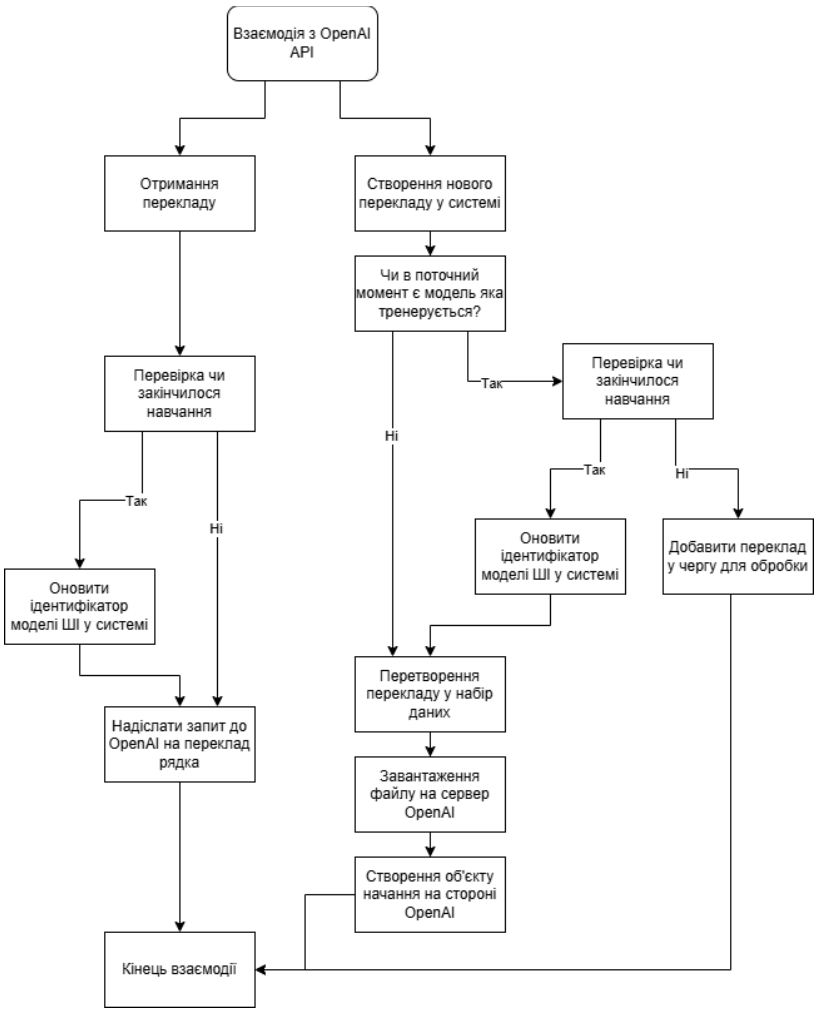


Рисунок 2.2 – Алгоритм взаємодії з OpenAI API

OpenAI надає доступ до моделей штучного інтелекту, а саме мовну модель GPT й засоби для тренерування (fine-tuning). Таким чином, застосунок використовує усі ці можливості для створення додатку, який дозволяє перекласти частину тексту автоматично, та в подальшому дотренерувати модель, використовуючи користувацькі переклади як набір даних. На рисунку 2.2 зображено принцип взаємодії та логіку, при використанні OpenAI API у системі.

2.4. Обґрунтування вибору інструментальних засобів розробки

Передбачається створення повноцінного вебзастосунку з розділенням на клієнтську і серверну частини. Для реалізації такого функціонального продукту було необхідно підібрати такий набір інструментальних засобів, який не лише відповідає технічним вимогам, а й сприяє швидкій та масштабованій розробці.

Серверною мовою програмування було обрано Python – одну з найпопулярніших у сферах веброзробки, наукових досліджень, автоматизації та машинного навчання. Серед інших мов вона відрізняється лаконічним синтаксисом, легкою читабельністю для розробників, широким вибором готових рішень та простотою розробки.

Як основний фреймворк для реалізації серверної логіки обрано Django REST Framework (DRF). Цей фреймворк є розширенням популярного Django і призначений для швидкої розробки на основі архітектури REST. DRF забезпечує зручну структуру для організації коду, підтримує авторизацію, серіалізацію, пагінацію, фільтрацію та дозволяє модифікувати кожну частину функціоналу на власну манеру – таким чином можлива інтеграція з іншими системами.

Серед альтернатив для розробки серверної частини є й інші фреймворки на основі Python, такі як FastAPI, Flask або Tornado. FastAPI, як приклад, є

сучаснішим й швидшим ніж DRF за рахунок більшої асинхронної бази та інших архітектурних рішень, що є ідеальним для написання мікросервісних застосунків. Проте, недостатність вбудованих рішень є значним недоліком при написанні великих рішень, які повинні забезпечувати безперебійну роботу багатьох модулів [16]. Також, наявність вбудованої ORM-системи для роботи з базою даних є суттєвою перевагою, дозволяючи не марнувати час розробки на інтеграцію інших рішень.

Клієнтська частина вебдодатку реалізована з використанням JavaScript, яка є безумовним стандартом сучасної веброзробки. З допомогою бібліотеки React розширюються основні можливості, що дозволяє створювати динамічні, інтерактивні інтерфейси на основі компонентної архітектури, яка полегшує повторне використання коду, тестування та масштабування проєкту. React є лідером серед бібліотек для frontend-розробки, і його екосистема включає такі інструменти, як React Router, Redux, Next.js, що дозволяє за потреби гнучко розширити функціонал.

Були розглянуті доступні альтернативи – Angular та Vue.js. Перший, маючи більше можливостей, є складнішим в освоєнні через складну архітектура. Vue.js є легким для засвоєння, але поступається React за популярністю та масштабом застосування у сфері вебдодатків [17]. Беручи до уваги зрозумілість документації, наявність готових варіантів, вибір зупинився на React.

Для зберігання даних про пісні, виконавців, альбоми та переклади використовується вбудована у Django реляційна база даних SQLite3. Її перевагами є відкритий код, мультиплатформеність та доступність. Легкість встановлення та використання є оптимальним рішенням для локальної розробки та прототипування. Водночас, завдяки гнучкості фреймворку для серверної сторони, не складе великих труднощів можливість переходу на іншу базу, таку як PostgreSQL, яка забезпечить вищу продуктивність і гнучкість при роботі з великим обсягом інформації.

Розробка відбувається у середовищі програмування PyCharm від компанії

JetBrains. На противагу популярному та безоплатному редактору коду Visual Studio Code (VS Code), PyCharm більш адаптований для розробки саме на мові Python, включаю в себе велику кількість інструментів для написання, тестування та оптимізації коду. Система доповнень, динамічна перевірка типів та правильності коду, вбудована інтеграція із інструментами веброзробки та відсутність необхідності налаштовувати це все вручну, стало вирішальним чинником при виборі середовища.

Окрім цього, для полегшення розгортання, тестування та ізоляції середовища розробки було обрано використання Docker. Docker – це програмне забезпечення, яке дозволяє на робочій машині створити окреме, ізольовна середовище. На відміну від віртуальних машин, Docker забезпечує більше налаштування, та малий у розмірах. Контейнери, які складаються з образів різних операційних систем, дозволяю використовувати найменші з них, наприклад, образ Ubuntu розміром всього лиш 68 Мегабайт. Docker дозволяє створювати контейнеризоване середовище, в якому весь вебзастосунок, включно з серверною частиною, клієнтським інтерфейсом та базою даних – запускається у незалежних контейнерах. Це забезпечує відтворюваність результатів, незалежність від операційної системи та спрощує процес розгортання на хмарних сервісах. Також, допомагає командам розробників запускати один й той самий проект у однакових умовах. Завдяки Docker Compose [18] можна легко керувати кількома сервісами одночасно, що пришвидшує процес розробки, спрощує конфігурацію залежностей та дозволяє легко масштабувати проєкт у майбутньому.

2.5. Особливості програмної реалізації програмного засобу

2.5.1 Серверна частина

Створення проекту розпочинається стандартною командою бібліотеки Django “django-admin startproject lyricsua .”. Таким способом генерується базова структура та модулі, необхідні для першого запуску додатку.

Перед тим як взаємодіяти з даними, необхідно їх зберігати. Для цього й використовується база даних, але потрібно з нею зв’язатися. Django має вбудовану ORM-систему, що дозволяє перетворювати класи Python у відповідні таблиці баз даних.

Ми оголосимо мінімальний набір моделей, необхідних для функціонування. Ними будуть моделі музичних об’єктів, додаткових метаданих до них, модель перекладу. На прикладі об’єкта пісні (див. Рисунок 2.3), представлені основні атрибути кожного об’єкту. Це як назва й картинка, так і зв’язок з пов’язаними об’єктами, та ідентифікатор зі сторони стороннього ресурсу. Також присутні поля, які вказують чи створення вручну. Таким чином, є простір для майбутніх оновлень, у вигляді підтримки власноруч створених об’єктів, а не тільки імпортованих з Genius. Схожу структуру моделі наслідують об’єкти виконавця та альбому.

```python
from django.db import models

class TimestampMixin(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    class Meta:
        abstract = True

class Song(TimestampMixin):
    genius_id = models.IntegerField(unique=True, null=True, blank=True)
    artist = models.ForeignKey(Artist, on_delete=models.CASCADE, related_name='songs')
    album = models.ForeignKey(Album, on_delete=models.CASCADE, blank=True,
                              related_name='songs', null=True)
    name = models.CharField(max_length=255)
    image = models.ImageField(upload_to='song_image/', blank=True)
    main_color = models.CharField(max_length=6, default='')
    manually_created = models.BooleanField(default=False)
    fully_imported = models.BooleanField(default=False)


class SongDetails(TimestampMixin):
    song = models.OneToOneField(Song, on_delete=models.CASCADE, related_name='details')
    description = models.TextField()
    release_date = models.DateTimeField(null=True)
    lyrics = models.TextField()
    youtube_link = models.URLField(max_length=255, blank=True)
    spotify_link = models.URLField(max_length=255, blank=True)
    soundcloud_link = models.URLField(max_length=255, blank=True)
    applemusic_link = models.URLField(max_length=255, blank=True)
```

Рисунок 2.3 – оголошення моделі пісні та її деталей

Django REST Framework пропонує фактично два види базових класів для створення виглядів, точок входу для запитів клієнтської частини. Розглянемо клас GenericViewSet, який покриває базові вимоги більшості ситуацій. За допомогою наслідування можна швидко отримати клас з вбудованими функціями. Використовуючи ще кілька класів, які називаються “mixins”, добавляється конкретна логіка, отримання списку об’єктів, одного екземпляру, чи то POST-запит на створення іншого. На рисунку 2.4 зображено такий клас, на прикладі вигляду пісні. Вбудовані атрибути дозволяють вказати модель, на основі якої робитимуться операції, серіалізатор, який перетворюватиме вхідні та вихідні дані, налаштування безпеки, обмежуючи доступ до певних ресурсів лише певному колу користувачів. Великою перевагою є можливість наслідування будь-якої функції, яка надається абстрактним класом – можна запросто під свої потреби оновити метод отримання даних з бази даних, застосувавши додаткові фільтри чи сортування.

```
class SongViewSet(GenericViewSet,
                  mixins.ListModelMixin,
                  mixins.RetrieveModelMixin,
                  ):
    queryset = Song.objects.all()
    serializer_class = SongSerializer
    permission_classes = [AllowAny]
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['artist', 'album']

    def get_queryset(self):
        if self.request.GET.get('new') == 'true':
            return Song.objects.filter().order_by("-created_at")[:10]
        return self.queryset
```

Рисунок 2.4 – Фрагмент коду з описом класу SongViewSet

Перед тим як система підтримуватиме застосування інструментів штучного інтелекту для допомоги у перекладі, потрібно реалізувати створення цих перекладів вручну. Для цього у нагоді стає заготовлений бібліотекою Django REST Framework клас, який ми наслідуємо, щоб успадкувати спільну логіку.

Саме створення не є складним: ми передаватимемо ідентифікатор пісні, до якої створюватимемо переклад, та переклад. Користувача ми дістанемо через об’єкт запиту на сервер. Цей запит потребує обов’язкової авторизації. У Django існують вбудовані Middleware (проміжне програмне забезпечення) класи, для обробки всіх запитів. Один з них, використовується для авторизації, й присвоює запиту об’єкт авторизованого користувача [19].

Надалі можна приступити до інтеграції штучного інтелекту у систему. Зручно, що для взаємодії з API OpenAI є готове рішення, однойменна бібліотека під Python. Після її встановлення та отримання ключів автентифікації з сторінки розробника на сайті OpenAI можна розпочинати інтеграцію.

З усіх ресурсів ми використовуватимемо насамперед “completions”. Він відповідальний за передачу вхідних даних у мовну модель та отримання обробленої відповіді. У запит також потрібно передавати назву моделі, до якої ми звертаємося. Було вибрано модель “gpt-4o-mini”, завдяки її економічним показникам, балансу між правильністю результатів та ціни [20].

Використовуємо заготовлену інструкцію на рівні системи для моделі, передаємо необхідний контекст та отримуємо в результаті готовий запит (рис. 2.5).

```
def translate_line(self, line, prev_line, next_line):
    prompt = ""

    if prev_line:
        prompt += f"Previous line: {prev_line}\n"
    prompt += f"Line to translate: {line}\n"
    if next_line:
        prompt += f"Next line: {next_line}\n"
    completion = self.client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": "You are a translator bot which translates songs lyrics to Ukrainian, "
                                          "and responds with only the translated lyrics. Take into account
             previous and next line for context"},
            {"role": "user", "content": line}
        ]
    )
    return completion.choices[0].message.content
```

Рисунок 2.5 – Фрагмент коду виклику доповнення тексту

Одного лише запиту недостатньо. Потрібно, щоб система підтримувала навчання моделі на перекладах, які створюють користувачі, таким чином покращуючи майбутні доповнення. Для цього ми створюємо окрему модель у базі даних, яка репрезентуватиме Fine-tune модель (рис. 2.6).

```python
class FineTunedModel(models.Model):
    fine_tune_id = models.CharField(max_length=255)
    job_id = models.CharField(max_length=255, null=True, blank=True)
    training_file_id = models.CharField(max_length=255, null=True, blank=True)

    pending_translations_ids = models.JSONField(default=list, blank=True)

    date_created = models.DateTimeField(auto_now_add=True)
    date_updated = models.DateTimeField(auto_now=True)

    last_check = models.DateTimeField(auto_now_add=True)

    def save(
        self, force_insert=False, force_update=False, using=None,
update_fields=None
        if self.pk:
            return super().save(force_insert, force_update, using, update_fields)

        if FineTunedModel.objects.all().count() > 0:
            raise Exception('Only one fine-tuned model is allowed')

        super().save(force_insert, force_update, using, update_fields)
```

Рисунок 2.6 – Клас об’єкту моделі тренерування

Атрибути “fine_tune_id”, “job_id” та “training_file_id” відповідатимуть за ідентифікацію ресурсів зі сторони OpenAI. За допомогою них ми матимемо поточну, крайню натреновану, модель та додаткові дані, потрібні для тренерування нової. А успадкований метод збереження моделі забезпечить існування в базі даних лише одного екземпляру.

Перед тим, як можна буде розпочати тренерування моделі, потрібно підготовити набір даних, які будуть використані для навчання. В своїй основі –

це просто набір повідомлень вказівок, якою повинна бути відповідь моделі при певному запиті. Вони складаються з оригінальної стрічки тексту пісні та відповідник перекладу. Наступним кроком є записування даних у файл формату JSON Lines [21] – формат, у якому кожен рядок є окремим JSON об’єктом, що дозволяє обробляти дані рядок за рядком. Такий формат часто використовується при обробці великої кількості інформації. Цей процес зображено на рисунку 2.7.

```
def convert_translation_as_training_data(self, original_lyrics, translated_lyrics):
    original_list = original_lyrics.split('\n')
    translated_list = translated_lyrics.split('\n')
    if len(original_list) != len(translated_list):
        raise Exception('Original and translated lyrics have different lines
count')
    training_data = []
    for i in range(len(original_list)):
        if original_list[i] == '' or translated_list[i] == '':
            continue
        training_data.append(
            {"messages": [
                {"role": "system", "content": "You are a translator bot which"
                 "translates songs lyrics to Ukrainian."},
                {"role": "user", "content": original_list[i]},
                {"role": "assistant", "content": translated_list[i]}
            ]}
        )

    return training_data

def write_training_data(self, training_data):
    with open('training_data_temp.jsonl', 'w', encoding='UTF-8') as file:
        for item in training_data:
            file.write("%s\n" % json.dumps(item, ensure_ascii=False)
```

Рисунок 2.7 – Підготовка даних у формат для навчання

З наявним файлом даних, можна розпочати тренерування моделі. Після успішного запиту до API, створюється новий процес донавчання. Важливим моментом є передавання крайньої натренованої моделі, відповідно, кожне нове донавчання буде на основі нашої моделі (рис. 2.8).

Надалі для взаємодії з об’єктом поточного процесу донавчання потрібно

кілька додаткових функцій. Одна – для отримання поточного статусу, а інша – для позначення процесу як завершеного.

```python
def upload_file(self):
    response = self.client.files.create(
        file=open('training_data_temp.jsonl', 'rb'),
        purpose='fine-tune',
    )

    self.fine_tuned_model.training_file_id = response.id
    LOGGER.info(f"Upload file response: {response}")
    return response

def create_fine_tune(self):
    response = self.client.fine_tuning.jobs.create(
        model=self.fine_tuned_model.fine_tune_id,
        training_file=self.fine_tuned_model.training_file_id,
    )
    self.fine_tuned_model.job_id = response.id
    self.fine_tuned_model.save()
    LOGGER.info(f"Create fine tune response: {response}")

    return response
```

Рисунок 2.8 – Запит на створення об’єкту навчання моделі за допомогою OpenAI API

Постає питання: як реалізувати постійну перевірку, чи процес донавчання завершений? Реалізація OpenAI API не має в наявності зворотних викликів, які б сповістили про завершення. Тому перевірку потрібно виконувати зі сторони серверу. Беручи до уваги невелику складність проєкту, немає сенсу інтегровувати автоматичні скрипти, які б у заданій періодичності перевіряли б статус. Натомість, ми використаємо функцію “precheck_fine_tune_status” (рис. 2.9), яка викликатиметься при кожному запиті на перекладання рядка. Таким чином, наша система не буде обізнана у закінченні процесу до моменту, як користувач не викличе функціонал автоматизованого перекладу. Якщо після

останньої перевірки пройшло більше п’яти хвилин та існує модель в стані тренування, її статус оновиться. Відповідно, користувач отримає переклад від нової, дотренерованої моделі.

```
def precheck_fine_tune_status(self):
    if self.fine_tuned_model.last_check + timedelta(minutes=5)
    < timezone.now():
        self.fine_tuned_model.last_check = timezone.now()
        self.fine_tuned_model.save()
        self.check_fine_tune_status()
```

Рисунок 2.9 – Періодична перевірка на завершення тренерування

На рисунку 2.10 зображена основна функція “process_new_fine_tune”, яка викликається при кожному створенні перекладу. Першочергово відбувається перевірка, чи не навчається на даний момент інша модель. Адже немає сенсу створювати новий процес навчання, бо отримаємо дві різні моделі, з різними наборами тренувальних даних. Натомість, зберігаємо у базі ідентифікатор перекладу, який у майбутньому потрібно буде додати у дані для тренерування. Якщо ж немає поточного навчання моделі, дістаємо усі переклади з черги, агрегуємо усі навчальні дані у необхідний формат. Завантажуємо його на сервер OpenAI та запускаємо новий процес навчання. Ця функція підсумовує логіку та життєвий цикл створення нового перекладу у системі – від його появи до оновлення моделі штучного інтелекту.

```
def process_new_fine_tune(self, song_translation):
    if self.fine_tuned_model.job_id is not None:
        available = self.check_fine_tune_status()
        if not available:
            self.fine_tuned_model.pending_translations_ids.append(
              song_translation.id
            )
            self.fine_tuned_model.save()
            return

    pending_translations_ids = self.fine_tuned_model.pending_translations_ids
+                              [song_translation.id]
    training_data = []
    for translation_id in pending_translations_ids:
        translation = SongTranslation.objects.get(id=translation_id)
        training_data += self.convert_translation_as_training_data(
            translation.song.details.lyrics, translation.lyrics
        )

    self.write_training_data(training_data)
    self.upload_file()
    self.create_fine_tune()

    self.fine_tuned_model.pending_translations_ids = []
    self.fine_tuned_model.save()
```

Рисунок 2.10 – Повний процес від отримання перекладу до створення об’єкту тренерування

2.5.2 Клієнтська частина

Клієнтська сторона є усім, що доступно користувачу для перегляду та взаємодії. Якщо серверна сторона цікава лише розробнику й не є візуальним представленням, то клієнтська частина є лицем проєкту. Саме тому розробка зручного, візуально приємного інтерфейсу є ключовим завданням при розробці. Необхідно звернути увагу також на взаємодію різних компонентів, обробку помилок, станів завантаження.

Клієнтська частина реалізована за допомогою фреймворку React. Першою командою, яка застосовується є “npx create-react-app”. Вона ініціалізує базову структуру, файли та віртуальне середовище. Після цього переходиться до

розробки компонентів та сторінок.

Так як React робить роботу з компонентами простою й зручною, їхнє використання не лишається осторонь. Вони дуже схожі на стандартні функції, але є ізольованими та повертають HTML-код [22]. Це дозволяє уникнути повторення часто вживаних фрагментів сайту на сторінці. До прикладу, заголовок сайту, його навігація та колонтитул. Часто вживані елементи, наприклад піктограма пісні й її назви, контейнери та структурні шаблони – всі вони реалізовані як незалежні компоненти.

Найважливіша сторінка – це текст пісні з перекладом. Оскільки вона є результатом перекладу і доступна всім, її дизайн має бути зручним та сучасним. Сторінка починатиметься із зображення обкладинки пісні та відомостей про неї. Далі представлено текст оригіналу у лівій колонці, та перекладу – у правій. Все це супроводжується зручним дизайном, кольоровим виділенням назв частин пісні (вступ, куплет, приспів). На рисунку 2.11 зображено сторінку перегляду перекладу пісні.

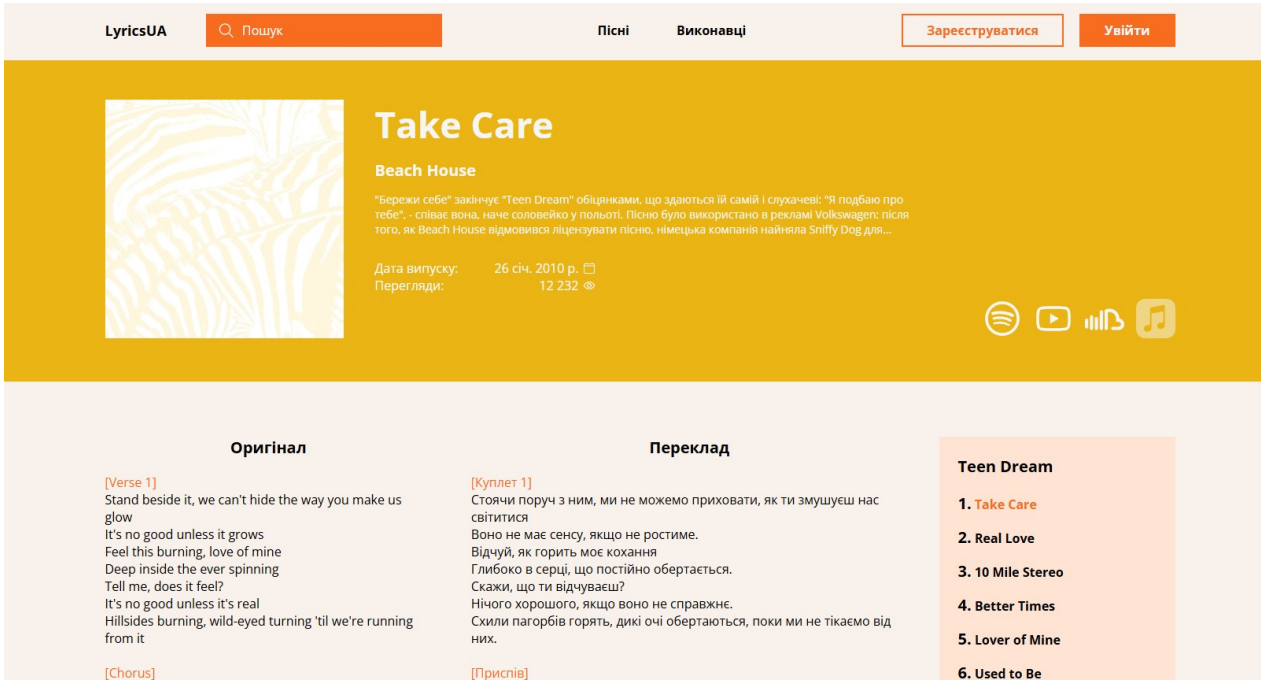


Рисунок 2.11 – Сторінка перегляду перекладу

Зі сторони клієнта необхідна реалізація відповідних інтерфейсів, які відповідатимуть за взаємодію користувача з цим функціоналом. Насамперед це сторінки авторизації та реєстрації користувача.

Щоби мати змогу оперувати над користувачем та мати доступ до об’єкту авторизованого користувача по всім компонентам React-додатку, використовують “useContext” [23] з стандартної бібліотеки. Таким чином, фактично будь-який компонент може викликати функцію, отримати доступ до користувача та JWT-токену. На рисунку 2.12 зображена реалізація цього контексту.

```
import { createContext, useState, useContext } from
'react';
const UserContext = createContext({});
const useUserContext = () => useContext(UserContext);

const UserProvider = ({ children }) => {
    const [user, setUser] = useState(null);
    const [token, setToken] = useState(null);

    const value = {
        user, setUser, token, setToken
    }

  return (
    <UserContext.Provider value={value}>
      {children}
    </UserContext.Provider>
  );
};
export { UserProvider, UserContext, useUserContext };
```

Рисунок 2.12 – Оголошення контексту UserContext

Тепер потрібно використати даний контекст для запису користувача у змінну. Для отримання з серверу користувача необхідно надіслати запит з дійсним JWT-токеном. JWT-токен (JSON веб-токен) це такий спосіб

представлення доступу, який формується та шифрується на стороні серверу [24]. Дані користувача, підпис шифрування та дата закінчення валідності токену зберігаються у вигляді компактного JSON-об’єкта. При наявності токена відбуватиметься запит на отримання даних про користувача. Інакше, всі пов’язані змінні будуть очищені (рис. 2.13).

```
useEffect(() => {
    (async () => {
        if (!userCtx.token) return

        const response = await fetch('http://localhost:8000/api/users/profile',
{           method: 'GET',
            headers: {
                'Authorization': `Bearer ${userCtx.token}`
            }
        })

        if (response.status === 200) {
            const data = await response.json()
            userCtx.setUser(data)
        } else {
            userCtx.setUser(null)
            userCtx.setToken(null)
            localStorage.removeItem('token')
        }
    })();
}, [userCtx.token])
```

Рисунок 2.13 – Функція отримання даних користувача

Останнім кроком є отримання токена автентифікації. Це відбувається стандартно, при введені користувачем імені користувача та паролю. Клієнт відправляє ці дані на сервер. У випадку успішної перевірки, сервер генерує JWT-токен та повертає його (рис. 2.14). Клієнт, отримавши відповідь, зберігатиме токен у локальному сховищі. Локальне сховище дозволяє сайтам й програмам зберігати пари ключів-значень прямо у веббраузері, довгостроково, так як після оновлення сторінки чи перезапуску застосунку, дані не зникають

[25]. Оскільки функція (хук) присвоєння користувача містить у собі масив залежностей з токеном, при його оновленні, виконається код.

```
const handleSubmit = async (e) => {
    e.preventDefault()
    const res = await fetch(
      'http://localhost:8000/api/users/login/',
      {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            username,
            password
        })
    }).catch(() => {})
    const data = await res.json()
    if (data.error) {
        setError(data.error)
    } else {
        userCtx.setToken(data.token)
        localStorage.setItem('token',
data}token)
```

Рисунок 2.14 – функція отримання даних користувача

Це є основою для мінімальної працездатності аутентифікації користувача у системі. Надалі можна перейти до функціоналу, що потребує авторизованого користувача. До цього належить імпорт даних у систему, створення перекладу та використання підказок за допомогою штучного інтелекту.

Сторінка створення перекладу повинна починатися з вибору, яку пісню користувач буде перекладати. Задля забезпечення цілісності даних система не дозволяє створювати об’єкти вручну. Всі потрібні дані імпортуються за допомогою API з бази даних вебсайту Genius. Зі сторони користувача це виглядатиме як окрема сторінка з полем для пошуку пісень за назвою пісні чи виконавця (рис. 2.15). Пошук обмежується п’ятьма найбільш відповідними назвами. При при переході за гіперпосиланням результату пошуку система

перевіряє, чи є ця пісня у системі. Якщо ні – виконується запит для імпорту всіх необхідних даних у систему. Даний процес має простір для вдосконалення – додавання станів завантаження, щоби покращити користувацький досвід.

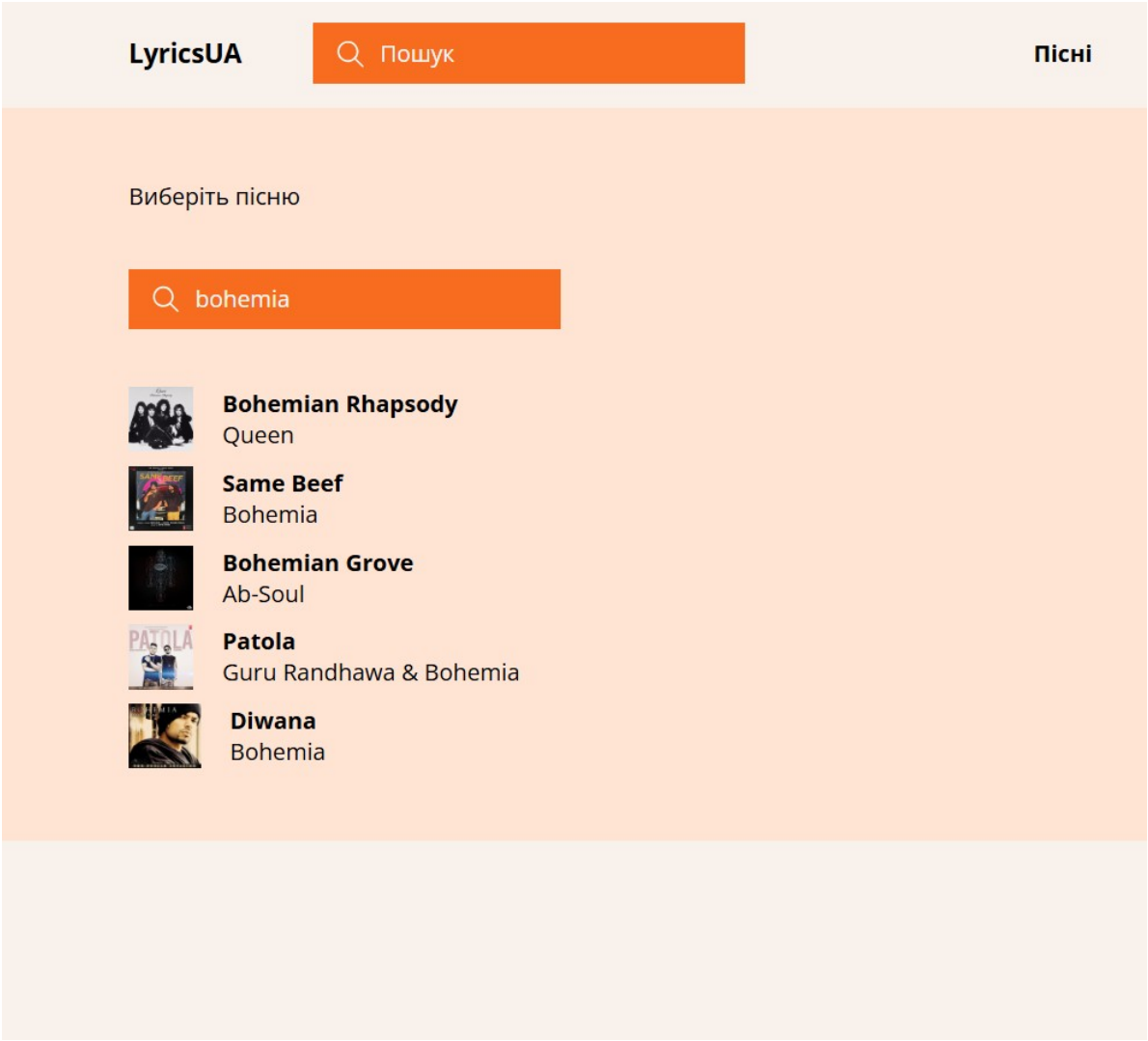


Рисунок 2.15 – Пошук пісні за допомогою Genius API

Після завантаження даних пісні у систему, користувача перенаправляє на сторінку створення перекладу. Інтерфейс забезпечує перегляд базової інформації про пісню, текст оригіналу на лівій половині екрану та поля для тексту перекладу на правій половині (рис. 2.16).

Рисунок 2.16 – Сторінка створення перекладу пісні

У процесі створення перекладу у користувача є можливість звернутися до інструментів штучного інтелекту для автоматизованого перекладу вибраного рядка з врахуванням попереднього й наступного, якщо такі наявні. Як й було реалізовано на серверній частині, результат автоматичного перекладу залежить від попередньо створених перекладів, які доповнюють контекст моделі. При наявності усіх заповнених полів, користувач може створити переклад. Функції виклику штучного інтелекту та створення перекладу зображені на рисунку 2.17.

```
const translateLineClick = async (index) => {
    index = parseInt(index)
    const line = lyricsAndTranslations[index].original
    const prev_line = lyricsAndTranslations[index - 1]?.original || ""
    const next_line = lyricsAndTranslations[index + 1]?.original || ""

    const response_data = await translateLine(line, prev_line,
next_line);const translated_line = response_data.translated_line;
    handleTranslationChange(index, translated_line)
};

const submitTranslation = async () => {
    for (let i = 0; i < lyricsAndTranslations.length; i++) {
        if (lyricsAndTranslations[i].translation === ""
            && lyricsAndTranslations[i].original[0] !== "") {
            setNotCompleteLyrics(true);
            return;
        }
    }
    setNotCompleteLyrics(false);
    const translation = Object.values(
      lyricsAndTranslations
    ).map(pair => pair.translation).join("\n");
    const response_data = await createSongTranslation(
      songData.id, translation, userCtx.user.id
    );
};
```

Рисунок 2.17 – Код функцій виклику штучного інтелекту й створення перекладу

З усього вищенаведеного складається повна структура програмної розробки, взаємодія її компонентів, серверної частини з клієнтською та процес взаємодії користувача з платформою.

2.6. Організація тестування та налагодження програмного засобу

Тестування проводиться з метою виявлення помилок та перевірки коректності функціонування. Даний засіб не передбачає особливих засобів для тестування, тому пропонується ручна перевірка основних функцій системи:

- сторінка пісні (коректне відображення зображень, текстів оригіналу та перекладу);

- імпортування даних пісень у систему;
- пошук пісень;
- створення перекладів;
- користувацька сторона (реєстрація, автентифікація);
- навчання моделі штучного інтелекту на створених перекладах;
- використання штучного інтелекту для перекладу.

2.7. Рекомендації по використанню та впровадженню програмного засобу

Вебзастосунок призначений для перегляду текстів пісень, їхніх перекладів та створення цих же перекладів у зручному та інтуїтивно зрозумілому інтерфейсі. Він може застосовуватись як особисто, для ознайомлення з перекладами пісень або для поширення цієї інформації мережею інтернет.

Мінімальні потреби для використання додатка залежать від сторони. Від клієнта вимагається наявність ОЗП не менше 2 ГБ для вебпереглядача, базовий процесор та мережеве підключення. Серверна частина може бути запущена на слабших обчислювальних потужностях, від 512 МБ ОЗП, але при збільшені наявної в базі інформації, потрібно збільшити наявні ресурси.

Вимогами до програмних засобів є Python 3.10 або новіша, Django 4.2.7 або новіша. React версії 19.

ВИСНОВКИ

Метою кваліфікаційної роботи було створення вебзастосунку для перегляду текстів пісень, їхніх перекладів, та створення цих перекладів за допомогою інструментів штучного інтелекту. Проведено аналіз аналогічних вебсайтів з створення перекладів, оцінено їхню зручність, функціональність та користувацький інтерфейс.

На початковому етапі було визначено інструменти для розробки, розглянуто основні тенденції веброзробки та вибрано модель, яка буде використовуватися для розробки.

У результаті розробки серверної частини було досліджено взаємодію з OpenAI API для інтеграції використання моделі штучного інтелекту у системі та його навчання. Були написані модулі, відповідальні за авторизацію користувачів, створення перекладів у системі, виклики до зовнішніх ресурсів та отримання даних з бази даних.

Розроблена клієнтська частина відповідає сучасним стандартам вебпрограмування, використовуючи актуальні інструменти та розробки. Користувача забезпечено зрозумілим та функціональним інтерфейсом, що дозволяє безперебійно використовувати увесь заявлений функціонал.

Вебдодаток може бути використаний будь-яким користувачем, який бажає дізнатися переклад тексту пісні іноземною мовою. Також окремі перекладачі можуть використовувати вебдодаток як інструмент для створення перекладів і їх поширення на широку аудиторію.

Подальший розвиток системи може включати в себе більше елементів соціальних мереж, дозволяючи переглядати профілі як звичайних користувачів, так і перекладачів, оглядати їхні переклади, рейтинги. Також можна розширити об’єкти перекладу, перекладаючи фактично усю інформацію, що можливо. Існує можливість інтеграції реклами українських виконавців.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 5 фреймворків та бібліотек для Front-end розробки - Digest.Pro. *Digest.Pro*. URL: https://www.digest.pro/news/5-freimvorkiv-ta-bibliotek-dlia-front-end-rozrobky/ (дата звернення: 10.03.2025).
2. maitray-gadhavi. 10 Best Backend Frameworks in 2025 | Radixweb. *Radixweb*. URL: https://radixweb.com/blog/best-backend-frameworks (дата звернення: 10.03.2025).
3. 13 Top Cloud Service Providers Globally (UPDATED 2024). *CloudZero*. URL: https://www.cloudzero.com/blog/cloud-service-providers/ (дата звернення: 10.03.2025).
4. Monolithic vs. Microservices Architecture - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/ (date of access: 10.03.2025).
5. Які існують типи нейронних мереж?. *DevZone*. URL: https://devzone.org.ua/qna/iaki-isnuiut-typy-neyronnykh-merez (дата звернення: 10.03.2025).
6. Chat GPT: What is it?. *University of Central Arkansas – UCA*. URL: https://uca.edu/cetal/chat-gpt/ (дата звернення: 10.03.2025).
7. Розуміння тонкого налаштування LLM: адаптація великих мовних моделей до ваших унікальних вимог. *Unite.AI*. URL: https://www.unite.ai/uk/understanding-llm-fine-tuning-tailoring-large-language-models-to-your-unique-requirements/ (дата звернення: 10.03.2025).
8. Учасники проектів Вікімедіа. Genius (вебсайт) – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Genius_(вебсайт) (дата звернення: 10.03.2025).
9. Ruby on Rails: Compress the complexity of modern web apps. Ruby on Rails: Compress the complexity of modern web apps. URL: https://rubyonrails.org/ (дата звернення: 24.05.2025).

10. GraphQL | A query language for your API. GraphQL | A query language for your API. URL: https://graphql.org/ (дата звернення: 24.05.2025).
11. QALight. Ітеративна модель (iterative model). QALight. URL: https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/ (дата звернення: 24.05.2025).
12. Django introduction - Learn web development | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django/Introduction (дата звернення: 24.05.2025).
13. Špela Giacomelli. Django REST Framework Basics. Test-Driven Development, Microservices, Web Development Courses | TestDriven.io. URL: https://testdriven.io/blog/drf-basics/ (дата звернення: 24.05.2025).
14. Що таке Docker: простими словами про контейнеризацію: Стаття з блогу IT-школи Hillel. Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel. URL: https://blog.ithillel.ua/articles/shcho-take-docker-prostimi-slovami-pro-konteynerizatsiyu (дата звернення: 24.05.2025).
15. Програмування баз даних на SQLite: перші кроки в освоєнні. FoxmindEd. URL: https://foxminded.ua/prohramuvannia-baz-danykh-na-sqlite/ (дата звернення: 25.05.2025).
16. Which Is the Best Python Web Framework: Django, Flask, or FastAPI? | The PyCharm Blog. The JetBrains Blog. URL: https://blog.jetbrains.com/pycharm/2025/02/django-flask-fastapi/ (date of access: 25.05.2025).
17. Elitex.systems. ELITEX: Leading Software Development Company. URL: https://elitex.systems/blog/most-popular-javascript-frameworks (дата звернення: 25.05.2025).
18. Docker Compose для DevOps – спрощення роботи з контейнерами. IT Education Center Blog. URL: https://itedu.center/ua/blog/review/what-is-docker-compose/ (дата звернення: 25.05.2025).

19. Middleware | Django documentation. Django Project. URL: https://docs.djangoproject.com/en/5.2/topics/http/middleware/ (дата звернення: 25.05.2025).
20. OpenAI API Pricing Calculator. AI add-ons for Excel Word, Sheets, Docs | GPT for Work. URL: https://gptforwork.com/tools/openai-chatgpt-api-pricing-calculator (дата звернення: 25.05.2025).
21. JSON Lines. JSON Lines. URL: https://jsonlines.org/ (дата звернення: 25.05.2025).
22. Компоненти і пропси – React. React – JavaScript-бібліотека для створення користувацьких інтерфейсів. URL: https://uk.legacy.reactjs.org/docs/components-and-props.html (дата звернення: 25.05.2025).
23. useContext – React. React. URL: https://react.dev/reference/react/useContext (дата звернення: 25.05.2025).
24. Як використовувати JSON Web Tokens (JWT) для автентифікації. DevZone. URL: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi (дата звернення: 25.05.2025).
25. Zakatsiura I. Як використовувати localStorage з хуками React для встановлення та отримання елементів. freeCodeCamp.org. URL: https://www.freecodecamp.org/ukrainian/news/yak-vykorystovuvaty-localstorage-z-khukamy-react-dlya-vstanovlennya-ta-otrymannya-elementiv/ (дата звернення: 25.05.2025).

ДОДАТКИ

Додаток А. ТЕХНІЧНЕ ЗАВДАННЯ

ВСТУП

Вебдодаток LyricsUA призначений для зручного перегляду текстів пісень та їх перекладів, їхнього створення за допомогою штучного інтелекту. Поєднуючи роботу серверної частини з клієнтською, застосунок забезпечує швидкий доступ до інформації та інтуїтивний інтерфейс.

ПІДСТАВИ ДЛЯ РОЗРОБКИ

Роботи з розробки вебдодатку для автоматизованого перекладу текстів здійснюються на підставі написання бакалаврської роботи.

ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмний продукт призначений для створення користувачами перекладів оригінальних текстів пісень власноруч чи за допомогою технологій штучного інтелекту, дозволяючи такі функції:

- імпорт пісень у систему;
- пошук по платформі;
- створення перекладів;
- перегляд інших перекладів.

ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вимоги до функціональних характеристик: програмний продукт повинен бути зрозумілим для використання та виконувати такі функції:

- відображати текст та переклад пісень;
- забезпечувати пошук по платформі;
- взаємодіяти з стороннім API задля створення об’єктів у базі даних;
- створення перекладів;
- використання штучного інтелекту задля допомоги при перекладі;
- навчання моделі штучного інтелекту для розширеного контексту перекладу;
- реєстрація та автентифікація користувача.

Вимоги до надійності: платформа повинна коректно працювати на серверах із Python 3.8 та вище. Клієнтська частина повинна працювати на усіх сучасних веббраузерах.

Умови експлуатації: програмна розробка повинна використовуватись на комп’ютерах та ноутбуках, з браузерами, що підтримують сучасні стандарти JavaScript та CSS. Серверна частина повинна підтримувати Python версії 3.8 та вище та забезпечувати стабільну роботу.

Вимоги до інформаційної і програмної сумісності: вхідними даними для програмного засобу повинні бути відповіді програмного інтерфейсу платформи Genius; дані користувачів та користувацький текст перекладів.

ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документів входять звіт до бакалаврської роботи та технічне завдання.

ТЕХНІЧНО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Після вирішення усіх недоліків програмного забезпечення, LyricsUA може бути опублікований у вільному доступі. В такому разі можна буде оцінити економічну ефективність та популярність серед користувачів шляхом аналізу відвідування та взаємодії з платформою. При цьому, потрібно буде вирішити питання авторських прав, адже не всі дані отримані з стороннього сервісу можна використовувати в комерційних цілях.

СТАДІЇ І ЕТАПИ РОЗРОБКИ

1. Попередня підготовка – вивчення і аналіз процесів, які потрібно реалізувати;
2. Визначення вимог до системи, що розробляється;
3. Проєктування, розробка програмного продукту на підставі складеного технічного завдання;
4. Впровадження системи, тестування в робочому режимі і здійснення необхідних доопрацювань, виявлених в процесі дослідної експлуатації.

ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Приймання результатів роботи здійснюється в процесі здачі бакалаврської роботи.

АНОТАЦІЯ

Андрущук П. Є. – Розробка сервісу з автоматизованим перекладом текстів із використанням технологій штучного інтелекту – Рукопис.

Кваліфікаційна робота за спеціальністю 122 Комп’ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2025 р.

Робота призначена розробці вебдодатку для перегляду та перекладу текстів пісень із використанням технологій штучного інтелекту як допоміжного засобу. Для цього, було розглянуто основні тенденції веброзробки, сучасні інструменти та проведено аналіз аналогічних програмних розробок, їхній функціонал та недоліки.

Серверна частина реалізована за допомогою вебфреймворку для Python під назвою Django, що дозволяє створювати масштабовані, гнучкі бізнес-рішення у короткі терміни, через наявність багатьох вбудованих рішень та функцій. Для створення інтерфейсу, клієнтської частини, використано JavaScript-бібліотеку React, яка затвердила себе як один з найкращих інструментів при розробці динамічних сторінок.

Додаток покладається на Genius API для поповнення внутрішньої бази даних піснями й текстами, а також OpenAI API для взаємодії з моделями штучного інтелекту, їхнього використання та навчання.

У результаті виконаної роботи було розроблено вебзастосунок, що дозволяє переглядати та створювати переклади текстів пісень, імпортувати оригінали текстів з стороннього сервісу та використовувати штучний інтелект як допоміжний засіб у перекладі. Важливою складовою була логіка навчання моделей ШІ на створених перекладах, збільшуючи його точність та розуміння контексту.

Ключові слова: вебдодаток, переклад текстів, штучний інтелект, Django, React, Docker, REST API, OpenAI API, Genius API.