

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Портал вивчення японської мови»

Виконав студент групи ПЗк-21-1
КЛИМЧУК Богдан Анатолійович

Керівник роботи:
КОНЦЕДАЙЛО Валерій Валерійович

Консультант:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КЛИМЧУК Богдан Анатолійович**

Керівник роботи: **КОНЦЕДАЙЛО Валерій Валерійович**

Тема роботи: **«Портал вивчення японської мови»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р. №74с**

Вихідні дані для роботи: **розробка порталу вивчення японської мови.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів,
що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В, Концедайло В.В.	27.02.2024р.	27.02.2024р.
2	Чижмотря О.В, Концедайло В.В.	27.02.2024р.	27.02.2024р.
3	Чижмотря О.В, Концедайло В.В.	27.02.2024р.	27.02.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота складається з програмної реалізації порталу вивчення японської мови, а також пояснювальної записки. Пояснювальна записка до дипломної роботи містить 95 сторінок, 30 ілюстрацій, 8 таблиць, 2 додатки та 30 літературних джерел.

Метою роботи є дослідження та розробка системи порталу, який надає користувачам певний набір інструментів, за допомогою яких вони можуть оптимізувати свій процес вивчення японської мови.

В роботі описано процес аналізу технічного завдання та вимог системи, оглянуто існуюче програмне забезпечення за тематикою проекту. Також описано процес розробки бази даних та реалізації системи освітнього порталу. Далі зазначено порядок встановлення та налаштування системи, приклади тестування та опис інтерфейсу додатку.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, НАВЧАННЯ, СЛОВНИК, ЯПОНСЬКА, FASTAPI, NEXTJS.

					ІПЗ.КР.Б – 121 – 24 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Климчук Б.А.			«Портал вивчення японської мови» Пояснювальна записка	Літ.	Арк.	Аркушів
Керівник		Концедайло В.В.					3	95
Рецензент		Концедайло А.М.						
Зав. каф.		Вакалюк Т.А.				Житомирська політехніка, група ІПЗк-21-1		

ABSTRACT

The final qualification work consists of a software implementation of a Japanese language learning portal, as well as an explanatory note. The explanatory note to the thesis contains 95 pages, 30 illustrations, 8 tables, 2 appendices, and 30 references.

The purpose of the work is to research and develop a portal system that provides users with a certain set of tools with which they can optimize their Japanese language learning process.

The paper describes the process of analyzing the terms of reference and requirements of the system, reviews existing software on the subject of the project. The process of developing the database and implementing the educational portal system is also described. Further, the procedure for installing and configuring the system, examples of testing and a description of the application interface are given.

KEYWORDS: WEB APPLICATION, LEARNING, DICTIONARY, JAPANESE, FASTAPI, NEXTJS.

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						4
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ПОРТАЛУ ВИВЧЕННЯ ЯПОНСЬКОЇ МОВИ.....	8
1.1 Технічне завдання на розробку системи.....	8
1.2 Аналіз існуючого програмного забезпечення.....	11
1.3 Обґрунтування вибору інструментальних засобів	15
Висновки до першого розділу.....	18
2. ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПОРТАЛУ ВИВЧЕННЯ ЯПОНСЬКОЇ МОВИ.....	19
2.1 Опис акторів та алгоритмів роботи.....	19
2.2 Розробка бази даних	24
2.3. Комунікації і послідовність взаємодії об'єктів системи.....	29
2.4. Реалізація системи порталу вивчення японської мови	32
Висновки до другого розділу.....	47
3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ПОРТАЛУ ВИВЧЕННЯ ЯПОНСЬКОЇ МОВИ.....	48
3.1 Порядок встановлення та налаштування системи.....	48
3.2 Опис інтерфейсу програмного продукту.....	53
3.3 Тестування системи порталу вивчення японської мови	64
Висновки до третього розділу	67
ВИСНОВКИ.....	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	69
ДОДАТКИ.....	72

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

FARM – стек розробки, який включає в себе такі технології, як: FastAPI, React, MongoDB.

БД – база даних.

MUI – бібліотека компонентів, що реалізує Material Design від Google.

SSR – генерація веб-сторінок на стороні сервера.

SSG – генерація статичних веб-сторінок.

UI – інтерфейс користувача.

JSON – текстовий формат обміну даними між комп'ютерами.

API – інтерфейс завдяки якому дві або більше програми можуть спілкуватися між собою.

CSS – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Вивчення іноземних мов, зокрема японської, за останні 2 роки набуло неабиякої актуальності. Для когось це гостра необхідність, у силу обставин, у яких вони опинились. Для інших це інструмент, їхня робота, якою вони займаються, аби принести в український простір іноземну літературу, при чому часто, замінюючи цим примірники мовою агресора.

Основною проблемою у вивченні та транслітерації багатьох японських термінів японської мови є обставини, у яких опинилися мовознавці, використовуючи єдину систему японської транслітерації, що була розроблена в першу чергу для російської мови. Хоча вона має під собою науковий підтекст, але у поєднанні з українською мовою, багато питань з приводу вимови, транслітерації та передачі звуків – лишаються невирішеними.

Мета даної кваліфікаційної роботи полягає в розробці порталу, який би поєднав у собі усі останні та сучасні тенденції з передачі японської мови на українську, що з одного боку допоможе учням у вивченні мови, а мовознавцям привнесе певну базу знань, на яку можна опиратися.

Завданням даної кваліфікаційної роботи є аналіз існуючих систем транслітерації японської мови на українську, запровадженні більш сучасних систем, з урахуванням особливостей української фонетики, та створення відповідного порталу для автоматичної транслітерації та навчання.

Об'єктом дослідження є процес вивчення японської мови, що має під собою багато невирішених питань з боку українського правопису, які потрібно проаналізувати на вирішити.

Предметом дослідження є японська мова в цілому, інтеграція її в сучасний український правопис. Упродовж тривалого часу для транскрибування японських слів українською мовою використовували російську систему транскрипції, розроблену Євгенієм Полівановим у 1917 році. Однак використання цієї системи в українській мові є предметом дискусій, оскільки вона заснована на правилах російської фонетики і не завжди точно передає звучання деяких японських складів.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ПОРТАЛУ ВИВЧЕННЯ ЯПОНСЬКОЇ МОВИ

1.1 Технічне завдання на розробку системи

Ціль – розробка системи, яка надає користувачеві інструменти для ефективного вивчення японської мови та автоматизації певних процесів. Даний портал також може бути застосовано для створення японсько-українського словнику на базі англомовних аналогів.

Мета – основна мета цього проекту полягає у створенні функціонального веб-застосунку, що містить різноманітні інструменти та ресурси для полегшення процесу ознайомлення та адаптації у володінні японською мовою. Для досягнення ефективності та практичності навчання, яке охоплює різні категорії та відповідає сучасним стандартам вивчення, визначені вимоги, яким повинен відповідати цей веб-портал. Потрібно належним чином структурувати роботу веб-застосунку, щоб користувач міг належним чином використовувати всі можливості порталу для вивчення японської мови.

Розробка дизайну веб-застосунку буде виконана з використанням бібліотеки інтерфейсу користувача MUI, яка чудово поєднується з проектом на NextJS.

1. Розробка інтерфейсу системи.

1.1. Оскільки MUI підтримує певний стиль серед усіх своїх компонентів, дизайн буде виглядати гармонійно та лаконічно, і при цьому відповідати принципам Material Design від Google, що дозволяє створити сучасний і дружній інтерфейс.

1.2. MUI легко інтегрується у проект NextJS, завдяки вбудованій інтеграції. Підтримка статичного підвантаження та рендерингу на стороні сервера робить MUI ідеальним вибором для Next.js, особливо для оптимізації продуктивності веб-застосунку.

2. Розробка ядра системи.

2.1. Необхідно створити такі програмні модулі:

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

- Авторизація та автентифікація користувачів. Користувачі мають розподілятися на певні ролі, а саме: адміністратор, мовознавець, учень та звичайний користувач.
- Створення японсько-українського словника на базі англомовних аналогів, а саме Jamdict [7]. Цей словник повинен мати зворотну сумісність зі своїм аналогом, для кращих можливостей мацерації у майбутньому.
- Створення системи колекцій включатиме в себе два окремих модулі взаємодії: їх можна створювати та переглядати безпосередньо на сайті. При цьому, має бути певний контроль над видимістю цих колекцій для інших користувачів. Другий модуль – це можливість експорту створеної колекції в інші програмні продукти, а саме Anki.
- Створення певних навчальних інструментів для автоматизації процесу навчання. Тобто, учень матиме можливість сам перевірити свої виконані завдання на помилки.
- Створення певних інструментів наповнення словника для мовознавців, що пришвидшить цей процес. Тобто, автоматична транслітерація японських слів відповідно до чинних систем транслітерації японської мови. Можливість пошуку синонім у українських словниках, перекладу слів з англійської на українську.
- Надати можливість доступу до певного функціоналу веб-застосунку через виклики API, аби надати іншим розробникам інтерфейс для взаємодії та створення інших інструментів на базі створеного застосунку.

2.2. Реалізація системи для надавання адміністратором ролей на сайті.

3. Реалізація збережених даних матиме наступну структуру:

3.1. Дані японсько-українського словника – унікальний UUID в системі, англійський унікальний ID, слово японською, слово хіраганою,

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

слово катаканою, слово ромаджі, транслітерація українською, кунне читання, онне читання, англійський переклад, український переклад та стан затвердження.

3.2. Дані користувача – унікальний UUID в системі, логін, пароль, пошта, роль, прогрес по вивченні двох алфавітів японської мови, список створених колекцій, посилання на аватар профілю, інформація про дату створення профілю.

3.3. Дані колекції – унікальний UUID в системі, назва колекції, опис колекції, публічність, список зі слів, взятих з таблиці словника.

3.4. Дані ролі – унікальний UUID в системі, назва ролі.

3.5. Дані статистики – базова хірагана, хірагана з дакутен, хірагана з дакутен та комбінаціями, базова катакана, катакана з дакутен, катакана з дакутен та комбінаціями.

4. Необхідно заповнити контент сайту демонстраційними даними для демонстрації функціоналу.

Буде реалізовано комбінований метод навчання японської мови, який поєднує в собі елементи лексичного, комунікативного та тестового методів.

Лексичний метод реалізується через можливість створення користувачем власних колекцій з японських слів, які наповнюються на сайті як за допомогою учнів, так і мовознавців. Цей метод дозволяє учням зосередитися на вивченні певної лексики, яка їм найбільш цікава або необхідна.

Комунікативний і тестовий метод використовується в системі вивчення японської абетки. Користувачу показується певний знак з цієї абетки, і він має обрати його значення з запропонованих варіантів. Цей метод допомагає учням не лише вивчити символи абетки, але й навчитися їх розпізнавати та використовувати в контексті.

Користувачі можуть самостійно обирати, які методи їм більше подобаються, та використовувати їх у зручний для себе час. Тестування та система відстеження прогресу допоможуть учням мотивуватися та продовжувати вивчення мови.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

Результатом реалізації поставленого завдання є досить об'ємний веб-орієнтований застосунок для вивчення японської мови, що має під собою обширний функціонал як для мовознавців, так і для учнів. При цьому певна частина функціоналу сайту буде відкритою для усіх користувачів, роблячи авторизацію опціональним рішенням кожного користувача.

1.2 Аналіз існуючого програмного забезпечення

Існує досить багато різноманітних сайтів для вивчення японської мови. Більшість з них направлено на певну цільову аудиторію, але здебільшого це англomовна частина населення. Розроблений додаток буде конкурувати з деякими існуючими аналогами, які мають попит серед учнів які займаються самонавчанням. Розглянемо такі:

Nihongo-e-na, головна сторінка якого зображена на рисунку 1.1 – це онлайн-ресурс, присвячений японській мові та культурі. Основна мета цього сайту це надати корисні матеріали та інструменти для навчання японської мови [24].

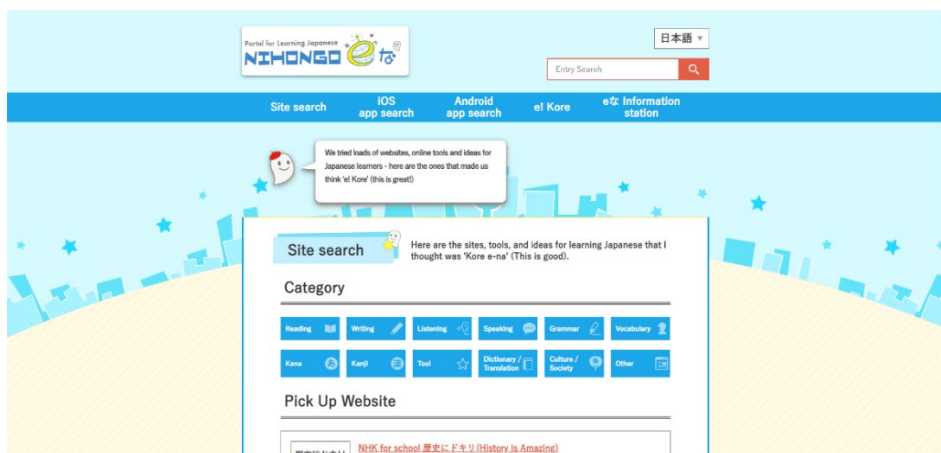


Рисунок 1.1 – Інтерфейс Nihongo-e-na

Переваги:

1. **Комплексність:** Сайт пропонує широкий спектр матеріалів для різних рівнів вивчення японської мови, починаючи від початкового до продвинутого.
2. **Наявність різноманітних ресурсів:** На сайті представлені інтерактивні вправи, відео роки, аудіо матеріали, тексти для читання, граматичні пояснення та інші корисні матеріали.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

3. Структурованість: Матеріали організовані у зручний для навчання формат, що дозволяє користувачам легко знаходити необхідну інформацію.
4. Культурна інтеграція: Окрім мовних матеріалів, сайт також надає інформацію про японську культуру, традиції та звичаї, що допомагає студентам краще зрозуміти контекст вивчення мови.

Недоліки:

1. Можливі обмеження доступу: Деякі розділи або матеріали можуть бути доступними лише за плату або після реєстрації, що може бути обмеженням для користувачів, які шукають безкоштовні ресурси.
2. Потенційна застарілість: Якщо сайт не оновлюється регулярно, матеріали можуть застаріти, а зміни в японській мові або культурі можуть не відображатися.
3. Можливість недостатньої інтерактивності: Деяким користувачам може бракувати інтерактивних можливостей, таких як можливість взаємодії з іншими учнями чи використання технологій штучного інтелекту для персоналізованого навчання.

Jisho, головна сторінка якого зображена на рисунку 1.2 – відомий онлайн-словник японської мови, який надає широкий спектр можливостей для перекладу та пошуку слів і виразів [25].

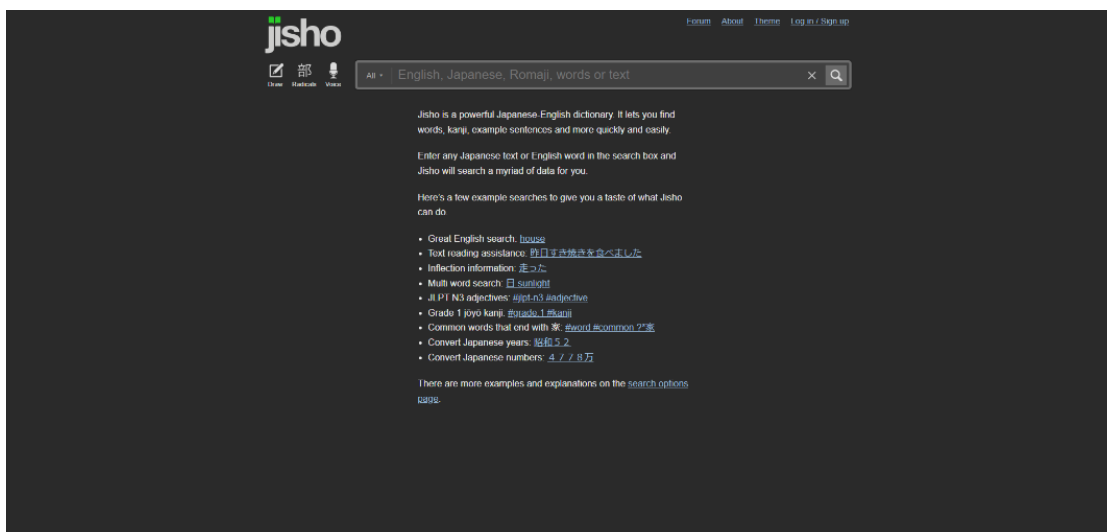


Рисунок 1.2 – Інтерфейс Jisho

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

Переваги:

1. Комплексність: Jisho.org пропонує широкий вибір функцій, включаючи пошук за словами, каної, ромадзі, а також пошук за виразами та фразами. Також доступні додаткові функції, такі як пошук по тегах, синоніми та антоніми.
2. Легкість використання: Сайт має інтуїтивно зрозумілий і простий у використанні інтерфейс, що робить його дружнім до користувача, навіть для початківців.
3. Перекладач: Jisho також функціонує як перекладач, дозволяючи користувачам переводити слова та фрази з англійської мови на японську та навпаки.
4. Додаткові ресурси: Окрім словника, на сайті є розділи з іншими корисними матеріалами, такими як граматичні правила, інформація про кану та ромадзі, що робить його цікавим та корисним для студентів японської мови.

Недоліки:

1. Наявність реклами: На сайті можуть бути рекламні банери, які деяким користувачам можуть викликати дискомфорт або відволікати від основного контенту.
2. Недостатня інтерактивність: Сайт може бути менш інтерактивним порівняно з деякими іншими ресурсами для навчання японської мови, наприклад, відсутність можливості спілкування з іншими користувачами чи використання вбудованих вправ.

Nihonbi, головна сторінка якого зображена на рисунку 1.3 – українська платформа для вивчення японської мови, ведеться у вигляді певного блогу від вчителів [26].

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		13

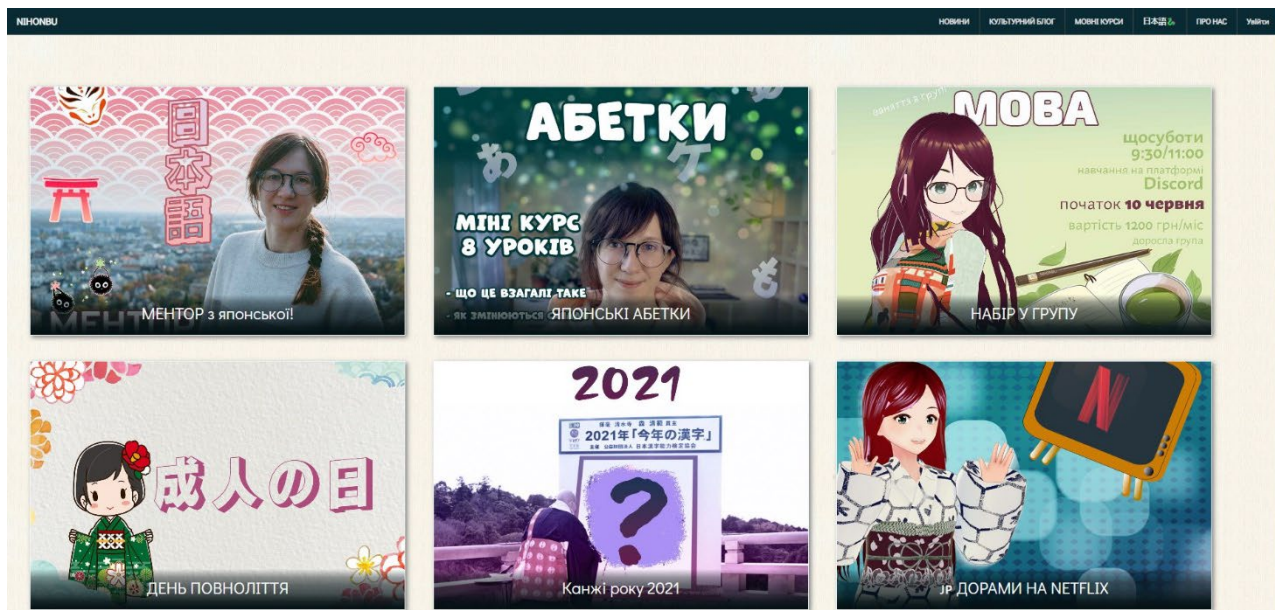


Рисунок 1.3 – Інтерфейс Nihonbu

Переваги:

1. Присутній викладач: учень завжди може звернутися до викладача, якщо в нього присутні якісь проблеми чи питання.
2. Культурний блог: на сайті ведеться певний блог про японську культуру, що допомагає зануритися у вивчення мови.
3. Проведення зустрічей з учнями: вивчення мови наживо, з кваліфікованими спеціалістами.

Недоліки:

1. Усі ресурси на сайті платні.
2. Відсутність будь-яких інструментів для вивчення мови самостійно.

Отже, проаналізувавши всі ці системи, які мають свої власні цілі та методи реалізації, було вирішено рухатися у сторону збору найкращих практик. Тобто, наявність зручного веб-інтерфейсу, модулі навчання, а саме система уроків, створення словника для використання як на сайті, так і за його межами, інтеграція з іншими сервісами. У результаті цього порівняння, було сформовано порівняльну характеристику, зображену у таблиці 1.1.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

Таблиця 1.1

Таблиця порівняння платформ з вивчення японської мови

Критерії	Nihongo-e-na	Jisho	Nihonbu	Портал вивчення японської мови
Є платні матеріали	-	-	+	-
Інтеграція з іншими платформами чи застосунками	-	-	-	+
Ведення внутрішнього словнику японської мови	-	+	-	+
Блог японської мови	+	-	+	-
Наявність вчителя, до якого можна звернутися	-	-	+	-
Наявність інструментів вивчення японських алфавітів	+	+	+	+
Створення колекцій з японських слів	-	-	-	+

1.3 Обґрунтування вибору інструментальних засобів

Для розробки було уважно проаналізовано різноманітні технології та архітектурні підходи з метою вибрати найбільш підходящу для досягнення цілей проекту. В результаті обрана архітектура FARM [18], проте модифікована та більш сучасна, що включає в себе FastAPI [3], React [11] (React замінено на NextJS [10]), та MongoDB [15].

1. FastAPI для бекенду

- Швидкість та ефективність: FastAPI відомий своєю високою швидкістю та ефективністю завдяки використанню сучасних технологій Python та асинхронних операцій. Це дозволяє забезпечити миттєву відповідь на запити користувачів.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

- Документація через Swagger UI: FastAPI надає автоматичну документацію через Swagger UI, що спрощує роботу з API для розробників і полегшує їхню взаємодію з бекендом.

2. MongoDB для бази даних

- Гнучкість у роботі з даними: MongoDB дозволяє легко працювати з різноманітними типами даних, що часто використовуються у системах для вивчення мов.

- Масштабованість: База даних MongoDB добре масштабується, що робить її ідеальним вибором для майбутнього розширення системи зі зростанням користувацької бази та обсягу даних.

3. Next.js для фронтенду

- Швидка реакція сторінок: Next.js надає можливість серверного рендеру та статичної генерації, що забезпечує швидку реакцію сторінок та покращує користувацький досвід.

- Просте управління станом: Next.js спрощує управління станом застосунку через вбудовані можливості, що полегшить розробку та підтримку системи.

Вибір стеку FARM як архітектури застосунку був здійснений з урахуванням кількох факторів, які включають в себе швидкість реалізації системи, гнучкість у роботі з даними, простоту у розробці та масштабованість. Ці технології допоможуть створити надійну та ефективну систему для користувачів, які бажають вивчати японську мову, забезпечуючи їм зручність у використанні та високу якість.

Під час вибору інструментальних засобів для розробки було важливо врахувати функціональність, продуктивність та можливості майбутнього розширення системи. Також було визначено конкретні вимоги до апаратного забезпечення, яке буде використовуватися для роботи системи.

Для створення веб-застосунку було обрано наступні технології:

- JSX для написання веб-сторінок.
- MUI для стилізації компонентів.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

- JavaScript, як фреймворк NextJS, для написання фронтенду.
- Python, для написання бекенду на FastAPI.
- Beanie [5] – як драйвер для БД MongoDB.
- Jamdict – як база для словнику.

JSX [12] – це розширення синтаксису JavaScript, яке використовується в React для опису структури користувацького інтерфейсу. JSX дозволяє розробникам писати HTML-подібний код безпосередньо в JavaScript. Цей підхід полегшує створення складних інтерфейсів, оскільки надає можливість поєднувати логіку та розмітку в одному файлі.

MUI [13][14] – це популярна бібліотека компонентів для React, яка реалізує дизайн-систему Material Design від Google. MUI надає велику кількість готових компонентів, стилізованих відповідно до принципів Material Design, що дозволяє розробникам швидко створювати сучасні, естетично привабливі та зручні для користувачів інтерфейси.

JavaScript, у вигляді Next.js реалізує веб-застосунок:

1. SSR та SSG: NextJS використовується для реалізації фронтенду з метою підвищення швидкості завантаження сторінок. Це надає ефективне керування над завантаженням сторінок.
2. Інтеграція з MUI: Використання MUI разом з NextJS є оптимальним рішенням, оскільки вони мають взаємну інтеграції між собою.

Python реалізує усю логіку веб-застосунку:

1. Продуктивність та швидкість: FastAPI був обраний через його високу продуктивність та швидкість. Це особливо важливо для системи, яка буде у майбутньому обробляти велику кількість запитів користувачів на різних етапах.
2. Автоматична документація: Один з головних плюсів FastAPI – це можливість автоматичної генерації документації за допомогою Swagger UI. Це полегшує розробку та сприяє розумінню API для інших розробників.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		17

3. Сумісність з Jamdict: Оскільки це є ключовою вимогою при побудові словника на сайті, сумісність з цим пакунком є необхідним для подальшої розробки.

Beanie та MongoDB реалізує БД:

1. Beanie – асинхронний об'єктно-документний маппер для MongoDB. Моделі даних базуються на Pydantic [6].
2. Кожна колекція бази даних має відповідний документ, який використовується для взаємодії з цією колекцією. Окрім пошуку даних, він дозволяє додавати, оновлювати або видаляти документи з колекції.

Для розробки веб-застосунку було використано середовище розробки Visual Studio Code.

Висновки до першого розділу

В першому розділі було проведено аналіз завдань, поставлених у дипломній роботі, і на його основі розглянуті різні інструменти для реалізації веб-застосунку. Необхідна реалізація збереження різноманітних даних у створеного словника, системі уроків та користувачів в цілому.

Основні вимоги до застосунку це ефективність освоєння нової мови користувачем, зручність для мовознавців задля наповнення сайту необхідним для учнів контентом, та інтеграція з іншими сервісами.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		18

2. ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ПОРТАЛУ ВИВЧЕННЯ ЯПОНСЬКОЇ МОВИ

2.1 Опис акторів та алгоритмів роботи

Наступним етапом у процесі моделювання порталу є аналіз алгоритму функціонування та створення діаграм станів.

Ключовими учасниками є користувачі з такими ролями: учень, мовознавець і адміністратор, що описані у таблиці 2.1. Взаємодія між ними здійснюється за допомогою веб-інтерфейсу. Поглибимо розуміння основних алгоритмів через моделювання діаграм активності.

Таблиця 2.1

Користувачі

Актор	Короткий опис
Учень	Має можливість наповнювати словник, користуватися словником на сайті, створювати власні колекції зі слів словника, проходити тестування на вивчення двох японських абеток.
Мовознавець	Має можливість робити все, що і учень. Також модерує усі слова словника, які були додані учнями, а не мовознавцями. Має повний доступ до словника.
Адміністратор	Має можливість робити все, що і учень. Також у праві надавати роль мовознавця будь-якому зареєстрованому користувачу на сайті.

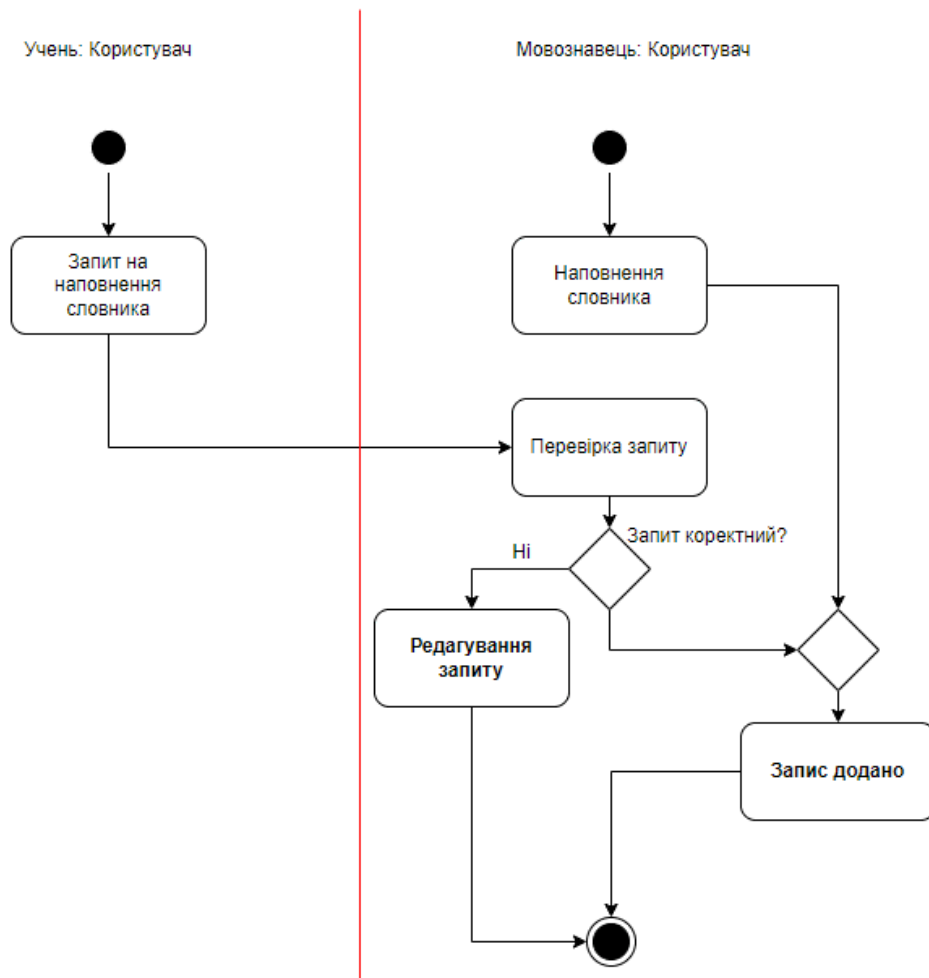


Рисунок 2.1 – Діаграма дій процесу наповнення словника

На рисунку 2.1 зображено порядок дій запиту на наповнення словника, що може ініціювати як мовознавець, так і звичайний учень. Якщо запит надійшов від учня, то мовознавець може опрацювати знайдений запит, та додати його до словника. При цьому не авторизований користувач, тобто звичайний відвідувач сайту, такої можливості немає.

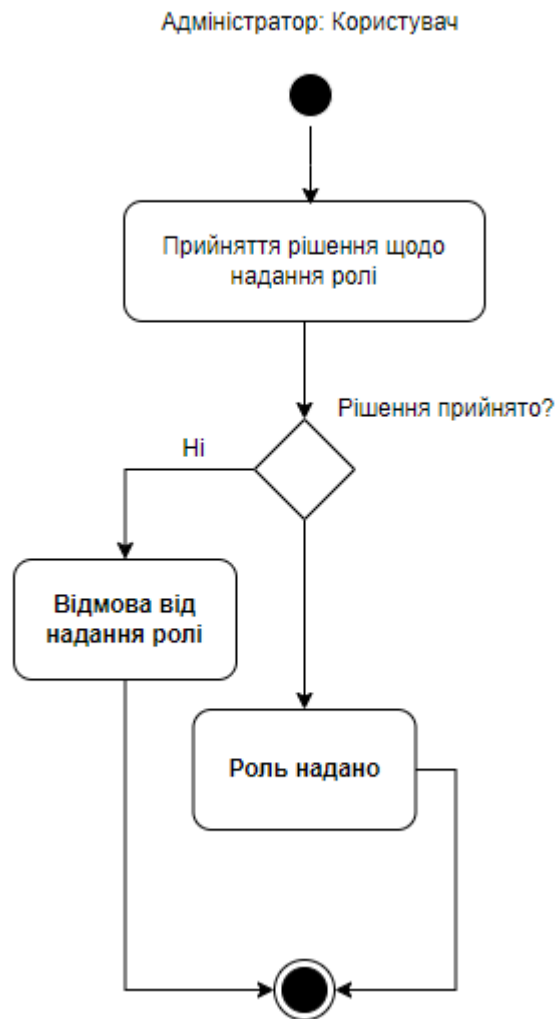


Рисунок 2.2 – Діаграма дій надання ролі мовознавця адміністратором

На рисунку 2.2 зображено порядок дій адміністратора, який на власний розсуд обирає кого на сайті можна обрати мовознавцем. При цьому користувач, що має роль мовознавця не генерується на сайті автоматично, як генерується адміністратор. Тому на адміністратора лежить роль знаходження мовознавця, реєстрація цієї людини на сайті та надання їй відповідної ролі. Якщо цього не зробити, то майже весь модуль з реалізації словника буде недоступний усім іншим користувачам.

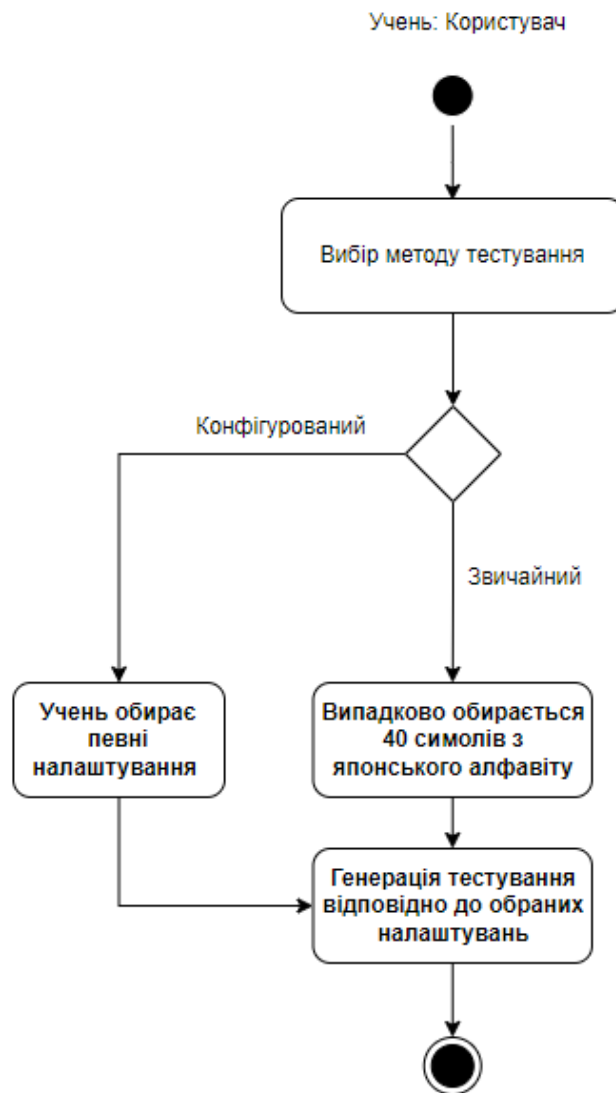


Рисунок 2.3 – Діаграма дій налаштування тестування

На рисунку 2.3 зображено порядок дій користувача на сайті, що може пройти два основних види тестування: на знання хірагани або катакани. При цьому, існує конфігуроване тестування, де користувач може обрати декілька опцій. До таких опцій входять: базові символи, комбінації, дакутен та комбінації дакутен. Поміж цього користувач може обрати кількість запитань, які буде запропоновано за один раз, а саме: 15, 20 та 30 запитань.

Якщо обрати неконфігурований варіант тестування, тоді ці налаштування набувають базових значень, що для хірагани, що для катакани. Тобто, обираються взагалі всі можливі тестові питання з певної абетки з кількістю запитань, що дорівнює 40. Таке тестування можна використовувати, якщо

користувач уже має солідну базу знань зі знань японської абетки, і просто хоче їх перевіряти час від часу.

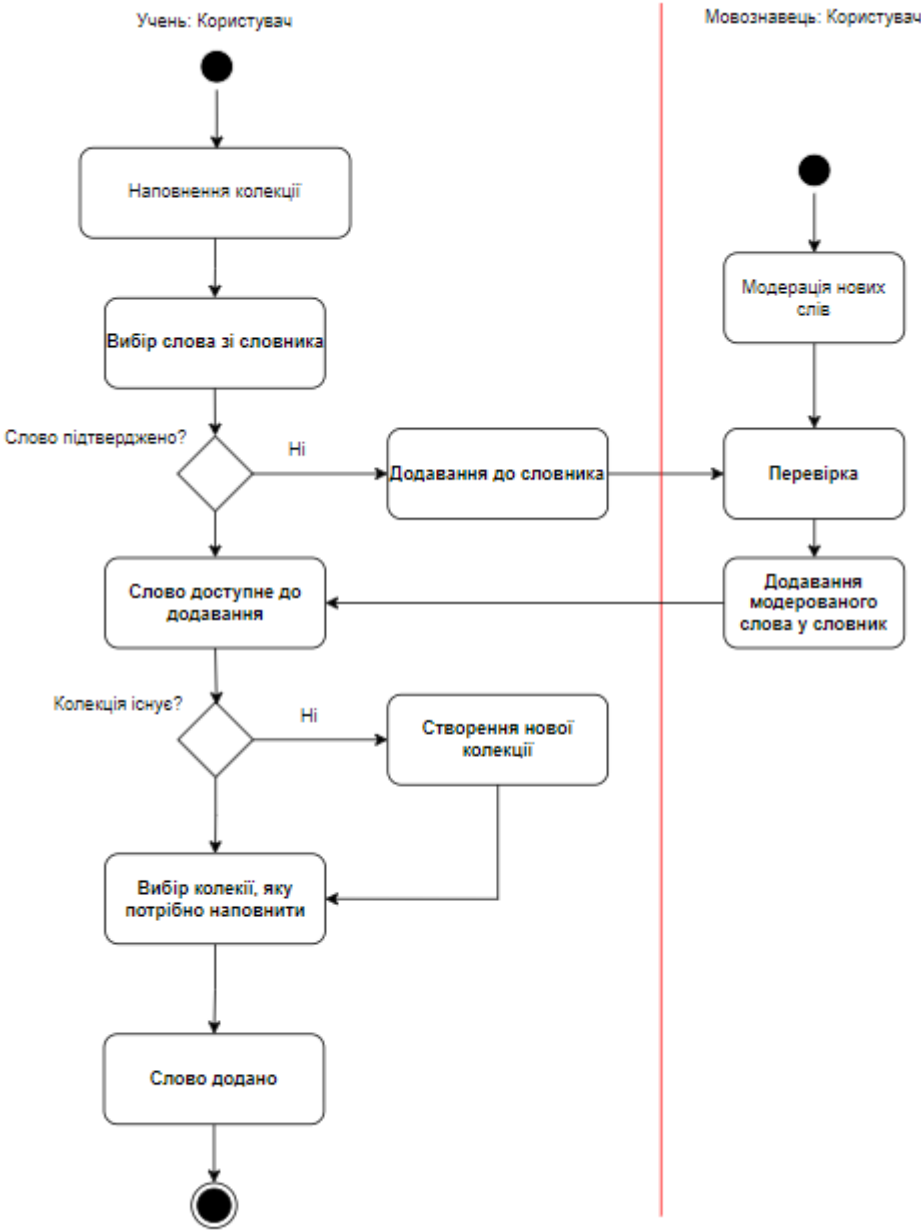


Рисунок 2.4 – Діаграма дій заповнення колекції

На рисунку 2.4 зображено порядок дій користувача сайту, що має змогу створювати власні колекції зі слів, які доступні у словнику. Варто зазначити, що до колекції можна додати лише слова, які пройшли модерацію та були додані до українського словника. Додати на сайт слово, яке було взято з англійського аналогу, проте все ще предствлене на сайті, неможливо. Користувач у такому випадку має власноруч додати це слово до словника, і чекати поки вільний модератор його підтвердить. Після того, як це станеться, чи бажане слово уже

наявне в українському словнику, користувач обирає чи створює нову колекцію, в яку бажає його додати та додає (рис. 2.4).

2.2 Розробка бази даних

Наступним важливим кроком розробки системи освітнього порталу є опис та створення бази даних. База даних складається з 5 таблиць, опис яких можна побачити у таблиці 2.2, розберемо їх.

Таблиця 2.2

Опис таблиць

Назва таблиці	Короткий опис
Dictionary	Містить в собі поля та значення, запозичені з англійського словника та певні додатки, аби реалізувати саме японсько-український словник.
Collection	Містить в собі певні налаштування, зокрема приватності, та список доданих до колекції слів.
Role	Відповідає за визначення ролей.
Stats	Відповідає за формування статистики з проходження тестування на знання японських абеток.
User	Містить в собі поля, необхідні для реєстрації користувача, визначення ролі, посилання на інші дані в таблицях.

Усі таблиці створені за допомогою бібліотеки – бібліотеки Beanie. Для валідації даних, дана бібліотека спілкується з популярним у цьому стеку валідатором Pydantic. Дана комбінація дозволяє оголосити модель таблиці використовуючи єдиний формат запису, який при цьому буде автоматично валідуватися відповідно до параметрів полів, які було визначено. Крім того, кожна з наведених таблиць також має по декілька представлень, тобто схем відображення. Використовуючи модель для запиту даних, та схему для відображення отриманих даних з БД, візуалізація якої зображена на рисунку 2.5,

ми можемо відображати ці дані так, як нам заманеться, до тих пір, поки це не конфліктує з валідацією, яка описана саме в моделі.

Role uuid str name str	Dictionary uuid str idseq num kanji str hiragana str katakana str romaji str transliteration str kunyomi str onyomi str en_translation str ua_translation str approved bool created_at date updated_at date created_by old approved_by old
User uuid str email str username str password str role old collections old stats old avatar_url str created_at str	
Stats hiragana_basic arr [0] str hiragana_kuten arr [0] str hiragana_kuten_combinations arr [0] str katakana_basic arr [0] str katakana_kuten arr [0] str katakana_kuten_combinations arr [0] str	Collection uuid str name str description str is_public bool words arr [0] old

Рис. 2.5 – Візуалізація бази даних

Оскільки Mongo – не реляційна база даних, то зв’язки тут працюють завдяки посиланням на об’єкти всередині БД. Ні в одній моделі, написаній мовою Python немає якихось спеціальних полів, для реалізації тих чи інших зв’язків, які часто можна зустріти у будь-якій реляційній БД. Через цю особливість, при створенні полів, що посилаються на інші поля потрібно враховувати, чи не станеться рекурсії при запиті до БД, і по можливості уникати

подібної поведінки, проте, якщо цього уникнути неможливо, тоді доведеться при запиті до БД налаштовувати як далеко можна отримувати дані.

Для кращого розуміння діаграми розберемо таблиці окремо.

Таблиця 2.3

Таблиця «Користувач»

User	
uuid	UUID
email	str
username	str
password	str
role	Link[Role]
collections	List[Link[Collection]]
stats	Link[Stats]
avatar_url	str
created_at	datetime

Таблиця 2.3 є ключовою на порталі, оскільки всі інші таблиці мають з нею певний зв'язок. Хоча кожне поле в БД MongoDB має свій унікальний ідентифікатор, було вирішено створити додатковий унікальний номер, аби взаємодіяти з БД. Це було зроблено для того, аби не виводити автогенеровану інформацію у вільний публічний простір, а використовувати власні ключі. Пароль користувача хешується за допомогою bcrypt – криптографічної функції формування ключа. Роль користувача визначається посиланням на певну роль у таблиці «Role». Колекції користувача визначається полем, що може зберігати масив даних, і у даному випадку він зберігає масив посилань на таблицю «Collection». Статистика користувача також посилається на таблицю «Stats», але оскільки вона одна, тому зберігати масив непотрібно. Також таблиця зберігає в собі посилання на аватар користувача, та дату створення акаунту.

Таблиця «Словник»

Dictionary	
uuid	UUID
idseq	int
kanji	str
hiragana	str
katakana	str
romaji	str
transliteration	str
kunyomi	str
onyomi	str
en_translation	str
ua_translation	str
approved	bool
created_at	datetime
updated_at	datetime
created_by	Link[User]
approved_by	Link[User]

Таблиця 2.4 містить в собі всю необхідну інформацію, яку мав би містити словник, та яку надає англійське джерело. До цих даних належить: унікальний ідентифікатор всередині англійського словника, поле канджі, хірагани, катакани, ромаджі, транслітерація, кунне читання, онне читання, англійський переклад, український та переклад.

Також існують поля дати додавання запису у таблицю, та дату останнього редагування. Кожне додане слово повинно містити посилання на користувача, хто його створив та посилання на мовознавця, який це слово затвердив у словнику. При цьому поки цей запис існує, затвердження може містити пусте значення.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця «Роль»

Role	
uuid	UUID
name	str

Таблиця 2.5 містить у собі 3 службові записи, які визначають ролі у межах даного застосунку. Ці записи генеруються автоматично при першому запуску застосунку, та мають такі ролі: учень, мовознавець та адміністратор. Зв'язок з цією таблицею показано у таблиці User.

Таблиця «Колекція»

Collection	
uuid	UUID
name	str
description	str
is_public	Bool
words	List[Link[Dictionary]]

Таблиця 2.6 містить у собі певні поля налаштування цієї колекції. Тобто, потрібно вказати назву колекції, її опис та визначити статус публікації. Якщо колекція публічна – інші користувачі сайту будуть бачити її на профілі користувача. Також у колекції є поле зі словами словника, які користувач може до неї додати. Тут ми можемо побачити зв'язок колекції з таблицею Dictionary, при чому це поле є масивом значень з цієї таблиці, а не просто єдиним можливим значенням. Таким чином у кожній колекції може бути безліч слів, що були додані зі словника.

Таблиця «Статистика»

Stats	
hiragana_basic	dict
hiragana_kuten	dict
hiragana_kuten_combinations	dict
katakana_basic	dict
katakana_kuten	dict
katakana_kuten_combinations	dict

Таблиця 2.7 містить статистику відповідно до кожного типу японських символів у алфавіті. При створенні користувача, ця таблиця автоматично генерується і заповнюється початковими значеннями. В кожному полі зберігається певний масив даних, що схожий на JSON і зберігає у собі структуру, що містить певний символ з японського алфавіту та два значення, які відповідають за кількість правильних та неправильних відповідей. При кожному проходженні тестування, ця таблиця оновлюється.

2.3. Комунікації і послідовність взаємодії об'єктів системи

Подальшим етапом в процесі моделювання порталу є розробка діаграм послідовностей, спрямованих на поліпшення розуміння взаємодії компонентів у системі.

Розглянемо процес додавання нового слова до словника звичайним користувачем, що зображено на рисунку 2.6, оскільки аналогічний процес для мовознавця відрізняється лише автоматичним заповненням одного поля. Отже, користувач знаходить слово, яке бажає додати до словника. Робить відповідний запит, під час якого більшість полів заповнюється автоматично, якщо якийсь з полів не підвантажилось автоматично, це означає що потрібна подальша модерація, або певних полів і немає існувати. У нашому випадку, користувач може або сам це перевірити, або відправити слово як є на подальшу модерацію

мовознавцю. Після відправки слова, мовознавець у будь-якому випадку оглядає отриману інформацію і приймає рішення.

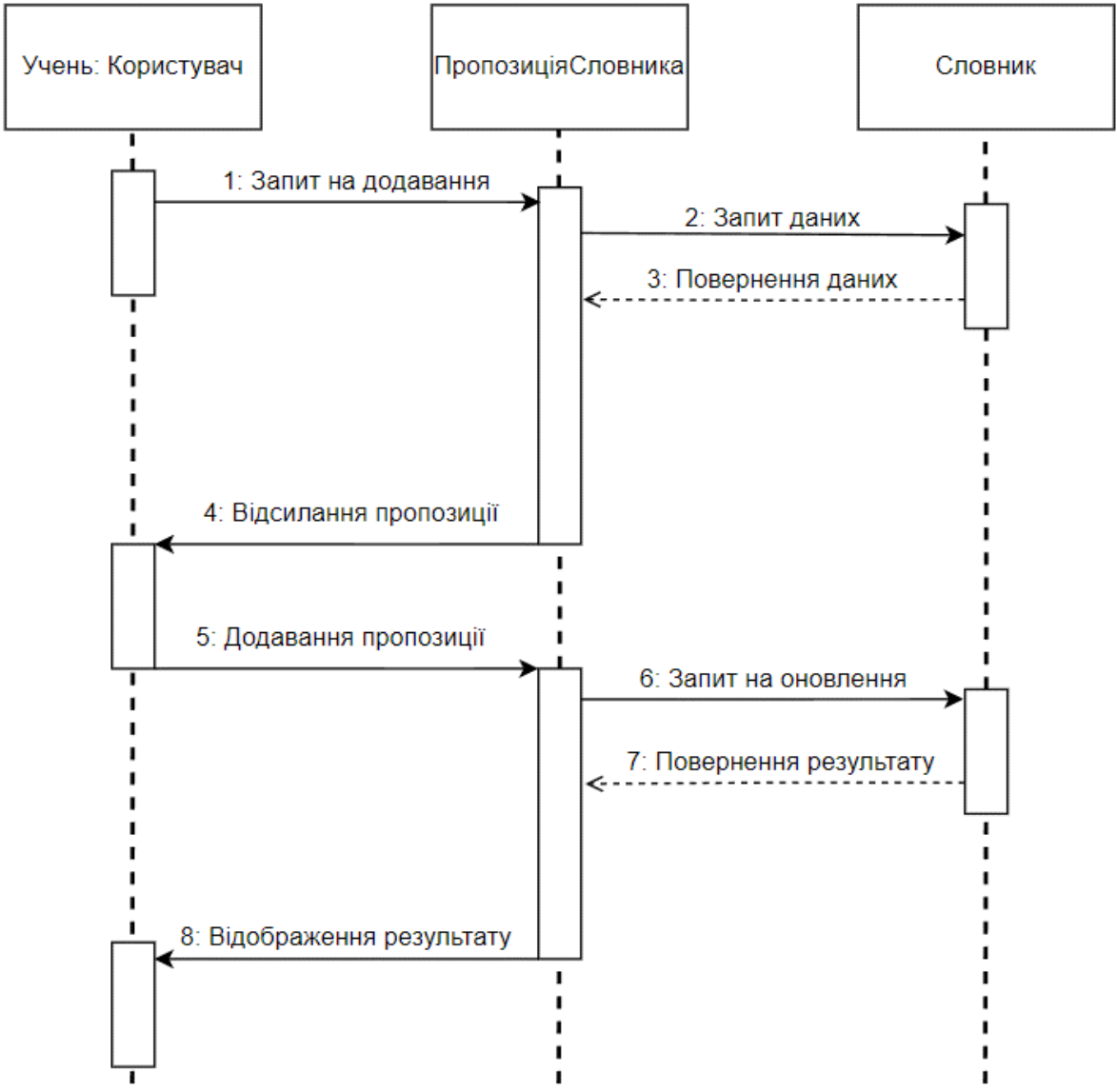


Рисунок 2.6 – Додавання слів до словника

Далі розглянемо процес проходження тестування, який представлено на рисунку 2.7. Користувач сайту звертається до сторінки, яка відповідає за тестування. Якщо ніяких параметрів до цієї сторінки не було передано, використовуються визначені початкові значення цього тестування. Якщо такий сценарій користувача не влаштовує, він може перейти на сторінку, яка дозволить йому сконфігурувати цей процес. На ній, він в праві вибрати по якій добірці символів він має намір проходити тестування, алфавіт та кількість питань, яку буде містити це сконфігуроване тестування. Після проходження тестування, користувачі відобразиться його результат, тобто помилки та правильні відповіді.

Також усі ці відповіді будуть збережені до статистики користувача, що проходив тестування, у вигляді кількості правильних і неправильних відповідей.

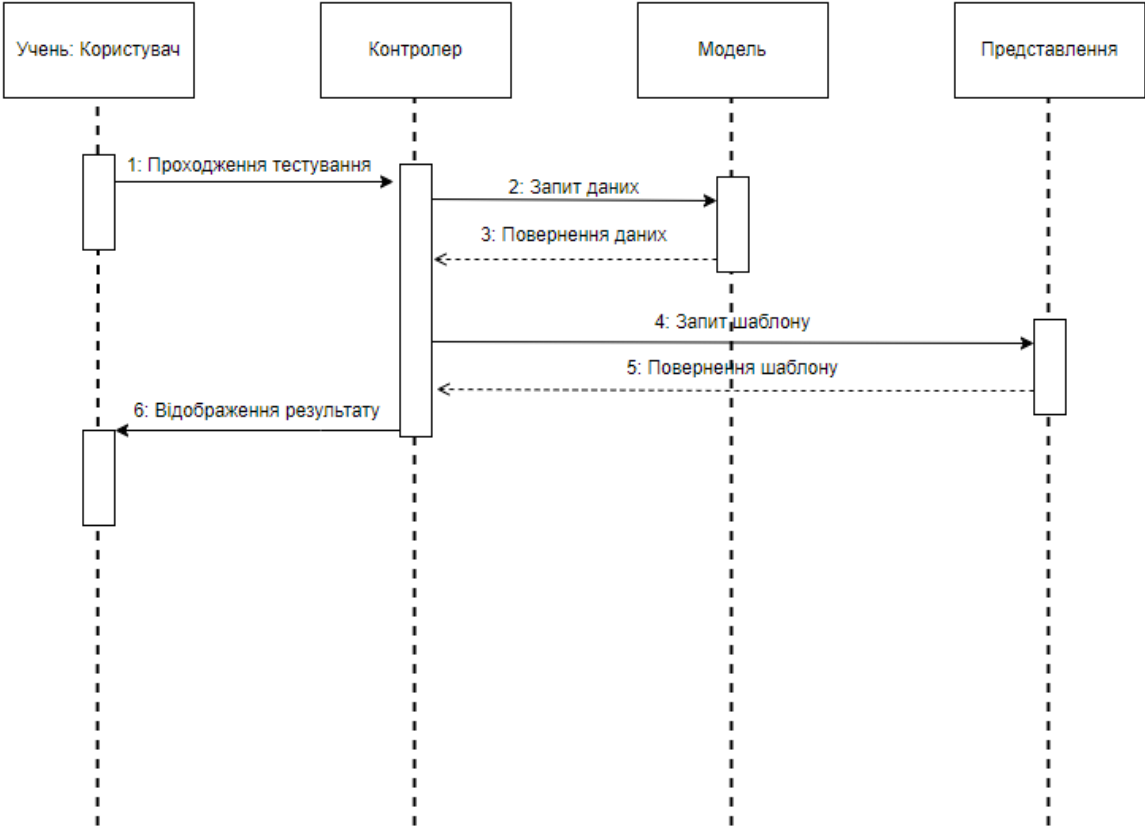


Рисунок 2.7 – Проходження тестування

Також розглянемо процес створення та наповнення колекцій, який зображено на рисунку 2.8. Користувач у праві створювати безліч колекцій. Для цього потрібно перейти на сторінку користувача, натиснути на відповідний елемент додавання колекції, записати назву, опис та модель відображення і створити колекцію. Тепер, якщо користувач буде шукати на сайті слова, він зможе додати їх до цієї колекції, яку він щойно створив. Для цього потрібно обрати слово, обрати колекцію та виконати необхідну дію для додавання запису у цю колекцію. Одне й те саме слово може бути додано в будь-яку колекцію, але один раз.

Якщо користувач хоче щось змінити у колекції, він може знову повернутися на сторінку користувача і відредагувати необхідну колекцію. Такі налаштування як назва чи опис колекції, відбуваються комплексним запитом на

усю колекцію. Якщо ж потрібно змінити лише публічність колекції, це можна зробити натисканням однієї кнопки.

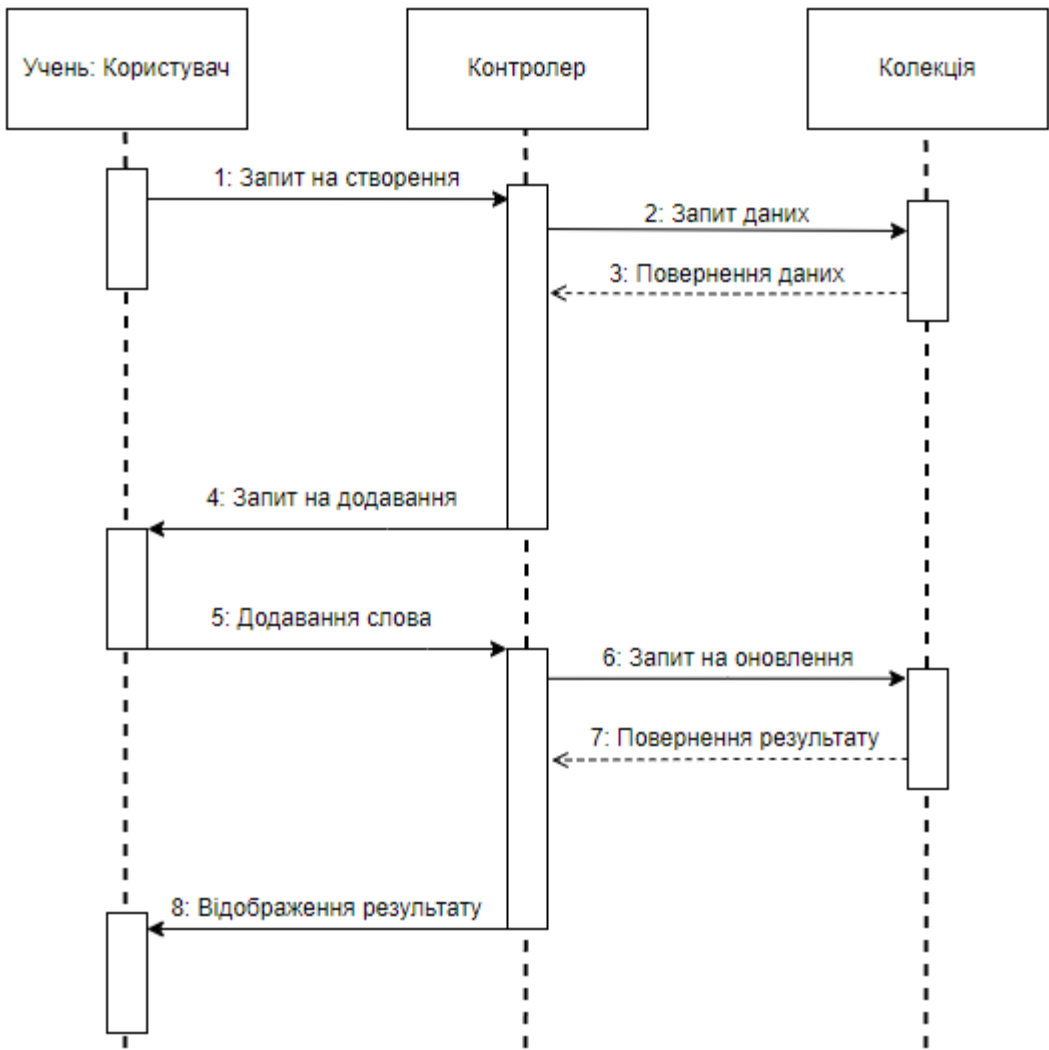


Рисунок 2.8 – Створення колекції

2.4. Реалізація системи порталу вивчення японської мови

Система поділена на дві окремі частини: бекенд та фронтенд. При чому перша частина, тобто бекенд взагалі не залежить від реалізації другої частини. Це дає змогу реалізовувати частину фронтенду будь-яким способом, в даному випадку було обрано NexJS, але інші розробники, які мають інтерес у розробленому застосунку також можуть використовувати написане API для створення своїх застосунків, на будь-яких інших платформах і у будь-якому іншому вигляді.

Тож розглянемо спочатку частину, яка реалізує API.

Основний файл цієї частини проекту має наступний вигляд:

```
from auth import authentication
from contextlib import asynccontextmanager
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
from routers import (user, role, dictionary, collection,
from models.role import Role
from models.user import User
from models.dictionary import Dictionary
from models.collection import Collection
from models.stats import Stats
from helpers.hash import Hash
from beanie import init_beanie
from motor.motor_asyncio import AsyncIOMotorClient
from config import settings

@asynccontextmanager
async def lifespan(app: FastAPI):
    app.client = AsyncIOMotorClient(
        settings.MONGO_HOST,
        settings.MONGO_PORT,
        username=settings.MONGO_USER,
        password=settings.MONGO_PASSWORD,
    )
    await init_beanie(database=app.client[settings.MONGO_DB],
document_models=[User, Role, Dictionary, Collection, Stats])

    role_admin = await Role.find_one({"name": "admin"})
    role_linguist = await Role.find_one({"name": "linguist"})
    role_user = await Role.find_one({"name": "user"})

    if not role_admin:
        role_admin = Role(name="admin")
        await role_admin.create()
    if not role_linguist:
        role_linguist = Role(name="linguist")
        await role_linguist.create()
    if not role_user:
        role_user = Role(name="user")
        await role_user.create()

    user = await User.find_one({"email": settings.ADMIN_EMAIL})
    if not user:
        user = User(
            email=settings.ADMIN_EMAIL,
            password=Hash.bcrypt(settings.ADMIN_PASSWORD),
            username="admin",
            role=role_admin,
            stats=await Stats().create(),
            avatar_url=f"{settings.APP_URL}/static/images/blank_avatar.jpg"
        )
        await user.create()

    yield
```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

```

app = FastAPI(lifespan=lifespan)
app.include_router(authentication.router)
app.include_router(role.router)
app.include_router(user.router)
app.include_router(dictionary.router)
app.include_router(collection.router)
app.include_router(quiz.router)

origins = [
    "*"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.mount("/static", StaticFiles(directory="static"), name="static")

```

Цей файл імпортує та оголошує весь проект. А саме, імпортуються усі шляхи, які реалізують відповідний функціонал, усі моделі, необхідні для того, аби запустити та ініціалізувати БД, створення користувача, який має роль адміністратора та налаштування збереження певних статичних файлів, які надходять з проектом, та які можуть генеруватися у результаті отримання певних запитів. Всі налаштування застосунку, які потребують заповнення, знаходяться у наборі змінних середовища, що також валідуються за допомогою бібліотеки Pydantic, яка валідує усі таблиці в БД.

Надалі, цей проект можна умовно розділити на 5 головних частини, а саме: моделі, що описують дані в БД, роутери, що описують шляхи та логіку до API, схеми, що описують представлення даних БД, допоміжні функції, що реалізують певну логіку, і використовуються у шляхах API, та модуль який відповідає суто за автентифікацію.

Розглянемо кожну частину. Почнемо з коду, що відповідає за шлях API словника:

```

from datetime import datetime
from typing import List
from uuid import UUID
from beanie.operators import RegEx
from beanie.odm.operators.find.logical import Or

```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

```

from fastapi import APIRouter, Depends, HTTPException
from auth.oauth2 import get_current_user
from schemas.dictionary import (
    DictionaryBase, DictionaryMassSearch, DictionaryCreate)
from models.user import User
from models.dictionary import Dictionary
from helpers.dicts import get_kanji_info

router = APIRouter(
    prefix="/dictionary",
    tags=["Dictionary"]
)

@router.post("/")
async def create_entry(request: DictionaryCreate, current_user: User =
    Depends(get_current_user)):
    dictionary = Dictionary(
        idseq=request.idseq,
        kanji=request.kanji,
        hiragana=request.hiragana,
        katakana=request.katakana,
        romaji=request.romaji,
        transliteration=request.transliteration,
        kunyomi=request.kunyomi,
        onyomi=request.onyomi,
        en_translation=request.en_translation,
        ua_translation=request.ua_translation,
        created_by=current_user.id,
        approved=current_user.role.name == "linguist",
        approved_by=current_user if current_user.role.name == "linguist" else
None
    )

    try:
        await dictionary.save()
        return dictionary
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.get("/entry/{uuid}", response_model=DictionaryBase)
async def get_entry_by_uuid(uuid: UUID):
    try:
        dic = await Dictionary.find_one(Dictionary.uuid == uuid)
        return dic
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.get("/search/{query}", response_model=DictionaryMassSearch)

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

```

async def search(query: str):
    dic = await Dictionary.find_many(Or(
        RegEx(Dictionary.kanji, query, 'i'),
        RegEx(Dictionary.hiragana, query, 'i'),
        RegEx(Dictionary.katakana, query, 'i'),
        RegEx(Dictionary.ua_translation, query, 'i'),
        RegEx(Dictionary.romaji, query, 'i'),
        RegEx(Dictionary.transliteration, query, 'i'),
    )).to_list()

    dic_base_list = [DictionaryBase(**d.model_dump())
                     for d in dic if d.approved]
    local = get_kanji_info(query)

    filtered_local = [l for l in local if l['idseq']
                     not in [d.idseq for d in dic_base_list]]

    results = DictionaryMassSearch(
        en_source=filtered_local, ua_source=dic_base_list)

    return results

@router.get("/jamdict/{idseq}")
def get_jamdict(idseq: int):
    result = get_kanji_info(idseq, deeplapi=True)[0]

    return result

```

Представлено лише незначну частину цього файлу, але цього достатньо аби наглядно побачити, як може виглядати роутер та які функції виконувати. Можна побачити декілька шляхів API. Почнемо зі шляху, що реалізує додавання слова до словника. Запит залежить від схеми, яка була спеціально об'явлена для того, аби запрошувати від запиту необхідні дані. Якщо якихось з обов'язкових даних не буде у запиті, повернеться відповідна помилка. Цей шлях також залежить від функції `get_current_user`, що означає, що цей він захищений автентифікацією на сайті. Тобто, аби отримати доступ до цього шляху, користувач має авторизуватися на сайті, або, на трохи технічному рівні, мати певний токен, який би дозволив йому виконати цей запит. У випадку, якщо цей користувач дійсно існує, ми можемо навіть звертатися до його полів у БД та витягувати звідти необхідні дані, як у цьому прикладі ми перевіряємо, чи є користувач мовознавцем.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

Наступний шлях, це пошук по словнику. Тут ми вже не бачимо ніяких залежностей, а це означає що його може виконати абсолютно будь-хто. Пошук розрахований на пошук по певним полям, які об'явлені відповідними параметрами пошуку. У даному випадку, пошук відбувається за полями: канджі, хірагана, катакана, український переклад, ромаджі та транслітерації. Дані параметри знаходять усе, що буде мати у собі хоча б частину із запиту. Для того, аби не дублювати дані з українського та англійського словників, усі запити фільтруються та відправляються у відповідь відповідно до оголошеної схеми.

Та самий простий, проте один з найголовніших шляхів, що відповідає за пошук у локальному, англійському словнику. Він немає жодних параметрів, які б його якось конфігурували у межах застосунку, оскільки це окремий компонент, що має просто повертати дані з англійського словнику по його унікальному ідентифікатору. Оскільки ця функція, по суті, універсальна до усього проекту, то ми завжди маємо отримувати перший елемент масиву.

Тепер розглянемо одну з моделей:

```
from typing import Annotated, List, Optional, TYPE_CHECKING
from datetime import datetime
from uuid import UUID, uuid4
from beanie import Document, Indexed, Link
from pydantic import EmailStr, Field
from config import settings
from models.role import Role
from models.stats import Stats

if TYPE_CHECKING:
    from models.collection import Collection

class User(Document):
    uuid: UUID = Field(default_factory=uuid4)
    email: Annotated[EmailStr, Indexed(unique=True)] = None
    username: Annotated[str, Indexed(unique=True)] = None
    password: str = None
    role: Link[Role] = None
    collections: Optional[List[Link['Collection']]] = []
    stats: Optional[Link[Stats]] = None

    avatar_url: Optional[str] =
f'{settings.APP_URL}/static/images/blank_avatar.jpg'
    created_at: Optional[datetime] = Field(default_factory=datetime.now)
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

Ця модель оголошує таблицю користувача у БД. Усі моделі наслідують певний Document, що надає бібліотека Beanie. Завдяки цьому ми можемо користуватися бібліотекою Pydantic для валідації даних, що надходять у таблицю та реалізовувати певні зв'язки між таблицями, функціонал якого також надає Beanie. Наприклад, поле email валідується використовуючи звичайні налаштування бібліотеки Pydantic у поєднанні зі звичайним функціоналом самої мови програмування Python. У той час поле role уже використовує запис зв'язку між таблицями, що у даному випадку дає нам можливість записати користувачу його роль з іншої таблиці. А також ці обидва варіанти комбінуються у полі collections. Використовується як і стандартні виклики Python, так і зв'язки. Тобто, дивлячись на таке оголошення, ми можемо сказати що це поле буде містити масив даних, що містить посилання на таблицю колекції. Такі поля як avatar_url чи created_at мають свої початкові значення, де у першому випадку це посилання на аватар користувача, що зберігається на сервері, а у другому виклик отримання поточної дати. Раніше згадана проблема вирішується тут за допомогою перевірки значення константи TYPE_CHECKING, оскільки валідацією займається Pydantic у середовищі мови Python, тому подібний імпорт викликав би рекурсію.

Далі розглянемо приклад оголошення схем даних:

```
from typing import List, Optional
from pydantic import BaseModel
from uuid import UUID
from schemas.dictionary import DictionaryCollection

class CollectionBase(BaseModel):
    uuid: UUID
    name: str
    description: Optional[str]
    is_public: bool
    words: Optional[List[DictionaryCollection]]

class CollectionCreate(BaseModel):
    name: str
    description: Optional[str]
    is_public: bool

class CollectionUpdate(BaseModel):
    name: Optional[str] = None
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

```
description: Optional[str] = None
is_public: Optional[bool] = None
```

Код моделей та схем виглядає подібно, проте варто зазначити, що це дві абсолютно різні речі. Суть полягає у тому, що із запиту, отриманого з БД, який відповідає його оголошеній моделі, ми можемо виключати чи включати необхідні нам поля, як це зроблено тут. Нам не потрібно вказувати значення поля `uuid`, для того, аби оновити чи створити нашу колекцію. Тому у відповідних схемах, ці поля і виключені. Подібний підхід до оголошення даних є досить зручним, оскільки не завжди користувачу потрібно відповідати усіма полями, що є у таблиці. Іноді, потрібно вказати лише деякі з них. Це допомагає знизити навантаження на сервер, та пришвидшити час відповіді певних запитів.

Далі розглянемо приклад допоміжної функції:

```
import genanki
import os
from slugify import slugify

ANKI_MODEL_ID = 1424945485
ANKI_DECK_ID = 1517805082

def generate_anki_deck(collection):
    model = genanki.Model(
        ANKI_MODEL_ID,
        'Japanese',
        fields=[
            {'name': 'Kanji'},
            {'name': 'Hiragana'},
            {'name': 'Katakana'},
            {'name': 'Transliteration'},
            {'name': 'Ua_Translation'}
        ],
        templates=[
            {
                'name': 'Card 1',
                'qfmt': '{{Kanji}}',
                'afmt': '{{FrontSide}}<br>{{Hiragana}}
({{Katakana}})<br>{{Transliteration}}<br>{{Ua_Translation}}',
            }
        ]
    )

    deck = genanki.Deck(ANKI_DECK_ID, collection.get('name'))

    for word in collection.get('words'):
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    note = genanki.Note(
        model=model,
        fields=[
            word.get('kanji'),
            word.get('hiragana'),
            word.get('katakana') if word.get('katakana') else '',
            word.get('transliteration'),
            word.get('ua_translation')
        ]
    )
    deck.add_note(note)

package = genanki.Package(deck)

if not os.path.exists('static/files'):
    os.makedirs('static/files')

out = f'static/files/{slugify(collection.get("name"))}.apkg'

package.write_to_file(out)

return out

```

Ця функція використовується для генерації файлу колекції, який був би сумісним з застосунком Anki. Згідно документації, подібний файл має містити деякі «унікальні» значення, що мають згенеруватися один раз та використовуватися для подальшої генерації усіх файлів. Цей фрагмент коду використовується у одному з роутерів, який передає йому дані певної колекції. На основі цих даних і генерується файл, що здатна прочитати програма Anki. Таким чином було реалізовано інтеграцію застосунку із зовнішніми програмами.

Далі розглянемо частину, що відповідає за автентифікацію та розподіл дозволів по шляхам:

```

from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from typing import Optional
from datetime import datetime, timedelta
from jose import JWTError, jwt
from config import settings
from models.user import User
from schemas.user import UserPrivate

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

SECRET_KEY = settings.SECRET_KEY
ALGORITHM = 'HS256'

```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

```
ACCESS_TOKEN_EXPIRE_MINUTES = 24 * 60 * 365
```

```
def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.now() + expires_delta
    else:
        expire = datetime.now() + timedelta(minutes=15)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt
```

```
async def get_current_user(token: str = Depends(oauth2_scheme)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Invalid Credentials",
        headers={"WWW-Authenticate": "Bearer"})

    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username = payload.get("sub")
        if username is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception

    user = await User.find_one(User.username == username, fetch_links=True)

    if user is None:
        raise credentials_exception

    return user

async def admin_check(current_user: UserPrivate = Depends(get_current_user)):
    if current_user.role.name != "admin":
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="You don't have admin permissions to perform this action."
        )
    return True

async def linguist_check(current_user: UserPrivate =
    Depends(get_current_user)):
    if current_user.role.name != "linguist":
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="You don't have linguist permissions to perform this
action."
        )
    return True
```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

Цей файл описує алгоритм шифрування токєну, отримання якого гарантує доступ до усіх шляхів, що ним захищені. Ми уже бачили залежність від функції `get_current_user`, і тут ми зможемо детально оглянути як вона працює. Ця функція очікує у запиті певний Bearer токен, який у свою чергу, генерується функцією `create_access_token`, що використовується при авторизації. Якщо токен вірний, то буде повернуто поточного користувача, якщо ні, то буде повернуто відповідно помилку. Також у файлі представлені ще декілька залежностей, що допомагають визначити роль користувача, у шляхах, де це є необхідним. Тобто, вказавши якусь з цих функцій як залежність, ми отримаємо валідацію щодо певної ролі користувача, який би мав отримати доступ до певного шляху.

Все, що було описано до цих пір, це лише логіка застосунку. Проте ще має існувати якийсь інтерфейс, тому перейдемо до нього. Для реалізації інтерфейсу було обрано NextJS. Цю частину також можна умовно поділити на декілька частин, це шляхи компонентів, контекст, допоміжні функції та самі компоненти. Розглянемо їх окремо.

Почнемо з головного макету усіх сторінок:

```
import { AppRouterCacheProvider } from '@mui/material-nextjs/v14-appRouter'
import { ThemeProvider } from '@mui/material/styles';
import CssBaseline from '@mui/material/CssBaseline';
import theme from './MuiTheme';
import Footer from '@components/Footer';
import Header from '@components/Header';
import { NotificationProvider } from '@context/NotificationContext';

export default function RootLayout({ children }) {
  return (
    <html>
      <body style={{ display: 'flex', flexDirection: 'column', minHeight: '100vh' }}>
        <NotificationProvider>
          <AppRouterCacheProvider options={{ enableCssLayer: true }}>
            <ThemeProvider theme={theme}>
              <CssBaseline />
              <Header />
              {children}
              <Footer />
            </ThemeProvider>
          </AppRouterCacheProvider>
        </NotificationProvider>
      </body>
    </html>
  )
}
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		42

```

    </html>
  )
}

```

Даний макет реалізує досить просту структуру звичайної HTML сторінки, описаної за допомогою JSX використовуючи певні компоненти. З візуального боку, у цьому макеті генерується хедер, футер та місце під основний контент. З функціонального, налаштовується використання бібліотеки MUI для відображення її стилів, вкладається провайдер для сповіщень та налаштовуються певні налаштування самого NextJS. В подальшому цей макет буде викликатися при побудові будь-якої іншої сторінки на сайті.

Наступний код реалізує зберігання та отримання Bearer токена, необхідного для авторизації:

```

'use server';

import { cookies } from 'next/headers';

export async function setCookie(name, value) {
  cookies().set(name, value, {
    maxAge: 365 * 24 * 60 * 60,
    httpOnly: true,
    sameSite: 'lax'
  });
}

export async function deleteCookie(name) {
  cookies().delete(name);
}

export async function getCookie(name) {
  return cookies().get(name)?.value;
}

export async function getCookies() {
  return cookies().getAll().values();
}

```

NextJS надає можливість отримання доступу до cookie незалежно від того, де їх потрібно отримати. Це може бути як серверний компонент, що рендериться на сервері, так і клієнтський компонент, що рендериться на стороні клієнта, і це навіть звучить логічніше, оскільки cookie зберігаються а браузері користувача, а

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

не на сервері, де виконується серверний код. Проте NextJS має можливість діставати ці cookie будь-де. Більшість з цих функцій реалізують функціонал по менеджменту даних cookie і можуть бути викликані у будь-якому з компонентів для доступу до цих значень, чи певної маніпуляції над ними.

Розглянемо реалізацію пошуку, який присутній у хедері сторінки:

```
"use client";
```

```
import SearchIcon from '@mui/icons-material/Search';
import { Search, SearchIconWrapper, StyledInputBase } from
'./HeaderClientComponents';
import { useState } from 'react';
import { useRouter } from 'next/navigation';

export default function HeaderSearch() {
  const [search, setSearch] = useState('');
  const router = useRouter();

  const searchSubmitHandler = (e) => {
    e.preventDefault();
    if (search.trim() === '') {
      return;
    }
    e.target.value = '';
    router.push(`/search/${search}`);
  }

  return (
    <Search>
      <SearchIconWrapper>
        <SearchIcon />
      </SearchIconWrapper>
      <StyledInputBase
        placeholder="Пошук..."
        inputProps={{ 'aria-label': 'search' }}
        onChange={e => setSearch(e.target.value)}
        onKeyDown={e => {
          if (e.key === 'Enter') {
            searchSubmitHandler(e);
          }
        }}
      />
    </Search>
  )
}
```

Оскільки цей компонент має отримувати певні дані від користувача та використовує звичний функціонал реакту, він має бути позначеним, як клієнтський. З цього коду можна побачити логіку, яка реалізує перехід на інший

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

шлях, описаний в застосунку, який в свою чергу уже буде мати логіку з отримання даних по цьому запиту з пошуку. Розглянемо його наступним:

```
import { Suspense } from 'react';
import DictionarySearch from '@components/DictionarySearch';

import { getCurrentUser } from '@utils/user';

export const fetchCache = 'only-no-store';

async function getUser() {
  const user = await getCurrentUser();
  return user;
}

export default async function Search({ params }) {

  const user = await getUser();

  return (
    <Suspense>
      <DictionarySearch currentUser={user} search={params.search} />
    </Suspense>
  );
}
```

Цей компонент відповідає за шлях, по якому переходить запит з пошуку хедера. Цей компонент є серверним, й усі компоненти в NextJS є серверними, доки не вказано інакше. По цьому шляху відбувається отримання даних поточного користувача, оскільки від цього залежить функціонал, який доступний у наступному компоненті, у який уже передається сам запит до словника, та інформацію про поточного користувача. Де компонент DictionarySearch уже буде повністю клієнтським, оскільки користувач повинен мати змогу маніпулювати даними на цій сторінці.

Далі розглянемо провайдер сповіщень:

```
"use client";
import React, { createContext, useState, useContext, useCallback } from
'react';
import { Snackbar, Alert } from '@mui/material';

const NotificationContext = createContext();

export const NotificationProvider = ({ children }) => {
  const [notification, setNotification] = useState({
    message: '',

```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    severity: 'info', // 'success', 'error', 'warning', 'info'
    open: false,
  });

  const showNotification = useCallback((message, severity = 'info') => {
    setNotification({ message, severity, open: true });
  }, []);

  const handleClose = () => {
    setNotification({ ...notification, open: false });
  };

  return (
    <NotificationContext.Provider value={showNotification}>
      {children}
      <Snackbar
        anchorOrigin={{ vertical: 'top', horizontal: 'right' }}
        open={notification.open}
        autoHideDuration={3000}
        onClose={handleClose}
      >
        <Alert onClose={handleClose} severity={notification.severity}
sx={{ width: '100%', marginTop: 7 }}>
          {notification.message}
        </Alert>
      </Snackbar>
    </NotificationContext.Provider>
  );
};

export const useNotification = () => useContext(NotificationContext);

```

Він також має бути клієнтським, по згаданим вище причинам. Цей провайдер можна імпортувати у будь-який відповідний компонент, та викликати як функцію, що буде показувати на сайті певне сповіщення. Цей функціонал є дуже корисним, оскільки може надавати користувачеві якійсь інформаційні повідомлення при виконання певних операцій. Якщо щось було дійсно зроблено, то можна відзвітувати про успішність операції, а якщо ні – то вивести помилку про це.

Дана частина проекту містить ще дуже багато інших компонентів, але в основному вони всі мають подібну структуру. Тобто, отримання даних з серверної чи клієнтської сторони та рендеринг сторінок в залежності від отриманих даних. Весь інтерфейс набуває свого стилю в основному за

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

допомогою бібліотеки MUI, та в деяких випадках дописується вручну за допомогою інтерфейсу, що допомагає вписувати CSS стилі прямо у компонент.

Висновки до другого розділу

У даному розділі було досить детально описали певні алгоритми, що присутні у застосунку, варіанти використання, які можуть виникнути при використанні застосунку та описали БД. Також було чітко описано внутрішню структуру обох частин даного проекту. Оголошення таблиць, виклик певних допоміжних функцій, використання шляхів API, та їх захист за допомогою залежностей. В той час, як у другій частину повністю описано процес реалізації усіх шляхів, збереження певних даних, отримання та відображення усіх шляхів, що реалізує перша частина. На основі цього і було розроблено систему порталу вивчення японської мови.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						47
Змн.	Арк.	№ докум.	Підпис	Дата		

3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ПОРТАЛУ ВІВЧЕННЯ ЯПОНСЬКОЇ МОВИ

3.1 Порядок встановлення та налаштування системи

Для отримання уявлення про взаємодію компонентів у системі порталу необхідно створити діаграму компонентів, яка відобразить залежності та взаємозв'язки між різними компонентами системи.

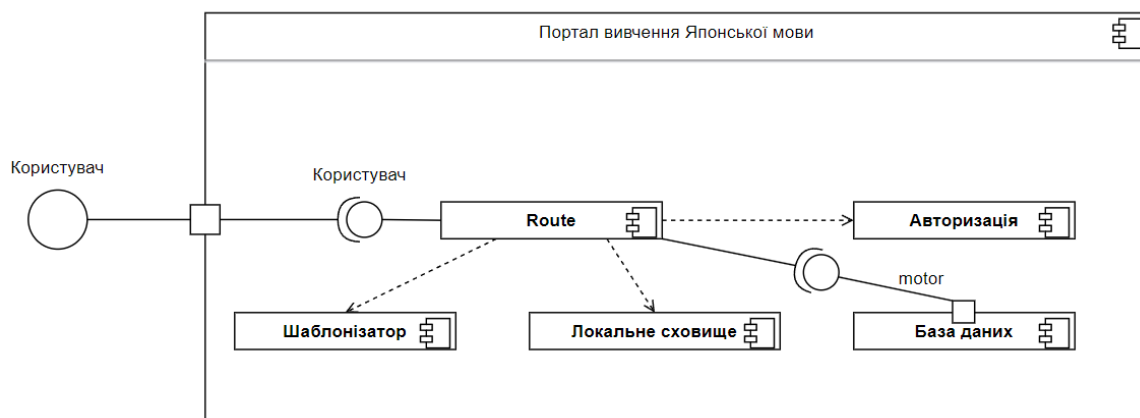


Рис. 3.1 Діаграма компонентів

Діаграма, зображена на рисунку 3.1, показує, як всі компоненти взаємодіють між собою. Користувач взаємодіє з маршрутизатором, який перенаправляє користувача на правильну сторінку. Сторінка, яку відвідує користувач, перевіряє автентичність користувача за допомогою компонента авторизації. Якщо користувач авторизований, компонент авторизації надає доступ до БД. База даних містить дані, які використовуються для відображення сторінок порталу.

Детально опишемо кожен компонент:

- Користувач – це людина, яка використовує систему. Користувач може взаємодіяти з системою за допомогою веб-браузера. Користувач може виконувати такі дії, як перегляд сторінок, маніпулювати та отримувати дані.
- Route – це маршрутизатор, який відповідає за перенаправлення користувача на правильну сторінку. Route отримує запит від користувача і використовує його для визначення сторінки, яку потрібно відобразити.

- Авторизація – це компонент, який відповідає за перевірку автентифікації користувача. Авторизація отримує запит від користувача і використовує його для визначення того, чи має користувач доступ до системи.
- Motor – це фреймворк, що надає API на основі коду для неблокуючого доступу до MongoDB.
- Шаблонізатор – це компонент, який використовується для генерації HTML-коду. Шаблонізатор отримує дані від інших компонентів системи і використовує їх для генерації HTML-коду, який використовується для відображення сторінок порталу.
- Локальне сховище – це сховище, яке використовується для зберігання тимчасових даних. Локальне сховище може використовуватися для зберігання даних, таких як кешовані сторінки або дані, які потрібно використовувати в кількох компонентах системи.
- База даних – це сховище, яке використовується для зберігання довгострокових даних. База даних може використовуватися для зберігання даних, таких як навчальний матеріал, користувачі та їхні налаштування.

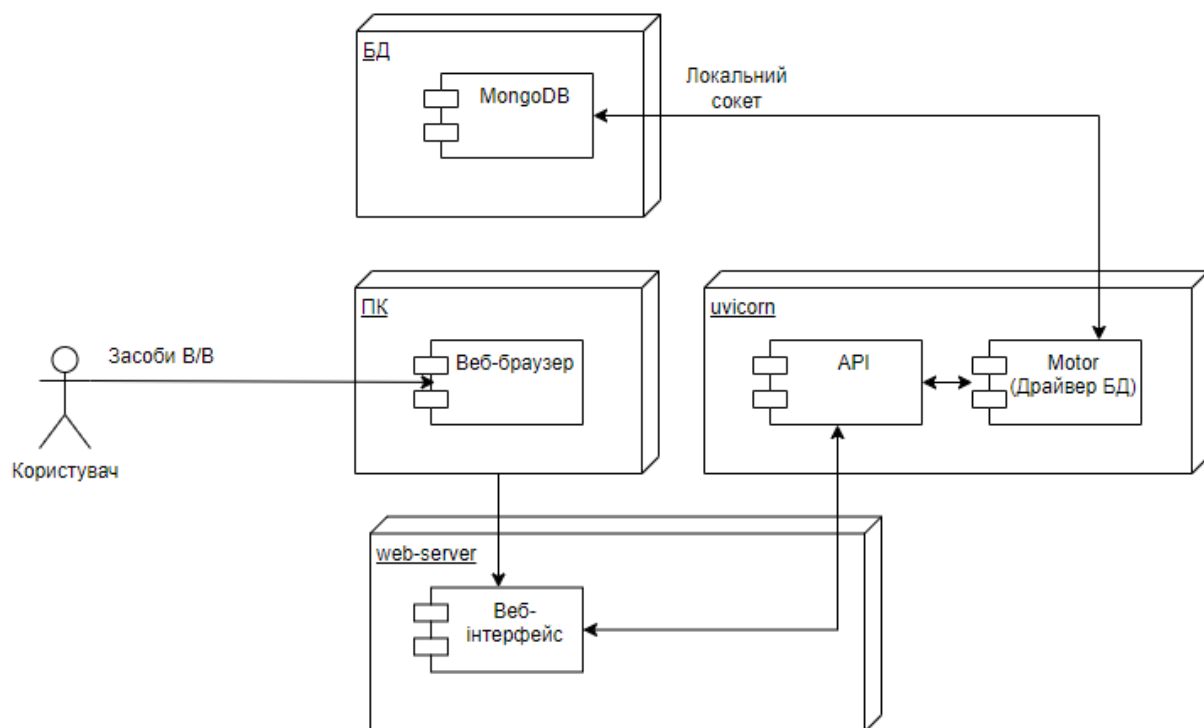


Рис. 3.2 Діаграма розгортання

На діаграмі розгортання, зображений на рисунку 3.2, показано, як користувач взаємодіє із системою, а всі інші компоненти цієї системи – між собою. Видно, що веб-сервер звертається через виклики API до іншого серверу, а саме uvicorn [4], який в свою чергу відповідає за ці запити і хостить основну частину розробленого застосунку. Він, у свою чергу взаємодіє з БД через драйвер motor. Тоді як сама БД, а саме MongoDB розміщується на своєму окремому сервері.

Після реалізації діаграми розгортання, стало зрозуміло, що необхідно забезпечити розгортання 3 окремих її модулів, а саме: БД, API та веб-застосунок. Потрібно врахувати, що всі ці модулі досить затратні по ресурсам, тому необхідно визначитися з досить потужним сервером. Зі сторони користувача особливих вимог немає, потрібно мати лише комп'ютер з доступом в інтернет, який у свою чергу має доступ до будь-якого браузера, з якого можна відвідати веб-застосунок.

Для розгортання проекту, було використано Docker [16]. Кожній частині було написано свій Dockerfile, за допомогою якого відбувається побудова контейнера, в якому буде працювати та чи інша частина. Розглянемо цей процес детальніше на прикладі 3 описаних модулів.

Сервер uvicorn описано наступним чином:

```
FROM python:3.12-slim as base

ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

RUN mkdir -p /app/static/files /app/static/images

COPY certs/privkey.pem /etc/letsencrypt/live/gakushuu.pp.ua/privkey.pem
COPY certs/fullchain.pem /etc/letsencrypt/live/gakushuu.pp.ua/fullchain.pem

FROM base as final
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

```
CMD ["unicorn", "main:app", "--host", "0.0.0.0", "--port", "443", "--ssl-keyfile", "/etc/letsencrypt/live/gakushuu.pp.ua/privkey.pem", "--ssl-certfile", "/etc/letsencrypt/live/gakushuu.pp.ua/fullchain.pem"]
```

Для того, щоб згенерувати контейнер та запустити цей модуль необхідно виконати наступні команди:

- docker build -t gakushuu-api .
- docker run -d --name gakushuu-api -p 8008:8000 gakushuu-api

Також, для успішного запуску даного проекту, потрібно подбати про наступні елементи середовища, які потрібно зберегти у корні застосунку у файлі під назвою «.env»:

- APP_URL – домен, на якому буде застосунок.
- MONGO_HOST – локальна адреса БД.
- MONGO_DB – назва БД.
- MONGO_PORT – порт БД.
- MONGO_USER – користувач БД.
- MONGO_PASSWORD – пароль користувача БД.
- SECRET_KEY – секретний ключ.
- ADMIN_EMAIL – пошта адміністратора.
- ADMIN_PASSWORD – пароль адміністратора.
- DEEPL_API_KEY – API ключ сервісу DeepL [9].

Також потрібно подбати і про генерацію сертифікатів до відповідного домену.

Веб-сервер описано наступним чином:

```
# Build
FROM node:20-alpine AS builder

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build
```

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		51

```
# Deploy
FROM node:20-alpine

WORKDIR /app

COPY --from=builder /app/package*.json ./
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/.next ./next
COPY --from=builder /app/public ./public

# Run
CMD ["npm", "start"]
```

Для того, щоб згенерувати контейнер та запустити цей модуль необхідно виконати наступні команди:

- docker build -t gakushuu-web .
- docker run -d --name gakushuu-web -p 3000:3000 gakushuu-web

Також, для успішного запуску даного проекту, потрібно подбати про один елемент середовища у файлі «next.config.js», що матиме наступний вигляд:

```
module.exports = {
  env: {
    NEXT_PUBLIC_API_URL: 'https://api.gakushuu.pp.ua',
  },
}
```

Оскільки для виведення застосунків у веб-простір застосовується інструментарій, який надає Cloudflare [17], потрібно також виконати наступні дії:

- Отримати домен та припаркувати його на Cloudflare.
- Створити тунель Cloudflare.
- Отримати команду docker, що має запустити певний міст між локальною мережею та сервісами Cloudflare та запустити її на сервері.
- Налаштувати, які саме локальні сервіси необхідно вивести в інтернет, вказати домен або субдомен.

База даних також буде запущено у докері, використовуючи відповідний docker-compose [23], що надається Mongo.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

3. Опис інтерфейсу програмного продукту

Під час переходу до сайту, користувача вітає головна сторінка (рис 3.3).

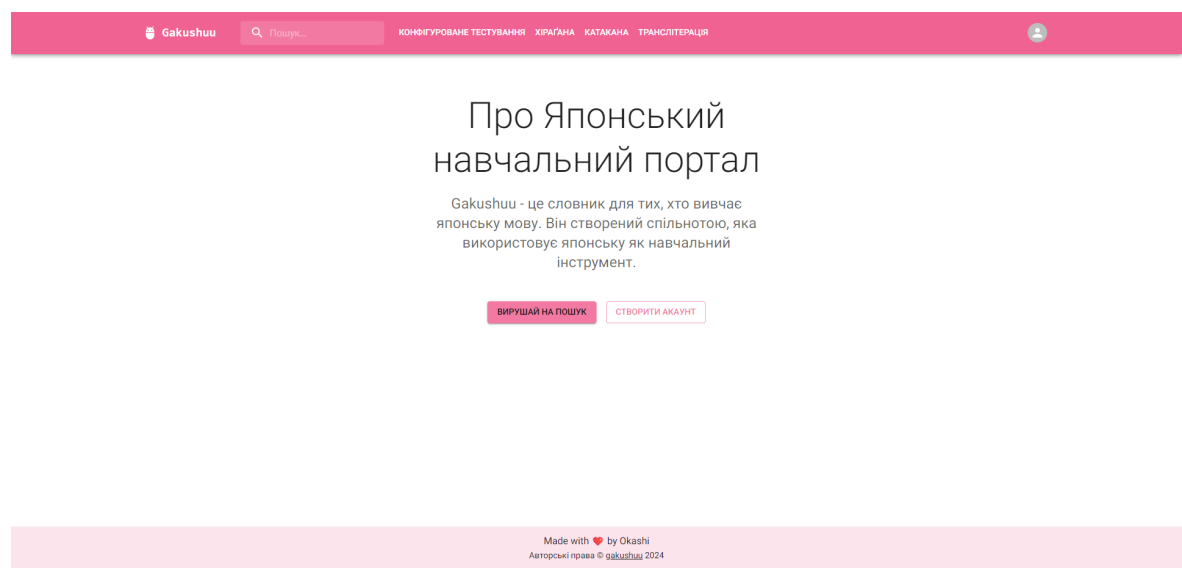


Рисунок 3.3 – Головна сторінка

З цієї сторінки користувач уже може почати користуватися певними сервісами, проте пропонується вже ж таки створити акаунт. Кнопка «Вирушай на пошук» направить користувача на пошук по словнику, вказавши назву сайту японською як критерій пошуку. Це надасть змогу користувачеві зрозуміти, як працює пошук у системі.

Розглянемо результат пошуку зі сторони різних користувачів.

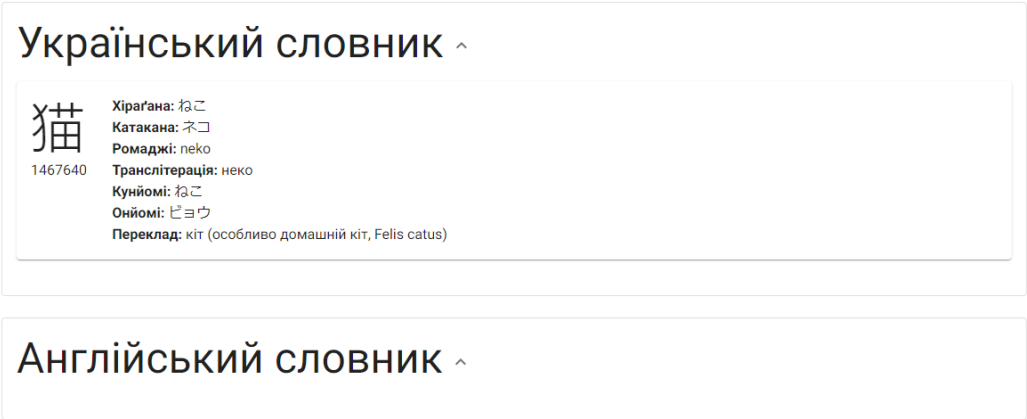


Рисунок 3.4 – Пошук неавторизованого користувача.

Український словник ^

猫

1467640

Хіраґана: ねこ
Катакана: ネコ
Ромаджі: neko
Транслітерація: неко
Кунйомі: ねこ
Онйомі: ビョウ
Переклад: кіт (особливо домашній кіт, Felis catus)

Колекції

Моя класна колекція ▾

ДОДАТИ ДО КОЛЕКЦІЇ

Англійський словник ▾

Рисунок 3.5 – Пошук учня.

Український словник ^

猫

1467640

Хіраґана: ねこ
Катакана: ネコ
Ромаджі: neko
Транслітерація: неко
Кунйомі: ねこ
Онйомі: ビョウ
Переклад: кіт (особливо домашній кіт, Felis catus)

Колекції

Приватна колекція ▾

ДОДАТИ ДО КОЛЕКЦІЇ

РЕДАГУВАТИ

Англійський словник ▾

Рисунок 3.6 – Пошук мовознавця.

В залежності від ролі користувача у системі, хоча результат пошуку завжди буде однаковим, функціонал, доступний різним користувачам буде дещо відрізнятися. Неавторизований користувач може лише бачити результати пошуку (рис. 3.4). Учень, тобто звичайний авторизований користувач сайту може додавати слова до створених колекцій (рис. 3.5). Тоді як мовознавець може до цього ж перейти до сторінки швидкого редагування слова (рис. 3.6).

Пошук по словнику має ще декілька особливостей, про які варто згадати.

Якщо слово буде відсутнє в одному з окремих словників, тобто, чи в українському, чи англійському, відповідний словник буде згорнуто і натомість показано той, який дійсно має результат. При цьому, пріоритет віддається саме українському словнику, тобто, якщо український словник щось знайшов разом з англійським словником, пріоритет на показ буде надано українському словнику.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

Інакше, відобразиться англійський словник (рис. 3.7). Також, дані з обох словників не дублюються.

Український словник ▾

Англійський словник ^

雇用
1267860

Хіраґана: こよう
Ромаджі: koyou
Транслітерація: койоу
Кунйомі: やと.う
Онійомі: コ
Англійський переклад: employment (long term)

ДОДАТИ ДО СЛОВНИКА

Рисунок 3.7 – Пріоритет розгортання.

Сторінка конфігурованого тестування має наступний вигляд (рис. 3.8).

☒ Хіраґана
☐ Катакана

☒ Базові
☐ Комбінації
☐ Дакутен
☐ Комбінації + дакутен

☒ 15 Питань
☐ 20 Питань
☐ 30 Питань

ПОЧАТИ ТЕСТУВАННЯ

Рисунок 3.8 – Конфігурація тестування.

Користувачу пропонується обрати які саме частини з японських алфавітів він бажає тестувати. На цій сторінці він може обрати або хірагану, або катакану. Інші налаштування вже підпорядковуються саме цим алфавітам. Також користувач має можливість обрати кількість запитань. Після вибору необхідних налаштувань, потрібно почати тестування натиснувши на відповідну кнопку.

Це ж тестування, але у преконфігурованому варіанті можна пройти натиснувши на будь-яке з посилань навігації: «Хіраґана» та «Катакана» (рис. 3.3). Воно буде більш об’ємним за запропоновані варіанти конфігурованого тестування, та міститиме геть усі символи з алфавіту.

Процес тестування відбувається наступним чином.

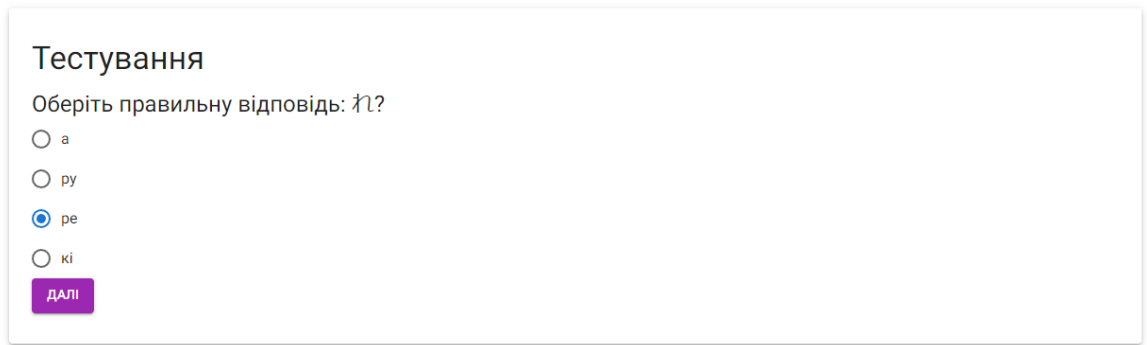


Рисунок 3.9 – Процес тестування.

Користувачу пропонується пройти певне згенероване тестування (рис 3.9).

Після проходження, користувач матиме змогу побачити усі правильні та неправильні відповіді, які він вказав. Також його проінформує про те, що результат цього тестування було записано до статистики цього користувача (рис 3.10). Також є можливість повторити тестування з тими ж самими налаштуваннями, що і цього разу, проте зрозуміло, що порядок тестування буде уже зовсім іншим.



Рисунок 3.10 – Результат тестування.

Переглянути згадану статистику можна на сторінці статистики користувача. Якщо натиснути на фото свого профілю, можна побачити які сторінки для переходу доступні для поточного користувача. Зараз нас цікавить сторінка «Статистика» (рис. 3.11).

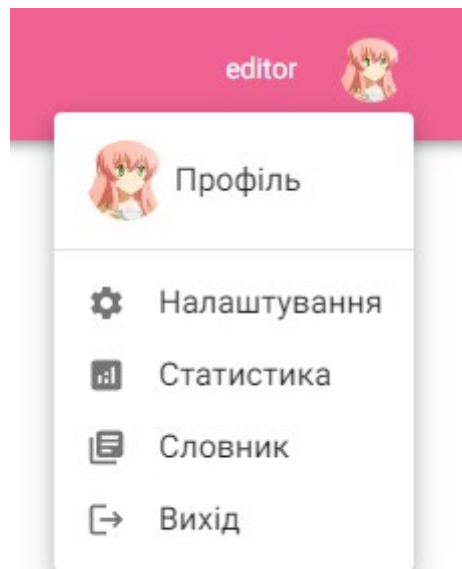


Рисунок 3.11 – Сторінки користувача.

Тож, статистика користувача має наступний вигляд.

Базові			Дакутен			Комбінації + дакутен		
Ієрогліф	Статистика (+/-)	Прогрес (%)	Ієрогліф	Статистика (+/-)	Прогрес (%)	Ієрогліф	Статистика (+/-)	Прогрес (%)
あ	1 / 0	100.00 %	が	0 / 0	0 %	きゃ	0 / 1	0.00 %
い	0 / 1	0.00 %	ぎ	0 / 0	0 %	きゅ	0 / 0	0 %
う	0 / 0	0 %	く	0 / 1	0.00 %	きょ	0 / 0	0 %
え	1 / 0	100.00 %	げ	0 / 0	0 %	しゃ	0 / 0	0 %
お	0 / 0	0 %	ご	0 / 1	0.00 %	しゅ	0 / 1	0.00 %
か	0 / 2	0.00 %	ざ	0 / 0	0 %	しょ	0 / 0	0 %
き	0 / 0	0 %	じ	0 / 1	0.00 %	ちゃ	0 / 0	0 %
く	0 / 0	0 %	ず	0 / 1	0.00 %	ちゅ	0 / 1	0.00 %
け	1 / 0	100.00 %	ぜ	0 / 0	0 %	ちょ	0 / 0	0 %
こ	0 / 1	0.00 %	ぞ	0 / 0	0 %	にゃ	0 / 1	0.00 %
さ	0 / 0	0 %	だ	0 / 0	0 %	にゅ	0 / 1	0.00 %
し	0 / 1	0.00 %	ぢ	0 / 0	0 %	によ	0 / 0	0 %
ず	0 / 0	0 %	づ	0 / 1	0.00 %	ひゃ	0 / 1	0.00 %
せ	0 / 0	0 %	で	0 / 0	0 %	ひゅ	0 / 1	0.00 %
そ	0 / 1	0.00 %	ど	0 / 1	0.00 %	ひょ	0 / 1	0.00 %
た	0 / 2	0.00 %	ば	0 / 0	0 %	みゃ	0 / 0	0 %
ち	0 / 0	0 %	び	0 / 0	0 %	みゅ	0 / 1	0.00 %
つ	0 / 0	0 %	ぶ	1 / 0	100.00 %	みょ	0 / 0	0 %
て	0 / 1	0.00 %	べ	0 / 0	0 %	りゃ	0 / 0	0 %
と	1 / 0	100.00 %	ぼ	0 / 0	0 %	りゅ	0 / 0	0 %

Рисунок 3.12 – Статистика користувача.

На цій сторінці можна побачити статистику по кожному алфавіту окремо, аби побачити катакану, потрібно пролистати нижче, де буде згенеровано таблицю з такою ж структурою до хірагани. Статистика розділена на 3 окремі частини, це «базові» символи, «дакутен», та «комбінації з дакутен». Це зумовлено тим, що при навчання «базові» символи алфавітів вивчають найпершими, оскільки дві інші категорії це по суті розширення до вивченого, тому є логічним розділення на такі частини. Сама ж статистика представляє

собою кількість правильних та неправильних відповідей, і відсоток прогресу по кожному з символів у алфавіті японської мови.

Також на сайті представлена сторінка «Транслітерація», яка потрібна скоріш мовознавцям і виконує суто інформаційну роль.

Рисунок 3.13 – Транслітерація.

На цій сторінці представлено лише поле для вводу, куди можна ввести будь-яку комбінацію з японських символів та отримати його транслітерацію за частково реалізованою системою Коваленка [1][2]. Ця сторінка підтримує транслітерацію як і хірагани, так і усіх канджі. Проте варто пам'ятати, що, наприклад, японські імена, записані канджі даним методом транслітеруються неправильно. Це зумовлено багатьма особливостями японської мови, тому, якщо потрібно транслітерувати саме ім'я, варто записати його саме хіраганою.

Далі розглянемо сторінку користувача (рис 3.14). На ній можна дізнатися певну інформацію про цього користувача, а саме його нікнейм, пошту, роль та дату приєднання. Якщо користувач перебуває на своїй сторінці, то він також має можливість завантажити новий аватар, створити нову колекцію чи редагувати уже наявні колекції. Колекцію можна зробити публічною чи приватною, чого можна досягти двома способами. Або натиснути кнопку видимості, або перейти до меню редагування усіх її полів. Також бажану колекцію можна експортувати у форматі Anki.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						58
Змн.	Арк.	№ докум.	Підпис	Дата		

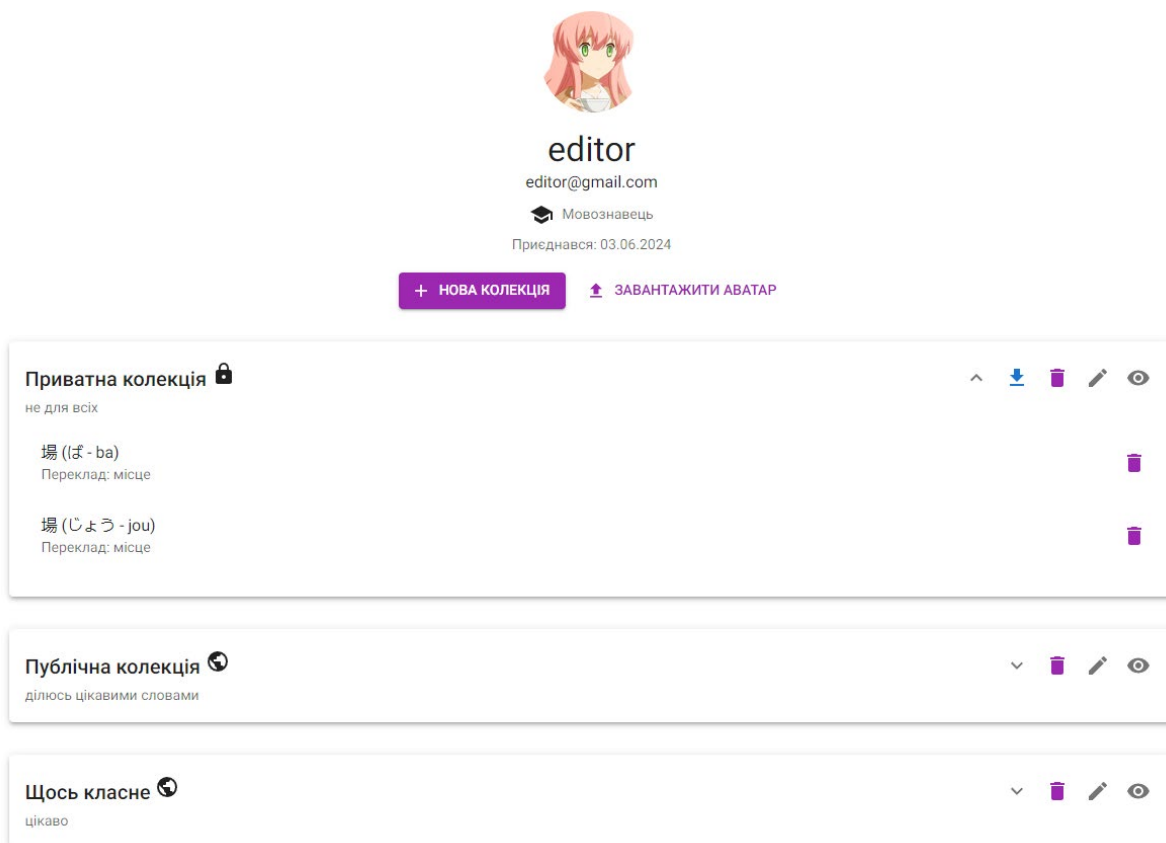


Рисунок 3.14 – Сторінка користувача.

Перейшовши на додаткову сторінку налаштувань (рис. 3.15), користувач має змогу змінити свій нікнейм, для цього потрібно ввести поточний пароль та нове значення нікнейму. На цій же сторінці можна повністю видалити свій акаунт із системи.

Рисунок 3.15 – Додаткові налаштування.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

Тепер розглянемо інтерфейс мовознавця для керування словником.

Gakushuu											
КОНFIGУРОВАНЕ ТЕСТУВАННЯ ХІРАГАНА КАТАКАНА ТРАНСЛІТЕРАЦІЯ											
editor											
IDSEQ	Канджі	Хірагана	Катакана	Ромаджі	Транслітерація	Куніомі	Онйомі	Переклад	Англійський переклад	Додано	Підтверджено
1355790	場	ば	バ	ba	ba	ば	ジョウ	місце	place	editor	editor
2652950	場	じょう	ジョウ	jou	джьюу	ば	ジョウ	місце	place	editor	editor
1189670	佐	か	カ	ka	ка		カ	прекрасний	beautiful	editor	editor
2028970	か	カ	ka	ка				Так?	yes?	editor	editor
1194900	花	はな	ハナ	hana	хана	はな	カ	квітка	flower	editor	editor
1258330	犬	いぬ	イヌ	inu	ину	いぬ	ケン	собака (Canis (lupus) fami...	dog (Canis (lupus) famili...	editor	editor
1467640	猫	ねこ	ネコ	neko	неко	ねこ	ビョウ	кіт (особливо домашній к...	cat (esp. the domestic cat, ...	editor	editor
1302680	山	やま	ヤマ	yama	яма	やま	ザン	гора	mountain	editor	editor
0	string	string	string	string	string	string	string	string	string	admin	editor
2029120	さ	サ	sa	са				нескожість	-ness	admin	editor

Made with ❤️ by Okachi
Авторські права © gakushuu 2024

Рисунок 3.16 – Сторінка керування словником.

Мовознавець у праві приховувати певні поля з цієї таблиці, фільтрувати дані за певними полями, користуватися пошуком, змінювати вигляд та експортувати його, якщо це необхідно. Окрім того, натиснувши на значок редагування поля, дозволені поля в цій таблиці стають доступними до змін прямо з цього ж інтерфейсу (рис. 3.17).

2652950	場	じょう	ジョウ	jou	джьюу	ば	ジョウ	місце	place	editor	editor	СХВАЛИТИ	ВИДАЛИТИ
---------	---	-----	-----	-----	-------	---	-----	-------	-------	--------	--------	----------	----------

Рисунок 3.17 – Вигляд редагованого рядка.

Після внесення змін, користувач натискає кнопку збереження даних, що відразу зберігає та відображає внесені зміни. Якщо потрібно відмінити внесені зміни, то користувач може зробити це натиснувши відповідну кнопку. Також існують дві окремі кнопки для підтвердження слів, що надійшли від звичайних користувачів та видалення запису з БД. З цієї сторінки користувач також може перейти на сторінку інших користувачів, що вказані в стовпцях «Додано» та «Підтверджено», які відповідають за посилання на користувача, що додав це слово та користувача що підтвердив його після модерації.

Розглянемо процес додавання слова у цей словник зі сторони звичайного користувача. Для цього, користувачу спершу потрібно отримати слово з англійського словника, аби додати його до словника. Після цього, потрібно натиснути кнопку «Додати до словника» (рис. 3.18).

Український словник ▾

Англійський словник ▲

居

2258160

Хірагана: きょ

Ромаджі: кyo

Транслітерація: кyo

Кунйомі: いゐる

Онйомі: キヨ

Англійський переклад: residence

ДОДАТИ ДО СЛОВНИКА

居

2258170

Хірагана: い

Ромаджі: i

Транслітерація: i

Кунйомі: いゐる

Онйомі: キヨ

Англійський переклад: being (somewhere)

ДОДАТИ ДО СЛОВНИКА

Рисунок 3.18 – Вибір слова.

Результатом цих дій буде відповідна форма на додавання (рис. 3.19), що спершу отримає це слово з англійського словника за його унікальний ідентифікатором та згенерує певні поля, якщо їх не вистачає. Одним з таких полів є переклад на українську мову. Це поле не береться нізвідки, а автоматично перекладається за допомогою перекладача DeepL. Так само і поле транслітерації, воно керується тими ж правилами, що і відповідна сторінка на сайті. Проте не варто сильно довіряти будь-яким згенерованим полям, у випадку додавання слова користувачем, це можна опустити, проте якщо цим займається мовознавець він має ретельно перевірити, чи всі поля дійсно правильні, і якщо ні – відредагувати їх.

Канджі

居

Хірагана *

い

Катакана

イ

Кунне читання

いゐる

Онне читання

キヨ

Англійський переклад

being (somewhere)

Переклад *

бути (десь)

Ромаджі *

i

Транслітерація *

i

ДОДАТИ

Рисунок 3.19 – Форма додавання.

Після натискання кнопки «Додати», слово буде додане у словник та користувача перенаправить на пошук найближчого значення цього слова, якщо воно було додано звичайним користувачем, адже цей запис все ще має пройти модерацию від мовознавця. Після того як це станеться, воно з'явиться у глобальному пошуку для усіх. Якщо це слово додає мовознавець, воно автоматично підтверджується ним же, тому після натискання кнопки «Додати» його відразу перенаправить на пошук із наявним доданим словом.

Для того, щоб отримати роль мовознавця, адміністратор має цю роль надати. Якщо адміністратор присутній у системі по замовчуванню, мовознавця ще потрібно призначити. Щоб цього досягти, адміністратор має перейти на сторінку користувача, кого потрібно призначити мовознавцем і натиснути відповідну кнопку (рис. 3.20).

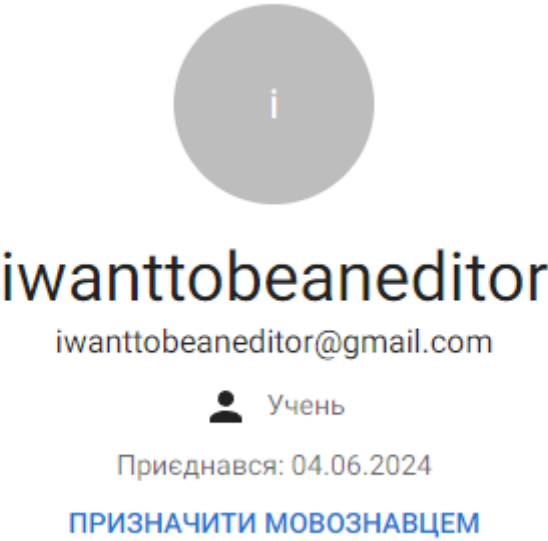


Рисунок 3.20 – Елемент інтерфейсу, відповідальний за надання ролі.

Після чого користувач отримає роль мовознавця, і матиме доступ до усіх його функцій та обов'язків.

Розглянемо роботу з колекціями. Для того, аби створити її, потрібно перейти на свою сторінку та натиснути на кнопку «Нова колекція». Після чого з'явиться відповідне поле, у якому необхідно заповнити певні поля, аби цю колекцію можна було створити (рис. 3.21).

Рисунок 3.21 – Форма додавання колекції

Колекція може бути публічною, та не публічною. Від цього налаштування залежить, чи буде вона відображатися на вашому профілі, якщо на нього зайде якийсь інший користувач. Після того як колекцію було створено, до неї можна додавати будь-які слова з українського словника.

Рисунок 3.22 – Інтерфейс додавання слова до колекцій

Якщо слово присутнє в українському словнику, користувач може обрати з випадającego меню одну зі створених ним колекцій, та натиснути на відповідну кнопку додавання, після чого з'явиться інформаційне повідомлення про успіх цієї дії. До однієї колекції можна додати одне унікальне слово лише один раз, інакше буде помилка.

4. Тестування системи порталу вивчення японської мови

В процесі розробки програмних систем, тестування відіграє важливу роль. Розроблений портал має гарантувати надійний та безперебійний доступ до своїх ресурсів, адже якість процесу навчання, чи модерації словника напряду залежить від стабільності та функціональності написаного коду, з яким взаємодіють мовознавці та учні. Тому тестування порталу вивчення японської мови є ключовим фактором для досягнення поставлених цілей.

Таблиця 3.1

Тестові випадки

Опис	Кроки	Вхідні дані	Очікуваний результат	Фактичний результат	Результат
1	2	3	4	5	6
Успішний пошук слова у словнику	1. Відкрити головну сторінку 2. Ввести запит у поле пошуку 3. Натиснути «Enter»	Запит: 猫	Успішне відображення результатів пошуку	Успішне відображення результатів пошуку	Пройшло
Створення нового акаунту у системі	1. Відкрити головну сторінку 2. Перейти на сторінку створення акаунту 3. Ввести необхідні дані 4. Підтвердити створення	Логін: testuser Пошта: testuser@gmail.com Пароль: testuserpasswd	Повідомлення про успішну операцію та перехід на сторінку входу	Повідомлення про успішну операцію та перехід на сторінку входу	Пройшло
Вхід у існуючий акаунт у системі	1. Відкрити головну сторінку	Логін: testuser	Повідомлення про успішну операцію та	Повідомлення про успішну операцію та	Пройшло

Продовження таблиці 3.1

1	2	3	4	5	6
	2. Перейти на сторінку входу 3. Ввести необхідні дані 4. Підтвердити створення	Пошта: testuser@gmail.com Пароль: testuserpasssword	перехід на головну сторінку	перехід на головну сторінку	
Надання ролі мовознавця	1. Відкрити сторінку бажаного користувача 2. Натиснути на кнопку надання ролі	Користувач: testuser	Повідомлення про надання ролі і відображення відповідних змін	Повідомлення про надання ролі і відображення відповідних змін	Пройшло
Створення колекції користувачем	1. Відкрити сторінку користувача 2. Натиснути на кнопку створення колекції 3. Ввести необхідні дані 4. Натиснути кнопку створення	Назва колекції: testcollection Опис колекції: testdescription Видимість: так	Повідомлення про створення колекції та фактичне її відображення на сторінці	Повідомлення про створення колекції та фактичне її відображення на сторінці	с
Наповнення колекції словами зі словника	1. Виконати пошук по словнику 2. Обрати слово на додавання 3. Обрати колекцію для додавання 4. Додати слово	Слово: 猫 Назва колекції: Назва колекції: testcollection	Повідомлення про успішне додавання та фактичний результат	Повідомлення про успішне додавання та фактичний результат	Пройшло

Продовження таблиці 3.1

1	2	3	4	5	6
Додавання слова у словник будучи звичайним користувачем	1. Виконати пошук по словнику 2. Обрати слово на додавання 3. Додати слово 4. Перевірити правильність автогенерованих значень 5. Підтвердити дію	Слово: 心	Перехід на сторінку пошуку по доданому слову. Саме слово не має відобразитися на сторінці.	Перехід на сторінку пошуку по доданому слову. Саме слово не має відобразитися на сторінці.	Пройшло
Модерація додавання слова у словник мовознавцем	1. Перейти на сторінку модерації 2. Відшукати слова, що потребують модерації 3. Провести модерацію 4. Затвердити слово	Слово: 心	Слово можна редагувати та підтвердити	Слово можна редагувати та підтвердити	Пройшло
Експорт колекції користувача у формат Anki	1. Перейти на сторінку користувача 2. Обрати колекцію для експорту 3. Натиснути кнопку завантаження	Назва колекції: testcollection	Успішне завантаження обраної колекції	Успішне завантаження обраної колекції	Пройшло

Завдяки ретельному тестуванню, що описано у таблиці 3.1, вдалося виявити та виправити недоліки в системі, що істотно зменшує ймовірність виникнення проблем під час її експлуатації. Це, у свою чергу, веде до зростання задоволеності користувачів та гарантує стабільну роботу порталу.

Висновки до третього розділу

У даному розділі описано структуру системи порталу вивчення японської мови за допомогою відповідних діаграм. Відштовхуючись від цього, було повністю налаштовано та розгорнуто систему використовуючи технології Docker для створення, налаштування та запуску контейнерів окремих її частин, та Cloudflare для виводу усієї системи у простір інтернету за її унікальним доменом. Також було детально описано основні частини розробленої системи, як нею користуватися, та які її компоненти за що відповідають.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Вивчення іноземних мов, зокрема японської, в Україні набуло значної актуальності, що обумовлено як особистими, так і професійними потребами. Зростання інтересу до японської мови підкреслює необхідність розробки відповідних інструментів, які б сприяли ефективному навчальному процесу. Однією з основних проблем, з якою стикаються мовознавці, є використання російської системи транслітерації, що не завжди точно передає звучання японських слів в українському правописі.

У процесі розробки порталу особливу увагу було приділено використанню системи транслітерації, запропонованої Коваленком, яка є більш адаптованою для української мови порівняно з російською системою Поліванова. Система Коваленка враховує фонетичні особливості української мови, що дозволяє більш точно і коректно передавати звучання японських термінів. Застосування цієї системи значно підвищує якість і точність транслітерації, забезпечуючи користувачів надійним інструментом для навчання та дослідження. Це також сприяє створенню єдиного стандарту для вивчення японської мови в Україні, що полегшує роботу як учнів, так і викладачів, усуваючи багато проблем і невідповідностей, пов'язаних з використанням застарілих методик.

У рамках даної кваліфікаційної роботи бакалавра було здійснено розробку спеціалізованого порталу, який об'єднує сучасні підходи до передачі японської мови українською. Цей портал стане цінним ресурсом як для учнів, так і для мовознавців, забезпечуючи доступ до актуальних знань. Дослідження процесу вивчення японської мови в українському контексті виявило низку невирішених питань, які потребують подальшого аналізу та вирішення.

Впровадження розробленого порталу сприятиме покращенню якості викладання японської мови, адаптуючи методи транслітерації та транскрипції до українського правопису, що, в свою чергу, допоможе уникнути неточностей і спростить процес навчання. Таким чином, виконана робота робить певний внесок у розвиток вивчення іноземних мов в Україні, а саме японської, та сприяє її інтеграції в український освітній простір на сучасному рівні.

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Збірник наукових праць Національного науково-дослідного інституту українознавства та всесвітньої історії. – К.: ННДІУВІ, 2012. – Т. XXIX. – 370 с.
2. Японсько-українські системи та практичний переклад [Електронний ресурс] – 2022. – Режим доступу до ресурсу: https://typicalamd.blogspot.com/2022/08/blog-post_96.html?m=1.
3. Документація FastAPI [Електронний ресурс] – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/>
4. Документація Uvicorn [Електронний ресурс] – Режим доступу до ресурсу: <https://www.uvicorn.org/>
5. Документація Beanie [Електронний ресурс] – Режим доступу до ресурсу: <https://beanie-odm.dev/>
6. Документація Pydantic [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.pydantic.dev/latest/>
7. Документація Jamdict [Електронний ресурс] – Режим доступу до ресурсу: <https://jamdict.readthedocs.io/en/latest/>
8. Документація Pykakasi [Електронний ресурс] – Режим доступу до ресурсу: <https://pykakasi.readthedocs.io/en/latest/>
9. Документація DeepL [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.deepl.com/docs>
10. Документація NextJS [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org/docs>
11. Документація React [Електронний ресурс] – Режим доступу до ресурсу: <https://react.dev/>
12. Документація JSX [Електронний ресурс] – Режим доступу до ресурсу: <https://legacy.reactjs.org/docs/introducing-jsx.html>
13. Документація Material UI [Електронний ресурс] – Режим доступу до ресурсу: <https://mui.com/material-ui/getting-started/>

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

14. Документація MUI X [Електронний ресурс] – Режим доступу до ресурсу:
<https://mui.com/x/react-data-grid/>
15. Документація MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs/>
16. Документація Docker [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.docker.com/>
17. Документація Cloudflare [Електронний ресурс] – Режим доступу до ресурсу: <https://developers.cloudflare.com/>
18. Introducing FARM Stack - FastAPI, React, and MongoDB [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.mongodb.com/developer/languages/python/farm-stack-fastapi-react-mongodb/>
19. Документація NodeJS [Електронний ресурс] – Режим доступу до ресурсу:
<https://nodejs.org/docs/latest/api/>
20. Документація Genanki [Електронний ресурс] – Режим доступу до ресурсу:
<https://github.com/kerrickstaley/genanki/blob/main/README.md>
21. Документація MDN Web [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/>
22. Документація Python-slugify [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/un33k/python-slugify/blob/master/README.md>
23. Документація Docker Compose [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/compose/>
24. Nihongo-e-na [Електронний ресурс] – Режим доступу до ресурсу:
<https://nihongo-e-na.com/eng>
25. Jisho [Електронний ресурс] – Режим доступу до ресурсу: <https://jisho.org/>
26. Nihonbu [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.nihonbu.club/>
27. Trobmley George. Japanese From Zero! 1 / Trobmley George, Yukari Takenaka. – USA & England: From Zero, LLC, 2014. – 333 с. – (YesJapan Corporation).

					ІІЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

28. Trobmley George. Japanese From Zero! 2 / Trobmley George, Yukari Takenaka. – USA & England: From Zero, LLC, 2014. – 391 с. – (YesJapan Corporation).
29. A Beginner's Guide to Building Your First Next.js Application [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.to/joshhortt/a-beginners-guide-to-building-your-first-nextjs-application-step-by-step-tutorial-glf>
30. Документація Linux [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.linux.org.ua/>

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 24 – ПЗ	Арк.
						72
Змн.	Арк.	№ докум.	Підпис	Дата		

FastAPI

Файл helpers/dicst.py:

```

import re
import deepl
from jamdict import Jamdict
import pykakasi
from config import settings

def hiragana_to_katakana(text):
    katakana_text = []

    for char in text:
        code_point = ord(char)

        if 0x3041 <= code_point <= 0x3096:
            katakana_char = chr(code_point + 0x60)
            katakana_text.append(katakana_char)
        else:
            katakana_text.append(char)

    return ''.join(katakana_text)

def get_kanji_info(query: str | int, deeplapi: bool = False):
    jam = Jamdict()
    kks = pykakasi.kakasi()

    results = []

    kanji_info = jam.lookup(query) if isinstance(
        query, str) else jam.lookup(f'id#{query}')

    if not kanji_info:
        return "Kanji not found in the dictionary."

    for index, entry in enumerate(kanji_info.entries):
        hiragana = entry.kana_forms[0].text
        results.append({
            "idseq": entry.idseq,
            "kanji": entry.kanji_forms[0].text if len(entry.kanji_forms) > 0
        else None,
            "hiragana": hiragana,
            "katakana": entry.kana_forms[-1].text if (len(entry.kana_forms) > 1
and bool(re.match(r'^[\u30A0-\u30FF]+$ ', entry.kana_forms[-1].text))) else
None,
            "romaji": kks.convert(hiragana)[0]['hepburn'],
            "transliteration": hiragana_to_ukrainian(hiragana),
            "kunyomi": kanji_info.chars[0].rm_groups[0].kun_readings[0].value
        if len(kanji_info.chars) > 0 and
        len(kanji_info.chars[0].rm_groups[0].kun_readings) > 0 else None,

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        "onyomi": kanji_info.chars[0].rm_groups[0].on_readings[0].value if
len(kanji_info.chars) > 0 and
len(kanji_info.chars[0].rm_groups[0].on_readings) > 0 else None,
        "en_translation": entry.senses[0].gloss[0].text,
        "ua_translation": None,
    })

    if deeplapi:
        translator = deepl.Translator(settings.DEEPL_API_KEY)
        results[0]["ua_translation"] = translator.translate_text(
            results[0]["en_translation"], source_lang='EN',
target_lang='UK').text

        if results[0]['katakana'] is None:
            results[0]['katakana'] = hiragana_to_katakana(
                results[0]['hiragana'])

    return results

def hiragana_to_ukrainian(query, info=False):
    kks = pykakasi.kakasi()
    force_hira = ''.join([f['hira'] for f in kks.convert(query)])
    htudict = hiragana_full
    ukrainian_text = ""
    i = 0
    while i < len(force_hira):
        if i + 1 < len(force_hira) and force_hira[i:i+2] in htudict:
            ukrainian_text += htudict[force_hira[i:i+2]]
            i += 2
        else:
            ukrainian_text += htudict.get(force_hira[i], force_hira[i])
            i += 1
    if info:
        return {
            "ukrainian": ukrainian_text,
            "hiragana": force_hira,
            "romaji": ''.join([f['hepburn'] for f in kks.convert(query)]),
        }
    return ukrainian_text

# Hiragana
hiragana_basic = {
    "あ": "a", "い": "i", "う": "y", "え": "e", "お": "o",
    "か": "ka", "き": "ki", "く": "ky", "け": "ke", "こ": "ko",
    "さ": "sa", "し": "shi", "す": "sy", "せ": "se", "そ": "so",
    "た": "ta", "ち": "chi", "つ": "cy", "て": "te", "と": "to",
    "な": "na", "に": "ni", "ぬ": "ny", "ね": "ne", "の": "no",
    "は": "ha", "ひ": "hi", "ふ": "fu", "へ": "he", "ほ": "ho",
    "ま": "ma", "み": "mi", "む": "my", "め": "me", "も": "mo",
    "や": "ya", "ゆ": "yu", "よ": "yo",
    "ら": "ra", "り": "ri", "る": "ry", "れ": "re", "ろ": "ro",

```

```

    "わ": "ba", "を": "o", "ん": "n",
}

hiragana_combinations = {
    "きゃ": "кя", "きゅ": "кю", "きょ": "кьо",
    "しゃ": "шя", "しゅ": "шю", "しょ": "шьо",
    "ちゃ": "чя", "ちゅ": "чю", "ちょ": "чьо",
    "にゃ": "ня", "にゅ": "ню", "にょ": "ньо",
    "ひゃ": "хя", "ひゅ": "хю", "ひょ": "хьо",
    "みゃ": "мя", "みゅ": "мю", "みょ": "мьо",
    "りゃ": "ря", "りゅ": "рю", "りょ": "рьо",
}

hiragana_kuten = {
    "が": "га", "ぎ": "gi", "ぐ": "gy", "げ": "ge", "ご": "go",
    "ざ": "дза", "じ": "джи", "ず": "дзу", "ぜ": "дзе", "ぞ": "дзо",
    "だ": "да", "ぢ": "джи", "づ": "дзу", "で": "де", "ど": "до",
    "ば": "ба", "び": "bi", "ぶ": "by", "べ": "be", "ぼ": "bo",
    "ぱ": "па", "ぴ": "pi", "ぷ": "py", "ぺ": "pe", "ぽ": "po",
}

hiragana_kuten_combinations = {
    "ぎゃ": "гя", "ぎゅ": "гю", "ぎょ": "гьо",
    "じゃ": "джя", "じゅ": "джю", "じょ": "джьо",
    "ぢゃ": "джя", "ぢゅ": "джю", "ぢょ": "джьо",
    "びゃ": "бя", "びゅ": "бю", "びょ": "бьо",
    "ぴゃ": "пя", "ぴゅ": "пю", "ぴょ": "пьо",
}

hiragana_full = {**hiragana_basic, **hiragana_combinations,
                  **hiragana_kuten, **hiragana_kuten_combinations}

hiragana_basic_stats = {k[0]: {"right": 0, "wrong": 0}
                        for k in hiragana_basic.items()}

hiragana_kuten_stats = {k[0]: {"right": 0, "wrong": 0}
                        for k in hiragana_kuten.items()}

hiragana_kuten_combinations_stats = {k[0]: {"right": 0, "wrong": 0}
                                      for k in {**hiragana_combinations,
**hiragana_kuten_combinations}.items()}

# Katakana
katakana_basic = {
    "ア": "a", "イ": "i", "ウ": "y", "エ": "e", "オ": "o",
    "カ": "ka", "キ": "ki", "ク": "ky", "ケ": "ke", "コ": "ko",
    "サ": "sa", "シ": "shi", "ス": "cy", "セ": "ce", "ソ": "co",
    "タ": "ta", "チ": "chi", "ツ": "cy", "テ": "te", "ト": "to",
    "ナ": "na", "ニ": "ni", "ヌ": "ny", "ネ": "ne", "ノ": "no",
}

```

```

    "ハ": "ха", "ヒ": "хи", "フ": "фу", "ヘ": "хе", "ホ": "хо",
    "マ": "ма", "ミ": "ми", "ム": "му", "メ": "ме", "モ": "мо",
    "ヤ": "я", "ユ": "ю", "ヨ": "йо",
    "ラ": "ра", "リ": "ри", "ル": "ру", "レ": "ре", "ロ": "ро",
    "ワ": "ва", "ヲ": "о", "ン": "н",
}

katakana_combinations = {
    "キャ": "кя", "キュ": "кю", "キヨ": "кьо",
    "シャ": "шя", "シュ": "шю", "シヨ": "шьо",
    "チャ": "чя", "チュ": "чю", "チヨ": "чьо",
    "ニャ": "ня", "ニユ": "ню", "ニヨ": "ньо",
    "ヒャ": "хя", "ヒユ": "хю", "ヒヨ": "хьо",
    "ミャ": "мя", "ミユ": "мю", "ミヨ": "мьо",
    "リャ": "ря", "リュ": "рю", "リヨ": "рьо",
}

katakana_kuten = {
    "ガ": "га", "ギ": "gi", "グ": "гу", "ゲ": "ге", "ゴ": "го",
    "ザ": "дза", "ジ": "джи", "ズ": "дзу", "ゼ": "дзе", "ゾ": "дзо",
    "ダ": "да", "ヂ": "джи", "ヅ": "дзу", "デ": "де", "ド": "до",
    "バ": "ба", "ビ": "bi", "ブ": "бу", "ベ": "бе", "ボ": "бо",
    "パ": "па", "ピ": "pi", "プ": "пу", "ペ": "пе", "ポ": "по",
}

katakana_kuten_combinations = {
    "ギャ": "гя", "ギユ": "гю", "ギヨ": "гьо",
    "ジヤ": "джя", "ジユ": "джю", "ジヨ": "джьо",
    "ヂャ": "джя", "ヂユ": "джю", "ヂヨ": "джьо",
    "ビャ": "бя", "ビユ": "бю", "ビヨ": "бьо",
    "ピャ": "пя", "ピユ": "пю", "ピヨ": "пьо",
}

katakana_full = {**katakana_basic, **katakana_combinations,
                  **katakana_kuten, **katakana_kuten_combinations}

katakana_stats = {k[0]: {"right": 0, "wrong": 0}
                   for k in katakana_full.items()}

katakana_basic_stats = {k[0]: {"right": 0, "wrong": 0}
                        for k in katakana_basic.items()}

katakana_kuten_stats = {k[0]: {"right": 0, "wrong": 0}
                        for k in katakana_kuten.items()}

katakana_kuten_combinations_stats = {k[0]: {"right": 0, "wrong": 0}
                                      for k in {**katakana_combinations,
                                      **katakana_kuten_combinations}.items()}

```

Файл helpers/quizes.py:

```
import random

def generate_quiz(questions_dict, quiz_size=10):
    if quiz_size > len(questions_dict):
        quiz_size = len(questions_dict)

    keys = list(questions_dict.keys())
    random.shuffle(keys)
    quiz_keys = keys[:quiz_size]

    quiz = []
    for key in quiz_keys:
        correct_answer = questions_dict[key]
        all_answers = list(questions_dict.values())
        wrong_answers = random.sample(
            [ans for ans in all_answers if ans != correct_answer], 3)

        options = wrong_answers + [correct_answer]
        random.shuffle(options)

        unique_options = list(set(options))
        while len(unique_options) < 4:
            remaining_wrong_answers = [
                ans for ans in all_answers if ans != correct_answer and ans not
in unique_options]
            if remaining_wrong_answers:
                new_answer = random.choice(remaining_wrong_answers)
                unique_options.append(new_answer)
            unique_options = list(set(unique_options))

        random.shuffle(unique_options)
        quiz.append((key, correct_answer, unique_options))

    return quiz
```

Файл routers/dictionary.py:

```
from datetime import datetime
from typing import List
from uuid import UUID
from beanie.operators import RegEx
from beanie.odm.operators.find.logical import Or
from fastapi import APIRouter, Depends, HTTPException
from auth.oauth2 import get_current_user, linguist_check
from schemas.dictionary import (
    DictionaryBase, DictionaryMassSearch, DictionaryCreate,
    DictionaryOptimized)
from models.user import User
from models.dictionary import Dictionary
from helpers.dicts import get_kanji_info, hiragana_to_ukrainian
```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						77
Змн.	Арк.	№ докум.	Підпис	Дата		

```

router = APIRouter(
    prefix="/dictionary",
    tags=["Dictionary"]
)

@router.post("/trasliteration")
async def transliteration(query: str):
    return hiragana_to_ukrainian(query, info=True)

@router.post("/")
async def create_entry(request: DictionaryCreate, current_user: User = Depends(get_current_user)):
    dictionary = Dictionary(
        idseq=request.idseq,
        kanji=request.kanji,
        hiragana=request.hiragana,
        katakana=request.katakana,
        romaji=request.romaji,
        transliteration=request.transliteration,
        kunyomi=request.kunyomi,
        onyomi=request.onyomi,
        en_translation=request.en_translation,
        ua_translation=request.ua_translation,
        created_by=current_user.id,
        approved=current_user.role.name == "linguist",
        approved_by=current_user if current_user.role.name == "linguist" else
None
    )

    try:
        await dictionary.save()
        return dictionary
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.get("/entry/{uuid}", response_model=DictionaryBase)
async def get_entry_by_uuid(uuid: UUID):
    try:
        dic = await Dictionary.find_one(Dictionary.uuid == uuid)
        return dic
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.post("/entry/{uuid}")
async def update_entry_by_uuid(request: DictionaryCreate, uuid: UUID,
current_user: User = Depends(get_current_user)):
    try:

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        dic = await Dictionary.find_one(Dictionary.uuid == uuid)
        dic.kanji = request.kanji
        dic.hiragana = request.hiragana
        dic.katakana = request.katakana
        dic.romaji = request.romaji
        dic.transliteration = request.transliteration
        dic.kunyomi = request.kunyomi
        dic.onyomi = request.onyomi
        dic.en_translation = request.en_translation
        dic.ua_translation = request.ua_translation
        dic.updated_at = datetime.now()
        await dic.save()
        return dic
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.patch("/entry/{uuid}", response_model=DictionaryBase)
async def approve_entry(uuid: UUID, current_user: User =
    Depends(get_current_user), linguist_check=Depends(linguist_check)):
    try:
        dic = await Dictionary.find_one(Dictionary.uuid == uuid)
        dic.approved_by = current_user
        dic.approved = True
        await dic.save()
        return dic
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.delete("/entry/{uuid}", response_model=DictionaryBase)
async def delete_entry_by_uuid(uuid: UUID):
    try:
        dic = await Dictionary.find_one(Dictionary.uuid == uuid)
        await dic.delete()
        return dic
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.get("/idseq/{idseq}", response_model=DictionaryBase)
async def get_entry_by_idseq(idseq: int):
    try:
        dic = await Dictionary.find_one(Dictionary.idseq == idseq)
        return dic
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

```

					ІІЗ.КР.Б – 121 – 24 – ІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		79

```

@router.get("/all", response_model=List[DictionaryOptimized])
async def get_all_entries():
    try:
        dic = await Dictionary.find_all(fetch_links=True,
nesting_depths_per_field={"approved_by": 2}).to_list()

        return dic
    except Exception as e:
        raise HTTPException(
            status_code=400, detail=f"Error: {e}"
        )

@router.get("/search/{query}", response_model=DictionaryMassSearch)
async def search(query: str):
    dic = await Dictionary.find_many(Or(
        RegEx(Dictionary.kanji, query, 'i'),
        RegEx(Dictionary.hiragana, query, 'i'),
        RegEx(Dictionary.katakana, query, 'i'),
        RegEx(Dictionary.ua_translation, query, 'i'),
        RegEx(Dictionary.romaji, query, 'i'),
        RegEx(Dictionary.transliteration, query, 'i'),
    )).to_list()

    dic_base_list = [DictionaryBase(**d.model_dump())
        for d in dic if d.approved]
    local = get_kanji_info(query)

    filtered_local = [l for l in local if l['idseq']
        not in [d.idseq for d in dic_base_list]]

    results = DictionaryMassSearch(
        en_source=filtered_local, ua_source=dic_base_list)

    return results

@router.get("/jamdict/{idseq}")
def get_jamdict(idseq: int):
    result = get_kanji_info(idseq, deeplapi=True)[0]

    return result

```

Файл schemas/user.py:

```

from datetime import datetime
from typing import Optional, List
from beanie import PydanticObjectId
from pydantic import BaseModel, EmailStr, Field
from uuid import UUID
from config import settings
from schemas.role import RoleBase, RoleUpdate
from schemas.collection import CollectionBase

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

from schemas.stats import Stats

class UserBase(BaseModel):
    """
    Shared User properties for registration.
    """
    email: str
    username: str
    password: str
    avatar_url: Optional[str] = Field(
        default=f'{settings.APP_URL}/static/images/blank_avatar.jpg')

class UserPublic(BaseModel):
    """
    Shared User properties for public usage.
    """
    email: EmailStr
    username: str
    avatar_url: str
    role: RoleUpdate
    created_at: datetime
    collections: List[CollectionBase]

class UserUpdate(BaseModel):
    """
    Shared User properties for updating.
    """
    email: Optional[EmailStr] = None
    username: Optional[str] = None
    password: Optional[str] = None
    avatar_url: Optional[str] = None

class UserPrivate(BaseModel):
    """
    Shared User properties for private usage.
    """
    id: PydanticObjectId
    uuid: UUID
    role: RoleBase
    email: str
    username: str
    collections: List[CollectionBase]
    avatar_url: str
    created_at: datetime

class UserStats(BaseModel):
    """
    Shared User properties for stats.
    """
    stats: Stats

```

NextJS

Файл components/Table.js:

```

"use client";
import {
  GridRowModes,
  DataGrid,
  GridToolbar,
  GridActionsCellItem,
  GridRowEditStopReasons
} from '@mui/x-data-grid';
import {
  Edit as EditIcon,
  Save as SaveIcon,
  Cancel as CancelIcon
} from '@mui/icons-material';
import { Box, Button, TextField } from '@mui/material';
import { ukUA } from '@mui/x-data-grid/locales';
import { getCookie } from '@utils/cookies';
import { useRouter } from 'next/navigation';
import { useState } from 'react';
import Link from 'next/link';
import { useSearchParams } from 'next/navigation'

const url = `${process.env.NEXT_PUBLIC_API_URL}/dictionary/entry`;

function ApproveButton({ uuid }) {

  const router = useRouter();

  const handleApprove = async () => {
    const options = {
      method: 'PATCH',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
        'Authorization': 'Bearer ' + await getCookie('token')
      },
    };

    const response = await fetch(`${url}/${uuid}`, options);

    router.refresh();
  }

  return <Button onClick={handleApprove} color={'success'}>Схвалити</Button>
}

function DeleteButton({ uuid }) {

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		82

```

const router = useRouter();

const handleDelete = async () => {
  const options = {
    method: 'DELETE',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json',
      'Authorization': 'Bearer ' + await getCookie('token')
    },
  };

  const response = await fetch(`${url}/${uuid}`, options);

  router.refresh();
}

return <Button onClick={handleDelete} color={'error'}>Видалити</Button>
}

export default function Table({ rowsInit }) {
  const searchParams = useSearchParams();
  const router = useRouter();
  const [rows, setRows] = useState([]);
  const [rowModesModel, setRowModesModel] = useState({});

  const handleRowEditStop = (params, event) => {
    if (params.reason === GridRowEditStopReasons.rowFocusOut) {
      event.defaultMuiPrevented = true;
    }
  };

  const handleEditClick = (id) => () => {
    setRowModesModel({ ...rowModesModel, [id]: { mode:
GridRowModes.Edit } });
  };

  const handleSaveClick = (id) => () => {
    setRowModesModel({ ...rowModesModel, [id]: { mode:
GridRowModes.View } });
  };

  const handleCancelClick = (id) => () => {
    setRowModesModel({
      ...rowModesModel,
      [id]: { mode: GridRowModes.View, ignoreModifications: true },
    });
  };

  const editedRow = rows.find((row) => row.id === id);
  if (editedRow?.isNew) {
    setRows(rows.filter((row) => row.id !== id));
  }
}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						83
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    });

    const processRowUpdate = async (newRow) => {
      const updatedRow = { ...newRow, isNew: false };
      setRows(rows.map((row) => (row.id === newRow.id ? updatedRow : row)));

      const unallowed = ['approved_by', 'created_by', 'updated_at', 'isNew',
        'approved', 'uuid']

      const filteredRow = Object.keys(updatedRow)
        .filter(key => !unallowed.includes(key))
        .reduce((obj, key) => {
          obj[key] = updatedRow[key];
          return obj;
        }, {});

      const response = await fetch(`${url}/${updatedRow.uuid}`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': 'Bearer ' + await getCookie('token'),
        },
        body: JSON.stringify(filteredRow),
      });

      if (!response.ok) {
        throw new Error('Network response was not ok');
      }

      router.refresh();

      return updatedRow;
    });

    const handleRowModesModelChange = (newRowModesModel) => {
      setRowModesModel(newRowModesModel);
    };

    const columns = [
      { field: 'idseq', headerName: 'IDSEQ', width: 80 },
      { field: 'kanji', headerName: 'Канджі', editable: true, width: 70 },
      { field: 'hiragana', headerName: 'Хіраґана', editable: true, width:
100 },
      { field: 'katakana', headerName: 'Катакана', editable: true, width:
100 },
      { field: 'romaji', headerName: 'Ромаджі', editable: true, width: 130 },
      { field: 'transliteration', headerName: 'Транслітерація', editable:
true, width: 130 },
      { field: 'kunyomi', headerName: 'Кунйомі', editable: true, width:
130 },
      { field: 'onyomi', headerName: 'Онйомі', editable: true, width: 130 },
    ]

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

					ІІЗ.КР.Б – 121 – 24 – ІЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        color="inherit"
      />,
    ],
  },
},
{ field: 'approve_button', headerName: '', width: 100, renderCell:
(params) => <ApproveButton uuid={params.row.uuid}></ApproveButton> },
{ field: 'delete_button', headerName: '', width: 100, renderCell:
(params) => <DeleteButton uuid={params.row.uuid}></DeleteButton> },
];

return (
  <Box>
    <DataGrid
      rows={rowsInit}
      columns={columns}
      initialState={{
        pagination: {
          paginationModel: { page: 0, pageSize: 10 },
        },
        columns: {
          columnVisibilityModel: {
            uuid: false,
          }
        },
        filter: {
          filterModel: {
            items: [{ field: 'idseq', operator: 'equals',
value: searchParams.get('idseq') }]
          },
        },
      }}
      pageSizeOptions={[10, 15]}
      editMode="row"
      rowModesModel={rowModesModel}
      getRowId={(row) => row.uuid}
      onRowModesModelChange={handleRowModesModelChange}
      onRowEditStop={handleRowEditStop}
      processRowUpdate={processRowUpdate}
      slots={{ toolbar: GridToolbar }}
      slotProps={{
        toolbar: { setRows, setRowModesModel, showQuickFilter:
true, },
      }}
      localeText={ukUA.components.MuiDataGrid.defaultProps.localeText
}

      disableRowSelectionOnClick
    />
  </Box>
)
}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						86
Змн.	Арк.	№ докум.	Підпис	Дата		

Файл components/DictionarySearch.js:

```
"use client";
import { useState, useEffect } from 'react';
import StyledLink from '@components/StyledLink';
import {
  Button,
  Box,
  Typography,
  Card,
  CardContent,
  Paper,
  Grid,
  Container,
  FormControl,
  InputLabel,
  Select,
  MenuItem,
  IconButton,
  Collapse,
} from '@mui/material';
import { ExpandLess, ExpandMore } from '@mui/icons-material';
import { getCookie } from '@utils/cookies';
import { useNotification } from '../../context/NotificationContext';

const url = process.env.NEXT_PUBLIC_API_URL;

const KanjiCard = ({ entry, currentUser, ua }) => {
  const showNotification = useNotification();
  const [collection, setCollection] = useState(currentUser?.collections[0] ?
currentUser?.collections[0].uuid : 'none');

  const handleCollectionChange = (e) => {
    setCollection(e.target.value);
  };

  const submitWordToCollection = async (wordUuid) => {
    try {
      const response = await
fetch(`${url}/collection/add/${collection}?word_uuid=${wordUuid}`, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer ' + await getCookie('token'),
  }
});

      if (!response.ok) {
        showNotification('Помилка додавання до колекції', 'error');
      } else {
        showNotification('Успішно додано до колекції', 'success');
      }
    }
  };
}
```

					ІІЗ.КР.Б – 121 – 24 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

    } catch (err) {
      console.log(err.message || 'An error occurred');
    }
  }

  return (
    <Paper sx={{ p: 2, mb: 2 }}>
      <Grid container spacing={1}>
        <Grid item xs={1}>
          <StyledLink href={` /search/${entry.kanji}`}><Typography
variant="h2">{entry.kanji ? entry.kanji :
entry.hiragana}</Typography></StyledLink>
          <span>{entry.idseq}</span>
        </Grid>
        <Grid item xs={6}>
          <Typography variant="body1"><strong>Хіраґана:</strong>
{entry.hiragana}</Typography>
          {entry.katakana ? <Typography
variant="body1"><strong>Катакана:</strong> {entry.katakana}</Typography> :
undefined}
          <Typography variant="body1"><strong>Ромаджі:</strong>
{entry.romaji}</Typography>
          <Typography
variant="body1"><strong>Транслітерація:</strong>
{entry.transliteration}</Typography>
          <Typography variant="body1"><strong>Кунйомі:</strong>
{entry.kunyomi}</Typography>
          <Typography variant="body1"><strong>Онйомі:</strong>
{entry.onyomi}</Typography>
          {ua ? <Typography
variant="body1"><strong>Переклад:</strong>
{entry.ua_translation}</Typography> : <Typography
variant="body1"><strong>Анґлійський переклад:</strong>
{entry.en_translation}</Typography>}
        </Grid>
        {ua ?
          <Box sx={{ mt: 2, display: 'flex', alignItems: 'top' }}>
            {currentUser ? <FormControl variant="outlined"
sx={{ minWidth: 220, mr: 2 }}>
              <InputLabel id="collection-
label">Колекції</InputLabel>
              <Select
                labelId="collection-label"
                id="collection-select"
                label="Collection"
                value={collection}
                onChange={handleCollectionChange}
              >
                <MenuItem value="none"><em>None</em></MenuItem>
                {currentUser?.collections.map((collection,
index) => (

```

```

                                <MenuItem key={index}
value={collection.uuid}>{collection.name}</MenuItem>
                                )))
                                </Select>
                                </FormControl> : undefined}
                                <Box
                                sx={{
                                display: 'flex',
                                flexDirection: 'column',
                                gap: 1,
                                }}
                                >
                                {currentUser ? <Button variant="contained"
color="secondary" onClick={() => submitWordToCollection(entry.uuid)}>Додати до
колекції</Button> : undefined}
                                {currentUser?.role.name === 'linguist' ? <Button
variant="contained" color="primary"
href={` /dictionary/edit?idseq=${entry.idseq}`}>Редагувати</Button> : undefined}
                                </Box>
                                </Box>
                                :
                                <Grid item xs={5} sx={{ display: 'flex', justifyContent:
'flex-end', alignItems: 'flex-end' }}>
                                {currentUser ? <Button variant="contained"
color="primary" href={` /dictionary/${entry.idseq}`}>Додати до
словника</Button> : undefined}
                                </Grid>
                                }
                                </Grid>
                                </Paper>
                                </Paper>
                                );
                                };

export default function DictionarySearch({ currentUser, search }) {
  const [data, setData] = useState(null);
  const [showEnSource, setShowEnSource] = useState(false);
  const [showUaSource, setShowUaSource] = useState(true);

  const fetchData = async () => {
    try {
      const response = await fetch(`${url}/dictionary/search/${search}`,
{
      method: 'GET',
      headers: {
        'Accept': 'application/json',
      },
    });

    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
  }
}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						89
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const result = await response.json();

if (result.ua_source.length === 0) {
  setShowUaSource(false);
  setShowEnSource(true);
}

if (result.en_source.length === 0) {
  setShowEnSource(false);
  setShowUaSource(true);
}

setData(result);
} catch (err) {
  console.log(err.message || 'An error occurred');
}
};

useEffect(() => {
  fetchData();
}, []);

return (
  data ?
    <Container >
      <Box sx={{
        marginTop: 3, marginBottom: 3
      }} maxWidth="xl">
        <Card variant="outlined" sx={{ marginBottom: 3 }}>
          <CardContent>
            <Box onClick={() =>
              setShowUaSource(!showUaSource)}>
              <Typography variant="h3" gutterBottom>
                Український словник
                <IconButton>
                  {showUaSource ? <ExpandLess /> :
                    <ExpandMore />}
              </IconButton>
            </Typography>
          </Box>
          <Collapse in={showUaSource}>
            {
              data.ua_source.map((item, index) => (
                <KanjiCard entry={item}
                  currentUser={currentUser} ua={true} key={index} />
              ))
            }
          </Collapse>
        </CardContent>
      </Card>
    <Card variant="outlined">
      <CardContent>

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		90

```

        <Box onClick={() =>
setShowEnSource(!showEnSource)}>
        <Typography variant="h3" gutterBottom>
            Англійський словник
        <IconButton>
            {showEnSource ? <ExpandLess /> :
<ExpandMore />}
        </IconButton>
        </Typography>
    </Box>
    <Collapse in={showEnSource}>
        {
            data.en_source.map((item, index) => (
                <KanjiCard entry={item}
currentUser={currentUser} key={index} />
            ))
        }
    </Collapse>
</CardContent>
</Card>
</Box >
</Container> : undefined
    );
};

```

Файл app/user/[name]/page.js:

```

import { Suspense } from 'react';
import UserPage from '@components/UserPage';
import { getCurrentUser, getByUserName } from '@utils/user';

export const fetchCache = 'only-no-store';

async function getUser(username) {
    const user = await getByUserName(username);
    const currentUser = await getCurrentUser();
    const current = currentUser?.username === user?.username

    if (current) {
        return { ...currentUser, current };
    }

    return { ...user, current, currentUser: currentUser };
}

export default async function User({ params }) {

    const user = await getUser(params.name);

    return (
        <Suspense fallback={<div>Loading...</div>}>
            <UserPage user={user} />
        </Suspense>
    );
}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

    </Suspense>
  );
}

```

Файл app/dictionary/[id]/page.js:

```

"use client";
import { useRouter } from 'next/navigation';
import { useState, useEffect } from 'react';
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Box from '@mui/material/Box';
import Container from '@mui/material/Container';
import { getCookie } from '@/utils/cookies';

const url = process.env.NEXT_PUBLIC_API_URL;

export default function AddToDictionary({ params }) {
  const router = useRouter();
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await
fetch(`${url}/dictionary/jamdict/${params.id}`, {
  method: 'GET',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json',
  }
});

        if (!response.ok) {
          throw new Error('Network response was not ok');
        }

        const result = await response.json();
        setData(result);
      } catch (err) {
        console.log(err.message || 'An error occurred');
      }
    };

    fetchData();
  }, []);

  const submitData = async (data) => {
    const sUrl = `${url}/dictionary`;
    console.log(sUrl);
    try {
      const response = await fetch(sUrl, {

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-Type': 'application/json',
          'Authorization': 'Bearer ' + await getCookie('token'),
        },
        body: JSON.stringify(data),
      });

      if (!response.ok) {
        throw new Error('Network response was not ok');
      }

      const result = await response.json();
      router.push(`/search/${result.kanji ? result.kanji :
result.hiragana}`);
    } catch (err) {
      console.log(err.message || 'An error occurred');
    }
  });

  const handleChange = (event) => {
    const { name, value } = event.target;
    setData((prevData) => ({
      ...prevData,
      [name]: value,
    }));
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    submitData({ ...data })
  };

  return (
    <>
    {data &&
      <Container >
        <Box
          sx={{
            marginTop: 8,
            display: 'flex',
            flexDirection: 'column',
            alignItems: 'center',
          }}
        >
        <form onSubmit={handleSubmit}>
          <TextField
            style={{ marginBottom: 10 }}
            label="Канджі"
            name="kanji"
            value={data.kanji}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        onChange={handleChange}
        fullWidth
      />
      <TextField
        style={{ marginBottom: 10 }}
        label="Хіраґана"
        name="hiragana"
        value={data.hiragana}
        onChange={handleChange}
        fullWidth
        required
      />
      <TextField
        style={{ marginBottom: 10 }}
        label="Катакана"
        name="katakana"
        value={data.katakana}
        onChange={handleChange}
        fullWidth
      />
      <TextField
        style={{ marginBottom: 10 }}
        label="Кунне читання"
        name="kunyomi"
        value={data.kunyomi}
        onChange={handleChange}
        fullWidth
      />
      <TextField
        style={{ marginBottom: 10 }}
        label="Онне читання"
        name="onyomi"
        value={data.onyomi}
        onChange={handleChange}
        fullWidth
      />
      <TextField
        style={{ marginBottom: 10 }}
        label="Англійський переклад"
        name="en_translation"
        value={data.en_translation}
        onChange={handleChange}
        fullWidth
      />
      <TextField
        style={{ marginBottom: 10 }}
        label="Переклад"
        name="ua_translation"
        value={data.ua_translation}
        onChange={handleChange}
        fullWidth
        required

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
						94
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        />
        <TextField
            style={{ marginBottom: 10 }}
            label="Ромаджі"
            name="romanization"
            value={data.romaji}
            onChange={handleChange}
            fullWidth
            required
        />
        <TextField
            style={{ marginBottom: 10 }}
            label="Транслітерація"
            name="transliteration"
            value={data.transliteration}
            onChange={handleChange}
            fullWidth
            required
        />
        <Button type="submit" variant="contained"
color="primary" >
            Додати
        </Button>
    </form>
</Box>
</Container>
    }
  </>
);
}

```

					ІІЗ.КР.Б – 121 – 24 – ІІЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		95