

Національний університет «Одеська політехніка»  
Інститут комп'ютерних систем  
Кафедра інформаційних систем

## КУРСОВА РОБОТА

з дисципліни «Об'єктно-орієнтоване програмування»

Тема «Веб-сервіс для вивчення англійських слів»

Студента   2   курсу   AI-221   групи

Спеціальності 122 – «Комп'ютерні науки»

                    Семиволос Л.Ю.                    

(прізвище та ініціали)

Керівник доцент, к.т.н. Годовиченко М.А.

\_\_\_\_\_  
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_

Кількість балів: \_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_

Члени комісії

_____	_____
(підпис)	(прізвище та ініціали)
_____	_____
(підпис)	(прізвище та ініціали)
_____	_____
(підпис)	(прізвище та ініціали)

м. Одеса – 2024 рік

Національний університет «Одеська політехніка»  
Інститут комп'ютерних систем  
Кафедра інформаційних систем

**ЗАВДАННЯ**  
**НА КУРСОВУ РОБОТУ**

студенту Семиволоса Лева Юрійовича

група AI-221

1. Тема роботи  
«Веб-сервіс для вивчення англійських слів»

2. Термін здачі студентом закінченої роботи 09.06.2024

3. Початкові дані до проекту (роботи)

Програма повинна виконувати: реєстрацію та авторизацію користувачів, автентифікація користувачів за логіном та паролем, надання користувачам карток зі словами, реченнями, перекладом та синонімами, використання інтервального повторення для ефективного запам'ятовування слів, можливість перевірки знання слова шляхом введення відповіді, збереження прогресу користувача з вивчення кожної картки, відображення загальної кількості вивчених слів, створення, читання, оновлення та видалення карток. Використані технології: Java Spring Boot, JPA, Spring MVC, Spring Security, AOP, Thymeleaf, Lombok, PostgreSQL, RESTful API, Maven.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити)

Вступ. Проектування і розробка веб-шару (REST-запити). Проектування і розробка сервісного шару. Проектування і розробка шару сховища. Проектування і розробка журналювання. Проектування і розробка безпеки. Інструкція користувача. Висновки.

Завдання видано \_\_\_\_\_

\_\_\_\_\_  
(підпис викладача)

Завдання прийнято до виконання \_\_\_\_\_

\_\_\_\_\_  
(підпис студента)

## АНОТАЦІЯ

Розглянуто актуальність розробки веб-сервісів для вивчення іноземних мов, зокрема з використанням карток та інтервального повторення.

Запропоновано багатоварову архітектуру веб-сервісу для вивчення англійських слів, що включає веб-шар, сервісний шар, шар сховища, журналювання та безпеки.

Розроблено REST API для взаємодії з веб-сервісом, сервіси для реалізації бізнес-логіки, Entity-класи та репозиторії для роботи з базою даних.

Реалізовано механізми автентифікації та авторизації користувачів, захист від CSRF та XSS атак, шифрування паролів, логування подій та розгортання додатку.

## ABSTRACT

The article considers the relevance of developing web services for learning foreign languages, in particular, using flashcards and interval repetition.

A multilayer architecture of a web service for learning English words is proposed, including a web layer, a service layer, a storage layer, a logging layer, and a security layer.

REST APIs for interacting with the web service, services for implementing business logic, Entity classes and repositories for working with the database have been developed.

We have implemented mechanisms for user authentication and authorisation, protection against CSRF and XSS attacks, password encryption, event logging and application deployment.

## ЗМІСТ

ВСТУП.....	6
1 ПРОЕКТУВАННЯ І РОЗРОБКА ВЕБ-ШАРУ .....	9
1.1 Теоретичні відомості. ....	9
1.2 Аналіз вимог та проектування REST контролерів.....	10
2 ПРОЕКТУВАННЯ І РОЗРОБКА СЕРВІСНОГО ШАРУ .....	13
2.1. Сервіс навчання (LearningService) .....	13
2.2. Сервіс статистики (StatsService) .....	15
2.3. Сервіс карток (CardService).....	15
3 ПРОЕКТУВАННЯ І РОЗРОБКА ВЕБ-ШАРУ .....	19
3.1 Сутності та їх атрибути. ....	19
3.2 Spring Data JPA .....	21
4 ПРОЕКТУВАННЯ І РОЗРОБКА ЖУРНАЛЮВАННЯ.....	23
4.1 Теоретичні відомості об АОР. ....	23
4.2 Визначення аспектів для журналювання.....	24
4.3 Налаштування журналювання з використанням SLF4J та Logback	25
5 ПРОЕКТУВАННЯ І РОЗРОБКА БЕЗПЕКИ .....	27
5.1 Теоретичні відомості о Spring Security. ....	27
5.2 Налаштування Spring Security .....	27
6 ТЕСТУВАННЯ ДОДАТКУ .....	31
6.1. Цілі та завдання тестування .....	31
6.2. Методи та інструменти тестування .....	31
6.3. Тестування API .....	32
6.4. Тестування інтерфейсу користувача.....	39

	5
6.5. Висновки .....	39
7 ІНСТРУКЦІЯ КОРИСТУВАЧА .....	41
7.1. Головна сторінка .....	41
7.2. Реєстрація .....	41
7.3. Логін .....	42
7.4. Навчання .....	43
7.5. Статистика.....	45
7.6. Вихід з системи.....	45
ВИСНОВКИ .....	47

## ВСТУП

Метою курсової роботи є закріплення та поглиблення знань, отриманих студентами в курсі "Розробка веб-додатків", розвиток навичок проектування та розробки багатоварштових веб-сервісів, оволодіння сучасними технологіями та інструментами для створення безпечних та ефективних веб-додатків.

Динамічні веб-сервіси, побудовані на основі RESTful API, є невід'ємною частиною сучасного Інтернету та застосовуються в різноманітних сферах, таких як електронна комерція, соціальні мережі, онлайн-освіта та багато інших. Веб-сервіси дозволяють різним системам та додаткам взаємодіяти між собою, обмінюючись даними та функціональністю.

Однією з ключових переваг RESTful API є їхня гнучкість та масштабованість. RESTful API використовують стандартні HTTP-методи (GET, POST, PUT, DELETE) для виконання операцій над ресурсами, що робить їх простими у використанні та інтеграції з різними платформами. Крім того, веб-сервіси можуть бути легко розширені та адаптовані до нових вимог, що робить їх ідеальним рішенням для сучасних динамічних систем.

У даній курсовій роботі розглядається розробка веб-сервісу для вивчення англійських слів. Сервіс побудовано за багатоваршовою архітектурою, що включає:

- Веб-шар: RESTful API для взаємодії з клієнтськими додатками.
- Сервісний шар: бізнес-логіка додатку, включаючи логіку вивчення слів, обробку відповідей та розрахунок статистики.
- Шар сховища: зберігання даних у реляційній базі даних (PostgreSQL).
- Журналювання: логування подій для відстеження роботи системи та виявлення помилок.
- Безпека: автентифікація та авторизація користувачів, захист від атак.

Для розробки веб-сервісу використано такі технології:

- Java Spring Boot: фреймворк для швидкої розробки веб-додатків на основі Java. Spring boot спрощує процес налаштування та розгортання додатку, а

також надає готові до використання компоненти для роботи з rest api, базами даних, безпекою та іншими аспектами веб-розробки.

- JPA (Java Persistence Api): стандарт java для об'єктно-реляційного відображення (ORM). JPA дозволяє працювати з даними в базі даних за допомогою об'єктів Java, спрощуючи розробку та підвищуючи продуктивність.
- Spring MVC: фреймворк для розробки веб-додатків за шаблоном model-view-controller. Spring mvc спрощує процес маршрутизації запитів, обробки даних та відображення результатів користувачеві.
- Spring Security: фреймворк для забезпечення безпеки веб-додатків. Spring Security надає готові до використання механізми для автентифікації, авторизації, шифрування паролів та захисту від атак.
- AOP (Aspect-Oriented Programming): парадигма програмування, що дозволяє розділити cross-cutting concerns, такі як логування та безпека, від основної бізнес-логіки додатку. AOP підвищує модульність коду та спрощує його підтримку.
- Thymeleaf: шаблонний двигун для створення динамічних html-сторінок. Thymeleaf інтегрується з Spring MVC та дозволяє легко створювати веб-сторінки з використанням даних з моделі.
- Lombok: бібліотека, що спрощує розробку java-класів шляхом автоматичної генерації коду для геттерів, сеттерів, конструкторів та інших стандартних методів.
- PostgreSQL: реляційна система управління базами даних (субд) з відкритим кодом. PostgreSQL відома своєю надійністю, гнучкістю та широкими можливостями.
- RESTful API: архітектурний стиль для створення веб-сервісів. RESTful API використовують стандартні HTTP-методи для виконання операцій над ресурсами, що робить їх простими у використанні та інтеграції.

- Maven: інструмент для управління залежностями та збірки проектів Java. Maven спрощує процес налаштування проекту, завантаження бібліотек та збірки додатку.

Розробка веб-сервісу для вивчення англійських слів дозволяє не тільки закріпити теоретичні знання, отримані під час навчання, але й набути практичного досвіду роботи з сучасними технологіями та інструментами розробки веб-додатків. Створення багатошарової архітектури, RESTful API, використання бази даних, журналювання та механізмів безпеки - все це є важливими аспектами сучасної веб-розробки, знання яких є необхідними для успішної кар'єри в IT-сфері.



## 1 ПРОЕКТУВАННЯ І РОЗРОБКА ВЕБ-ШАРУ

### 1.1 Теоретичні відомості.

REST (Representational State Transfer) - це архітектурний стиль для розробки розподілених систем, зокрема веб-додатків. Основна ідея REST полягає у використанні стандартних методів HTTP (GET, POST, PUT, DELETE) для виконання операцій над ресурсами, які ідентифікуються за допомогою URL. RESTful API (Application Programming Interface) - це інтерфейс, який надає доступ до ресурсів веб-сервісу, дотримуючись принципів REST.

RESTful API характеризується такими ключовими принципами:

- Клієнт-серверна архітектура: клієнт і сервер розглядаються як незалежні компоненти, що сприяє їхньому окремому розвитку та еволюції.
- Клієнт-серверна архітектура: клієнт і сервер розглядаються як незалежні компоненти, що сприяє їхньому окремому розвитку та еволюції.
- Безстатевість: кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для обробки запиту, не покладаючись на збережений стан на стороні сервера. Сервер не зберігає інформацію про стан клієнта між запитами, забезпечуючи незалежність кожного запиту.
- Кешування: відповіді сервера можуть бути кешовані, що покращує продуктивність системи, зменшуючи навантаження на сервер та час очікування для клієнта.
- Уніфікований інтерфейс: взаємодія з ресурсами здійснюється за допомогою стандартних HTTP-методів:
  - 1) GET: для отримання даних.
  - 2) POST: для створення нових ресурсів.
  - 3) PUT: для оновлення існуючих ресурсів.
  - 4) DELETE: для видалення ресурсів.
- Шарова система: архітектура може включати декілька шарів, що забезпечують додаткову функціональність, таку як безпека, балансування

навантаження та кешування. Це сприяє розподілу відповідальності та підвищенню масштабованості системи.

Java надає потужні інструменти для розробки RESTful API, зокрема Spring MVC. Spring MVC дозволяє легко створювати REST контролери, які обробляють HTTP-запити та повертають відповіді у зручному форматі, такому як JSON.

## 1.2 Аналіз вимог та проектування REST контролерів.

Веб-сервіс повинен надавати REST API з для користувачів та адміністраторів. Користувачі повинні мати можливість отримувати набори карток для вивчення, враховуючи індивідуальний прогрес користувача, відправку відповідей на картки та збереження результатів навчання, отримувати статистику навчання, включаючи кількість вивчених слів, прогрес за день, тиждень, місяць тощо.

Адміністратори мають доступ до всіх функцій користувачів, CRUD операцій з картками: створення, читання, оновлення та видалення карток.

Для реалізації цих функцій спроектовано наступні REST контролери:

- LearningController. Цей контролер відповідає за обробку запитів, пов'язаних з отриманням карток для навчання та відправкою відповідей. Має наступні ендпоінти:

1) `api/learn/get-cards`: запит для отримання набору карток для вивчення.

Сервіс враховує індивідуальний прогрес користувача, повертаючи картки, які потребують повторення, або нові картки, якщо користувач вивчив всі доступні. HTTP метод – GET. Відповідь – статус код 200 (OK) та JSON масив з картками у форматі CardProgressDto, який містить інформацію про слово, речення, переклад, синоніми, дату останньої відповіді та інші дані, необхідні для відображення картки на клієнтській стороні.

2) `api/learn/answer`: запит для відправки відповідей на картки. Очікує JSON об'єкт, який містить пари "ідентифікатор картки" - "правильність відповіді" (true/false). HTTP метод – POST. Відповідь – статус код 200

(OK) та JSON об'єкт з результатами відповідей у форматі `AnswerResultDto`. Цей об'єкт містить інформацію про те, чи була відповідь на кожну картку правильною, а також оновлені дані про наступне повторення картки, враховуючи алгоритм інтервального повторення.

- `StatsController`. Цей контролер відповідає за обробку запитів, пов'язаних з переглядом статистики навчання. Має наступні ендпоінти:

- 1) `/api/stats`: запит для отримання статистики навчання користувача. HTTP метод – GET. Відповідь – статус код 200 (OK) та JSON об'єкт з даними статистики у форматі `UserStats`. Цей об'єкт містить інформацію про загальну кількість вивчених слів, кількість вивчених слів за день, тиждень, місяць та дату останнього оновлення статистики.

- `CardController`. Цей контролер відповідає за обробку запитів, пов'язаних з CRUD операціями з картками. Має наступні ендпоінти:

- 1) `api/cards/`: запит для отримання списку всіх карток. HTTP метод – GET. Відповідь – статус код 200 (OK) та JSON масив з усіма картками у форматі `Card`.

- 2) `api/cards/{id}`: запит для отримання картки за ідентифікатором. HTTP метод – GET. Відповідь – статус код 200 (OK) та JSON об'єкт з даними картки у форматі `Card`, якщо картка знайдена. Статус код 404 (Not Found), якщо картка з таким ідентифікатором не існує.

- 3) `api/cards/{id}`: запит для оновлення картки. HTTP метод – PUT. Очікує JSON об'єкт з оновленими даними картки. Відповідь – статус код 200 (OK) та JSON об'єкт з оновленою картокою, якщо оновлення успішне.

- 4) `api/cards/{id}`: запит для видалення картки. HTTP метод – DELETE. Відповідь – статус код 200 (OK), якщо видалення успішне.

- 5) `api/cards/`: запит для створення нової картки. Очікує JSON об'єкт з даними нової картки. HTTP метод – POST. Відповідь – статус код

201 (Created) та JSON об'єкт з даними створеної картки, якщо створення успішне.

## 2 ПРОЕКТУВАННЯ І РОЗРОБКА СЕРВІСНОГО ШАРУ

Сервісний шар є ключовою складовою частиною архітектури веб-додатку, відповідальною за реалізацію бізнес-логіки. Він містить сервіси, які взаємодіють з репозиторіями для отримання та збереження даних, а також виконують обчислення, перевірки та іншу логіку, необхідну для роботи додатка.

### 2.1. Сервіс навчання (LearningService)

LearningService відповідає за логіку вивчення англійських слів, включаючи отримання карток для навчання та обробку відповідей користувача.

- Метод `getCardsForLearning(User user)`: цей метод отримує набір карток для навчання, враховуючи індивідуальний прогрес користувача. Вхідні дані: об'єкт `User`, що представляє поточного користувача. Вихідні дані: список карток у форматі `CardProgressDto`, готових для відображення користувачеві. Логіка роботи. Отримання карток з колоди (`IN_DECK`): метод спочатку перевіряє, чи є у користувача картки в активній колоді (статус `IN_DECK`). Якщо є, то повертаються ці картки, сортовані за датою наступного повторення (`due`). Формування нової колоди: якщо в колоді менше 5 карток або вона порожня, метод формує нову колоду, використовуючи картки зі статусом `READY` (готові до вивчення) та нові картки, яких користувач ще не бачив. Картки додаються в колоду по чергово (стара-нова-стара), щоб забезпечити різноманітність. Оновлення статусу карток: Для всіх карток, доданих до колоди, встановлюється статус `IN_DECK`. Формування DTO: для кожної картки формується об'єкт `CardProgressDto`, який містить необхідні дані для відображення на клієнтській стороні.
- Метод `processAnswers(User user, Map<Long, Boolean> answers)` обробляє відповіді користувача на картки та оновлює прогрес навчання. Вхідні дані: об'єкт `User`, що представляє поточного користувача. `Map<Long, Boolean> answers`: мапа, де ключем є ідентифікатор картки, а значенням - правильність відповіді (`true` - правильно, `false` - неправильно). Вихідні дані: список

результатів у форматі `AnswerResultDto`, що містить інформацію про правильність відповіді та оновлені дані про наступне повторення картки. Логіка роботи. перевірка даних: Метод перевіряє, чи всі ідентифікатори карток з мапи `answers` відповідають карткам з активної колоди користувача. оновлення прогресу: для кожної картки з мапи `answers`: отримується об'єкт `UserProgress` з бази даних. викликається метод `updateProgress()`, який оновлює прогрес навчання, враховуючи правильність відповіді та алгоритм інтервального повторення. Збереження оновленого об'єкта `UserProgress` в базі даних. Формування результатів: для кожної картки формується об'єкт `AnswerResultDto`, що містить інформацію про правильність відповіді та дату наступного повторення. Позначка останньої картки: якщо обробляється остання картка з колоди, відповідний об'єкт `AnswerResultDto` позначається прапорцем `isLastCard`. Скріншоти: Додайте скріншоти коду методу `processAnswers` з `LearningService`. Зверніть увагу на логіку перевірки даних, оновлення прогресу, формування результатів та позначки останньої картки.

- Допоміжний метод `updateProgress(UserProgress progress, boolean isCorrect)` оновлює прогрес навчання для однієї картки, враховуючи правильність відповіді та алгоритм інтервального повторення. Вхідні дані: об'єкт `UserProgress`, що представляє прогрес користувача для конкретної картки. `boolean isCorrect`: прапорець, що вказує на правильність відповіді. Вихідні дані: немає. Метод оновлює об'єкт `UserProgress`. Логіка роботи. Правильна відповідь: якщо відповідь правильна (`isCorrect == true`), метод збільшує кількість правильних відповідей (`reps`), оновлює інтервал повторення (`interval`) та дату наступного повторення (`due`) згідно з алгоритмом інтервального повторення, збільшує рівень вивченості (`learnedLevel`) та встановлює статус картки `READY`. Неправильна відповідь: якщо відповідь неправильна (`isCorrect == false`), метод скидає кількість правильних відповідей (`reps`), встановлює інтервал повторення (`interval`) на 1, зменшує дату наступного повторення (`due`), зменшує рівень вивченості (`learnedLevel`) та встановлює статус картки `READY`.

## 2.2. Сервіс статистики (StatsService)

StatsService відповідає за розрахунок та надання статистики навчання користувача.

- Метод `updateStats(User user)` оновлює статистику навчання користувача. Вхідні дані: об'єкт `User`, що представляє поточного користувача. Вихідні дані: немає. Метод оновлює об'єкт `UserStats` в базі даних. Логіка роботи. Отримання статистики: метод отримує об'єкт `UserStats` для поточного користувача з бази даних. Якщо статистика ще не існує, створюється новий об'єкт. Оновлення денної, тижневої та місячної статистики: якщо дата останнього оновлення статистики не співпадає з поточною датою, скидається денна статистика (`wordsLearnedToday`). Якщо сьогодні понеділок, скидається тижнева статистика (`wordsLearnedThisWeek`). Якщо сьогодні перший день місяця, скидається місячна статистика (`wordsLearnedThisMonth`). Розрахунок статистики: розраховує кількість слів, вивчених сьогодні, за тиждень та за місяць, використовуючи дані з `UserProgress` та алгоритм інтервального повторення. Оновлення загальної статистики: оновлює загальну кількість вивчених слів (`totalWordsLearned`). Збереження статистики: об'єкт `UserStats` зберігається в базі даних.
- Метод `getStatsForUser(User user)` отримує статистику навчання користувача з бази даних. Вхідні дані: об'єкт `User`, що представляє поточного користувача. Вихідні дані: об'єкт `UserStats`, що містить статистику навчання користувача. Логіка роботи: метод отримує об'єкт `UserStats` для поточного користувача з бази даних.

## 2.3. Сервіс карток (CardService)

CardService відповідає за CRUD операції з картками.

- Метод `getCardById(Long id)` отримує картку за її ідентифікатором. Вхідні дані: `Long id`: ідентифікатор картки. Вихідні дані: об'єкт `Optional<Card>`, що містить картку, якщо вона знайдена, або порожній `Optional`, якщо картка з таким ідентифікатором не існує. Логіка роботи: метод використовує репозиторій `CardRepository` для отримання картки за ідентифікатором.
- Метод `getCardsByWord(String word)` отримує список карток, слово яких містить заданий рядок. Вхідні дані: `String word`: рядок для пошуку. Вихідні дані: список карток `List<Card>`, що відповідають критерію пошуку. Логіка роботи: метод використовує репозиторій `CardRepository` для отримання карток, слово яких містить заданий рядок.
- Метод `getCardsByType(String type)` отримує список карток заданого типу. Вхідні дані: `String type`: Тип картки. Вихідні дані: список карток `List<Card>`, що відповідають заданому типу. Логіка роботи: використовує репозиторій `CardRepository` для отримання карток заданого типу.
- Метод `createCard(Card card)` створює нову картку. Вхідні дані: об'єкт `Card` з даними нової картки. Вихідні дані: створений об'єкт `Card`. Логіка роботи: метод використовує репозиторій `CardRepository` для збереження нової картки в базі даних.
- Метод `updateCard(Card card)` оновлює існуючу картку. Вхідні дані: об'єкт `Card` з оновленими даними картки. Вихідні дані: оновлений об'єкт `Card`. Логіка роботи: Метод використовує репозиторій `CardRepository` для оновлення картки в базі даних.
- Метод `deleteCard(Long id)` видаляє картку за її ідентифікатором. Вхідні дані: `Long id`: ідентифікатор картки. Вихідні дані: немає. Логіка роботи: використовує репозиторій `CardRepository` для видалення картки з бази даних.

## 2.4. Сервіс користувача (UserService)



UserService відповідає за логіку управління користувачами, включаючи реєстрацію та авторизацію.

- Метод `createUser(String username, String password)` створює нового користувача в системі. Вхідні дані: `String username`: ім'я користувача. `String password`: пароль користувача. Вихідні дані: об'єкт `User` - створений користувач. Логіка роботи. метод перевіряє, чи існує вже користувач з таким ім'ям. Якщо так, генерується виняток `IllegalArgumentException`. Створюється новий об'єкт `User` з заданим ім'ям та захешованим паролем. Користувачу призначається роль `USER`. Користувач зберігається в базі даних.
- Метод `findByUsername(String username)` шукає користувача за ім'ям. Вхідні дані: `String username`: ім'я користувача. Вихідні дані: об'єкт `Optional<User>`, що містить користувача, якщо він знайдений, або порожній `Optional`, якщо користувач з таким ім'ям не існує. Логіка роботи: метод використовує репозиторій `UserRepository` для пошуку користувача за ім'ям.
- Метод `getCurrentUser()` метод отримує об'єкт поточного користувача, що аутентифікований в системі. Вхідні дані: немає. Вихідні дані: об'єкт `User`, що представляє поточного користувача. Логіка роботи: отримує об'єкт `Authentication` з контексту безпеки `Spring Security`. Перевіряється, чи користувач аутентифікований. Якщо ні, генерується виняток `IllegalStateException`. Отримується ім'я користувача з об'єкта `Authentication`. Метод шукає користувача за ім'ям, використовуючи `findByUsername()`. Якщо користувач знайдений, повертається його об'єкт. Якщо ні, генерується виняток `IllegalStateException`.

## 2.5. Сервіс форматування часу (TimeFormattingService)

`TimeFormattingService` надає допоміжні методи для форматування часу в зручний для користувача формат.

- Метод `formatTimeAgo(LocalDateTime dateTime)` форматує дату та час в форматі "скільки часу тому" (наприклад, "5 хвилин тому", "2 дні тому").

Вхідні дані: `LocalDateTime` `dateTime`: дата та час, які потрібно відформатувати. Вихідні дані: `String`: відформатований рядок. Логіка роботи: Метод обчислює різницю між заданою датою та часом і поточною датою та часом, та повертає рядок у відповідному форматі.

- Метод `formatTimeUntil(LocalDateTime dateTime)` форматує дату та час в форматі "через скільки часу" (наприклад, "через 10 хвилин", "через 3 дні").
- Допоміжні методи `formatMinutes`, `formatHours`, `formatDays` використовуються в методах `formatTimeAgo` та `formatTimeUntil` для форматування хвилин, годин та днів.

### 3 ПРОЕКТУВАННЯ І РОЗРОБКА ВЕБ-ШАРУ

Шар сховища відповідає за збереження та управління даними в додатку. Він забезпечує взаємодію з базою даних, виконання CRUD-операцій (Create, Read, Update, Delete) та підтримує цілісність даних. У цьому розділі розглянемо проектування сутностей та реалізацію шару сховища з використанням Spring Data JPA та PostgreSQL.

#### 3.1 Сутності та їх атрибути.

Для ефективного функціонування веб-сервісу було визначено наступні сутності, що відображають ключові об'єкти предметної області:

1) User (Користувач). Атрибути:

- id: Унікальний ідентифікатор користувача (Long).
- username: Ім'я користувача (String).
- password: Хешований пароль користувача (String).
- roles: Список ролей користувача (Collection).

2) Role (Роль):

- id: Унікальний ідентифікатор ролі (Long).
- name: Назва ролі (String).

3) Card (Картка):

- id: Унікальний ідентифікатор картки (Long).
- word: Слово на англійській мові (String).
- sentence: Речення, що містить слово (String).
- translation: Переклад слова (String).
- synonyms: Синоніми слова (String).
- type: Тип слова (наприклад, іменник, дієслово) (String).

4) UserProgress (Прогрес користувача):

- id: Складений ключ, що містить ідентифікатори користувача та картки (UserCardId).
- user: Посилання на користувача (User).

- card: Посилання на картку (Card).
- learnedLevel: Рівень вивченості слова (int).
- lastAnswered: Дата та час останньої відповіді на картку (LocalDateTime).
- ease: Коефіцієнт легкості запам'ятовування (double).
- due: Дата та час наступного повторення (LocalDateTime).
- interval: Інтервал повторення (int).
- reps: Кількість правильних відповідей поспіль (int).
- status: Статус картки (READY, IN\_DECK) (CardStatus).

Зв'язки між сутностями

- 1) User - Role (Багато-до-багатьох): користувач може мати декілька ролей, і одна роль може бути призначена багатьом користувачам.
- 2) User - UserProgress (Один-до-багатьох): один користувач може мати прогрес по багатьох картках.
- 3) Card - UserProgress (Один-до-багатьох): одна картка може бути вивчена багатьма користувачами.

На основі виділених сутностей, їх атрибутів та зв'язків між сутностями створено інформаційну модель в нотації Мартіна, яка представлена на рисунку 3.1

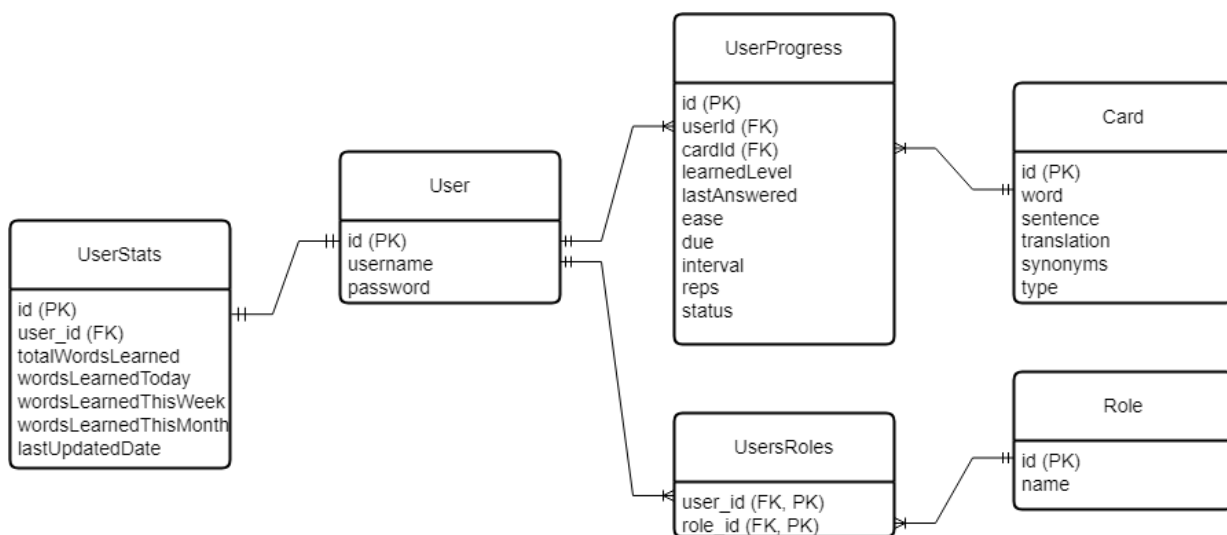


Рисунок 3.1 – Інформаційна модель в нотації Мартіна

## 3.2 Spring Data JPA

Spring Data JPA – це фреймворк, що спрощує розробку шару сховища для додатків Spring. Він надає абстракцію над JPA (Java Persistence API), що дозволяє легко створювати репозиторії, які взаємодіють з базою даних.

Можливості Spring Data JPA:

- 1) Spring Data JPA автоматично генерує запити до бази даних на основі імені методів репозиторіїв, що зменшує обсяг коду, який потрібно писати розробнику.
- 2) Spring Data JPA підтримує різні системи управління базами даних, включаючи PostgreSQL.
- 3) Spring Data JPA легко інтегрується з іншими компонентами Spring, такими як транзакції та безпека.
- 4) Spring Data JPA дозволяє писати власні запити до бази даних, використовуючи JPQL (Java Persistence Query Language) або native SQL.
- 5) Spring Data JPA надає розширені можливості, такі як пагінація, сортування, кастомні запити (JPQL або native SQL) та аудит.

Налаштування підключення до бази даних:

- Встановлюємо PostgreSQL.
- Створюємо базу даних для веб-сервісу.
- Конфігуруємо Spring Boot для підключення до бази даних, вказуючи URL, ім'я користувача та пароль в файлі `application.properties`.
- Визначаємо стратегію створення/оновлення схеми бази даних за допомогою властивості `spring.jpa.hibernate.ddl-auto`. Наприклад, `update` дозволить Spring Data JPA автоматично оновлювати схему бази даних при запуску додатка, синхронізуючи її з сутностями.

Створення репозиторіїв: для сутностей створюємо інтерфейс репозиторію, розширюючи інтерфейс `JpaRepository`. Spring Data JPA автоматично генерує реалізації цих інтерфейсів під час запуску додатка.

```

public interface UserProgressRepository extends JpaRepository<UserProgress, UserCardId> {

    1 usage  levsemyvolos
    Optional<UserProgress> findByUserAndCard(User user, Card card);

    1 usage  levsemyvolos
    long countByUserAndStatus(User user, CardStatus cardStatus);

    3 usages  levsemyvolos
    @Query("SELECT up FROM UserProgress up " +
           "JOIN FETCH up.card " +
           "WHERE up.user = :user AND up.status = :status")
    List<UserProgress> findUserProgressWithCardByUserAndStatus(User user, CardStatus status);

    1 usage  levsemyvolos
    @Query("SELECT up FROM UserProgress up WHERE up.user.id = :userId AND up.card.id = :cardId")
    Optional<UserProgress> findByUserAndCardId(Long userId, Long cardId);

    3 usages  levsemyvolos
    long countByUserAndStatusAndLastAnsweredAfter(User user, CardStatus cardStatus, LocalDateTime localDateTime);
}

```

Рисунок 3.2 – Приклад реалізації репозиторію

## 4 ПРОЕКТУВАННЯ І РОЗРОБКА ЖУРНАЛЮВАННЯ

Журналювання відіграє важливу роль у розробці та підтримці програмного забезпечення. Воно дозволяє відстежувати події та дії в системі, допомагаючи виявляти та усувати помилки, моніторити роботу, аналізувати поведінку користувачів та забезпечувати безпеку.

### 4.1 Теоретичні відомості об АОР.

Аспектно-орієнтоване програмування (АОР) - це парадигма програмування, що дозволяє розділити cross-cutting concerns, такі як журналювання, безпека та обробка транзакцій, від основної бізнес-логіки додатка.

Основні поняття АОР:

- 1) Аспект (Aspect): Модуль, що містить код, який реалізує cross-cutting concern.
- 2) Точка з'єднання (Join Point): Місце в коді, де може бути застосований аспект (наприклад, виклик методу, обробка винятку).
- 3) Поінткат (Pointcut): Вираз, який визначає набір точок з'єднання, до яких буде застосований аспект.
- 4) Порада (Advice): Код, який виконується в точці з'єднання.

SLF4J (Simple Logging Facade for Java) - це абстракція для різних бібліотек журналювання, таких як Logback, Log4j та java.util.logging. SLF4J дозволяє легко перемикатися між різними бібліотеками журналювання без зміни коду додатка.

Logback - це потужна та гнучка бібліотека журналювання, що надає розширені можливості налаштування, включаючи:

Рівні журналювання: TRACE, DEBUG, INFO, WARN, ERROR.

Апендери: Визначають, куди будуть записуватися журнали (наприклад, в консоль, файл, базу даних).

Формат повідомлень: Визначає формат журнальних повідомлень.

Фільтри: Дозволяють фільтрувати журнали за різними критеріями.

### Переваги журналювання

- Журнали дозволяють швидко знаходити та усувати помилки в коді, аналізуючи послідовність подій та інформацію про винятки.
- Журнали допомагають моніторити роботу додатка, відстежуючи ключові метрики та події.
- Журнали можуть використовуватися для аналізу поведінки користувачів, що дозволяє покращувати інтерфейс та функціональність додатка.
- Журнали допомагають відстежувати підозрілу активність та виявляти потенційні загрози безпеці.

Отже, журналювання є важливим аспектом розробки та підтримки веб-сервісу. Використання AOP та Logback дозволяє реалізувати гнучку та потужну систему журналювання, яка допомагає покращити якість коду, моніторити роботу, аналізувати поведінку користувачів та забезпечувати безпеку.

### 4.2 Визначення аспектів для журналювання

У веб-сервісі для вивчення англійських слів використовується AOP та Logback для журналювання подій, пов'язаних з роботою REST контролерів, сервісів та репозиторіїв.

Для журналювання створено аспект `LoggingAspect`, який перехоплює виклики методів у REST контролерах, сервісах та репозиторіях. Аспект `LoggingAspect` логує інформацію про:

- Виклик методу: назва методу, клас, аргументи.
- Повернене значення: результат виконання методу.
- Винятки: якщо під час виконання методу виник виняток, аспект логує інформацію про виняток.

В проєкті ми використовували:

- 1) анотації `@Around advice` для перехоплення викликів методів.
- 2) `pointcut expressions` для визначення набору методів, до яких буде застосований аспект.



- 3) `ProceedingJoinPoint` для отримання інформації про викликаний метод.
- 4) `Logger` для запису журнальних повідомлень.

#### 4.3 Налаштування журналювання з використанням SLF4J та Logback

Для налаштування журналювання використовується файл `logback-spring.xml`.

Налаштування в `logback-spring.xml`(рис 4.1):

- Встановлено рівень `INFO`, що означає, що будуть логуватися повідомлення рівнів `INFO`, `WARN` та `ERROR`.
- `ConsoleAppender`: Виводить журнали в консоль.
- `RollingFileAppender`: Записує журнали у файли, що обертаються (`rolling files`). Налаштовано патерн імен файлів, максимальну кількість файлів та максимальний розмір файлу.
- Налаштовано формат журнальних повідомлень, що включає дату, час, рівень журналювання, назву класу та повідомлення.

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <include resource="org/springframework/boot/logging/logback/base.xml"/>

  <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>logs/application.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>logs/application.%d{yyyy-MM-dd}.log</fileNamePattern>
      <maxHistory>30</maxHistory>
    </rollingPolicy>
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
  </root>
</configuration>

```

Рисунок 4.1 – logback-spring.xml

## 5 ПРОЕКТУВАННЯ І РОЗРОБКА БЕЗПЕКИ

### 5.1 Теоретичні відомості о Spring Security.

Spring Security - це потужний фреймворк, що надає комплексні рішення для забезпечення безпеки веб-додатків на базі Java. Spring Security інтегрується з Spring MVC та надає механізми для автентифікації, авторизації, захисту від поширених атак та інші функції безпеки.

Основні концепції безпеки:

Автентифікація: процес підтвердження особи користувача, що намагається отримати доступ до системи. Вона зазвичай включає перевірку облікових даних, таких як ім'я користувача та пароль.

Авторизація: процес визначення, чи має аутентифікований користувач доступ до певних ресурсів або функцій системи.

Шифрування: процес перетворення даних у нечитабельний формат для захисту конфіденційної інформації.

Захист від атак: заходи, спрямовані на запобігання та блокування різних типів атак, таких як Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), SQL-ін'єкції та інші.

### 5.2 Налаштування Spring Security

Клас `SecurityConfiguration` відповідає за конфігурацію Spring Security у веб-додатку.

Налаштування в проєкті:

- `@Configuration` та `@EnableWebSecurity`: Ці анотації вказують, що клас `SecurityConfiguration` є конфігураційним класом Spring та активує Spring Security.
- `passwordEncoder()`: Цей метод визначає бін `PasswordEncoder`, що використовується для хешування паролів. У нашому випадку

використовується BCryptPasswordEncoder, що забезпечує надійне хешування паролів за допомогою алгоритму bcrypt.

- `filterChain(HttpSecurity http)`: Цей метод конфігурує обробку HTTP-запитів Spring Security.

Детальний опис конфігурації `filterChain(HttpSecurity http)` (рис 5.1):

- `http.authorizeHttpRequests()`: Конфігурація правил авторизації для різних URL-адрес.
- `.requestMatchers("/register/**", "/index", "/css/**", "/js/**").permitAll()`: Дозволяє доступ до сторінок реєстрації, головної сторінки, CSS та JS файлів без автентифікації.
- `.requestMatchers("/api/cards/**").hasAuthority("ADMIN")`: Дозволяє доступ до API для роботи з картками тільки користувачам з роллю ADMIN.
- `.requestMatchers("/api/learn/**", "/learn", "/stats").authenticated()`: Дозволяє доступ до API для навчання, сторінки навчання та сторінки статистики тільки аутентифікованим користувачам.
- `.anyRequest().authenticated()`: Всі інші запити вимагають автентифікації.
- `formLogin()`: Конфігурація форми логіна.
- `.loginPage("/login")`: Визначає URL-адресу сторінки логіна.
- `.loginProcessingUrl("/login")`: Визначає URL-адресу, на яку відправляється форма логіна.
- `.defaultSuccessUrl("/learn")`: Визначає URL-адресу, на яку перенаправляється користувач після успішного логіна.
- `.permitAll()`: Дозволяє доступ до сторінки логіна без автентифікації.
- `logout()`: Конфігурація виходу з системи.
- `.logoutRequestMatcher(new AntPathRequestMatcher("/logout"))`: Визначає URL-адресу для виходу з системи.
- `.permitAll()`: Дозволяє доступ до URL-адреси виходу з системи без автентифікації.

```

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests((authorize) -> authorize
            .requestMatchers("@ /register/**", "@ /index", "@ /css/**", "@ /js/**").permitAll()
            .requestMatchers("@ /api/cards/**").hasAuthority("ADMIN")
            .requestMatchers("@ /api/learn/**", "@ /learn", "@ /stats").authenticated()
            .anyRequest().authenticated()
        )
        .httpBasic(Customizer.withDefaults())
        .formLogin(form -> form
            .loginPage("/login")
            .loginProcessingUrl("/login")
            .defaultSuccessUrl("/learn")
            .permitAll()
        )
        .logout(logout -> logout
            .logoutRequestMatcher(new AntPathRequestMatcher("@ /logout"))
            .permitAll()
        )
        .csrf(AbstractHttpConfigurer::disable);
    return http.build();
}

```

Рисунок 5.1 – filterChain

Інтерфейс `UserDetailsService` використовується `Spring Security` для завантаження інформації про користувача з бази даних під час автентифікації. Клас `UserDetailsServiceImpl` реалізує цей інтерфейс та визначає, як завантажувати користувача з бази даних.

Методи:

- `loadUserByUsername(String username)`: отримує ім'я користувача та завантажує об'єкт `User` з бази даних за допомогою репозиторія `UserRepository`.
- `mapRolesToAuthorities(Collection<Role> roles)`: перетворює список ролей користувача у список `GrantedAuthority`, який використовується `Spring Security` для авторизації.

Отже, `Spring Security` надає потужні та гнучкі механізми для забезпечення безпеки веб-додатків. Клас `SecurityConfiguration` визначає конфігурацію `Spring Security`, включаючи правила авторизації, налаштування форми логіна та виходу з системи. Клас `UserDetailsServiceImpl` відповідає за завантаження інформації про користувача з бази даних під час автентифікації. Використання `Spring Security`

дозволяє захистити веб-сервіс від несанкціонованого доступу та поширених атак, забезпечуючи безпеку даних користувачів.

## 6 ТЕСТУВАННЯ ДОДАТКУ

### 6.1. Цілі та завдання тестування

Тестування веб-застосунку для вивчення англійської мови має на меті забезпечити його коректну роботу, відповідність функціональним вимогам та виявлення потенційних помилок. Основними цілями тестування є:

- 1) Перевірка функціональності API: Забезпечити, щоб усі API-ендпоінти працювали коректно та повертали очікувані дані, а саме:
  - правильність отримання карток за ID, словом та типом;
  - коректність створення, оновлення та видалення карток;
  - правильність отримання карток для навчання;
  - обробка відповідей користувача;
  - отримання статистики навчання.
- 2) Перевірка логіки навчання: Забезпечити, щоб алгоритм інтервального повторення та оновлення прогресу навчання працювали правильно, відповідно до заданих параметрів та логіки, описаної в розділі 3.
- 3) Перевірка роботи Spring Security: Забезпечити, щоб автентифікація та авторизація працювали коректно, а доступ до ресурсів був обмежений відповідно до ролей користувачів. Адміністратор має мати доступ до керування картками, а звичайні користувачі - лише до навчання та перегляду статистики.
- 4) Перевірка обробки помилок: Забезпечити, щоб застосунок коректно обробляв помилки, такі як некоректні запити, неавторизований доступ, відсутність даних тощо, та повертав відповідні коди стану HTTP.
- 5) Перевірка роботи інтерфейсу користувача: Забезпечити зручність та інтуїтивність інтерфейсу користувача, коректність відображення даних, обробки введених даних та взаємодії з API.

### 6.2. Методи та інструменти тестування

Для тестування веб-застосунку було використано наступні методи та інструменти:

- Ручне тестування API: використання Postman для надсилання запитів до API та перевірки відповідей сервера.
- Ручне тестування інтерфейсу користувача: взаємодія з інтерфейсом користувача в браузері для перевірки його функціональності та зручності.

### 6.3. Тестування API

Тестування API проводилося за допомогою Postman. Нижче наведено описи запитів та результатів для кожного API-ендпоінту.

CardController API(рис 6.1-6.7):

Отримання картки за ID (авторизований адмін):

URL: `http://localhost:8080/api/cards/5`

Метод: GET

Авторизація: Basic Auth (admin:admin)



Рисунок 6.1 – Результат запиту

Отримання картки за ID (не авторизований):

URL: `http://localhost:8080/api/cards/5`

Метод: GET



Авторизація: - Basic Auth (test:test)

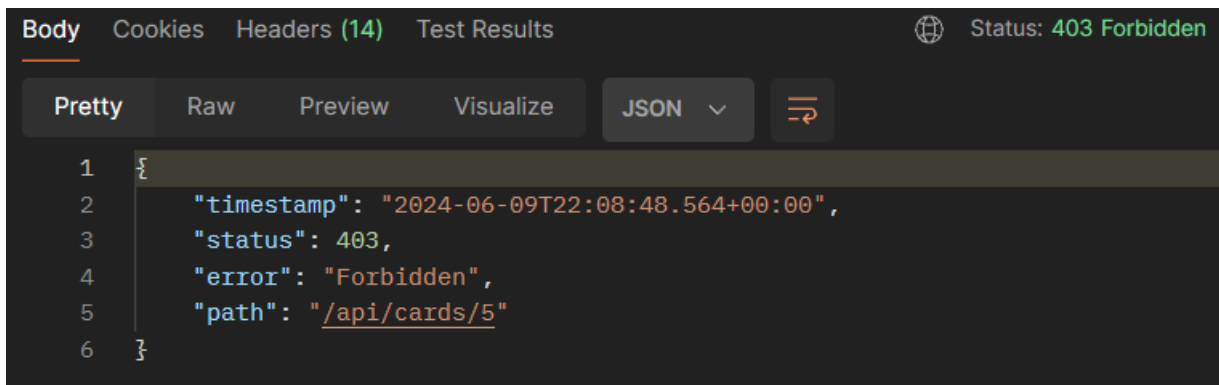


Рисунок 6.2 – Результат запиту

Отримання карток за словом (авторизований адмін):

URL: <http://localhost:8080/api/cards/word/learn>

Метод: GET

Авторизація: Basic Auth (admin:admin)

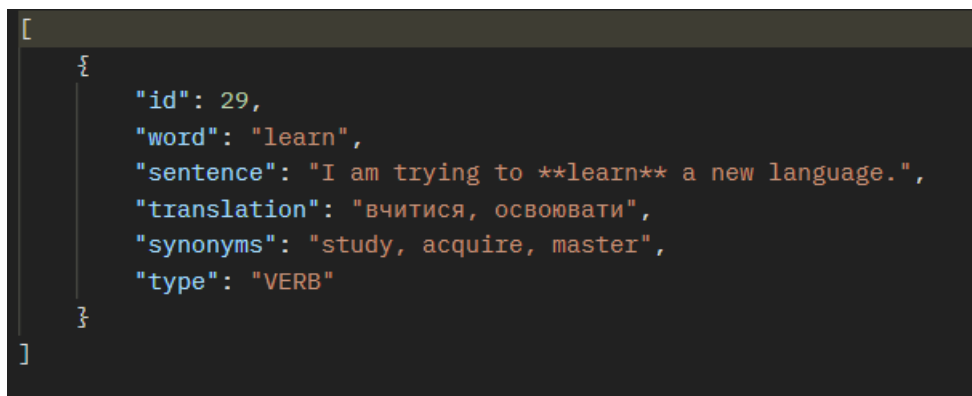


Рисунок 6.3 – Результат запиту

Отримання карток за типом (авторизований адмін):

URL: <http://localhost:8080/api/cards/type/verb>

Метод: GET

Авторизація: Basic Auth (admin:admin)

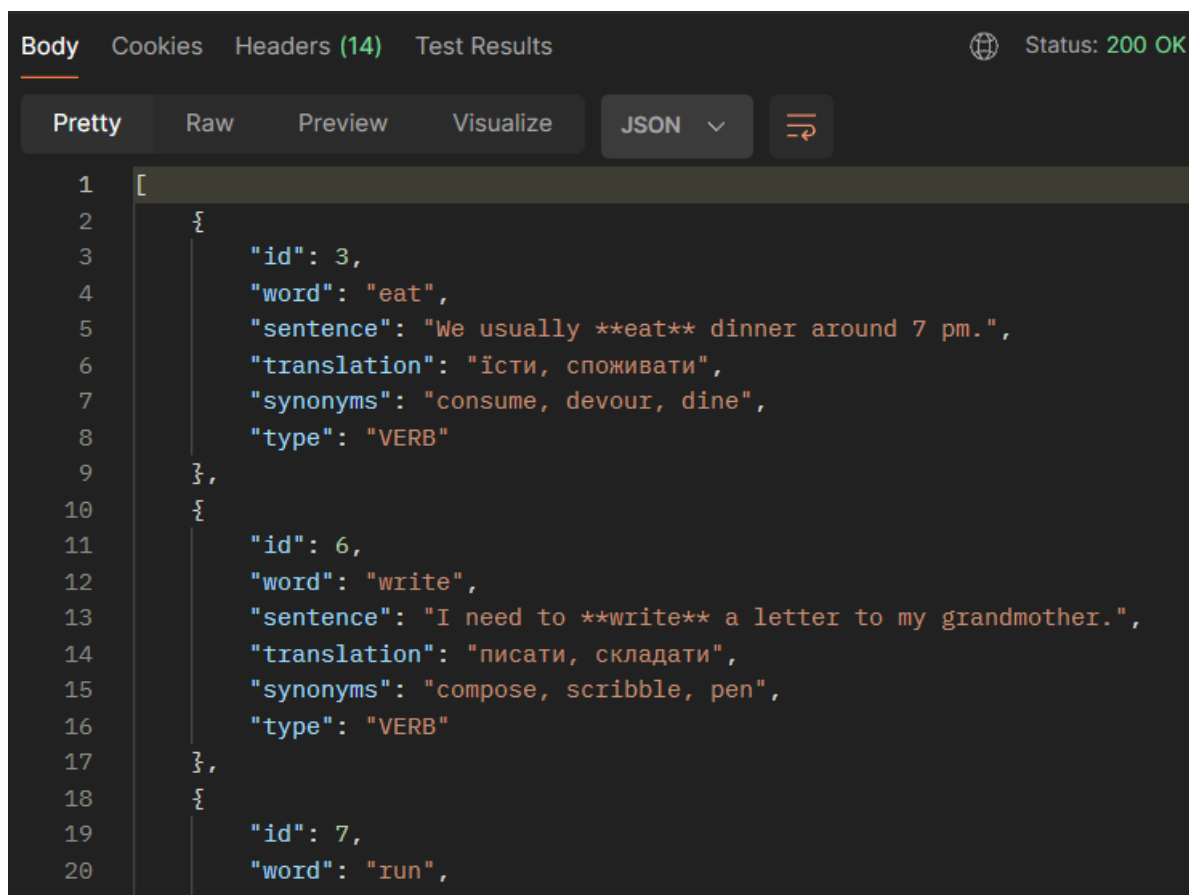


Рисунок 6.4 – Результат запиту

Створення картки (авторизований адмін):

URL: <http://localhost:8080/api/cards>

Метод: POST

Авторизація: Basic Auth (admin:admin)

Тіло запиту: JSON-об'єкт з даними картки (word, sentence, translation, synonyms, type).

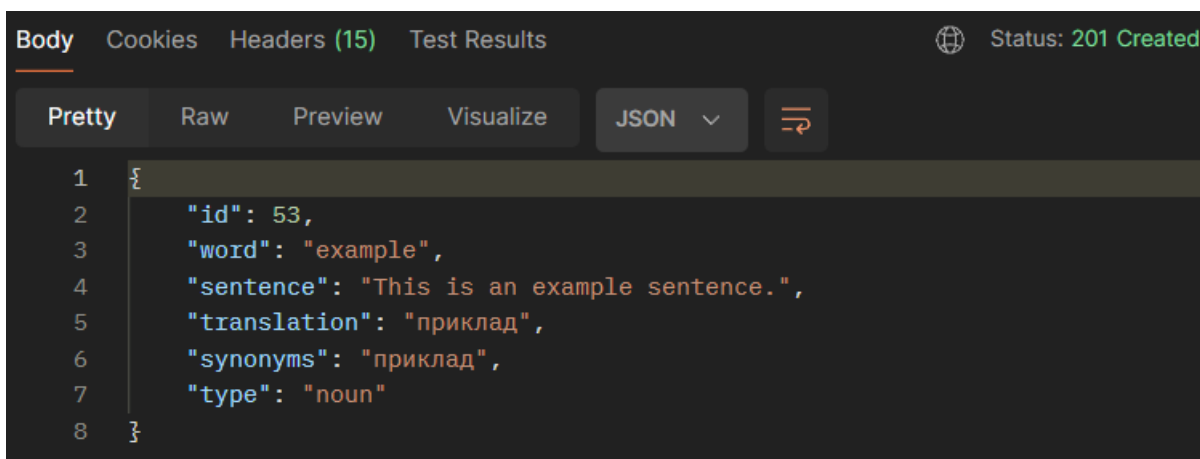


Рисунок 6.5 – Результат запиту

Оновлення картки (авторизований адмін):

URL: <http://localhost:8080/api/cards/1>

Метод: PUT

Авторизація: Basic Auth (admin:admin)

Тіло запиту: JSON-об'єкт з оновленими даними картки.

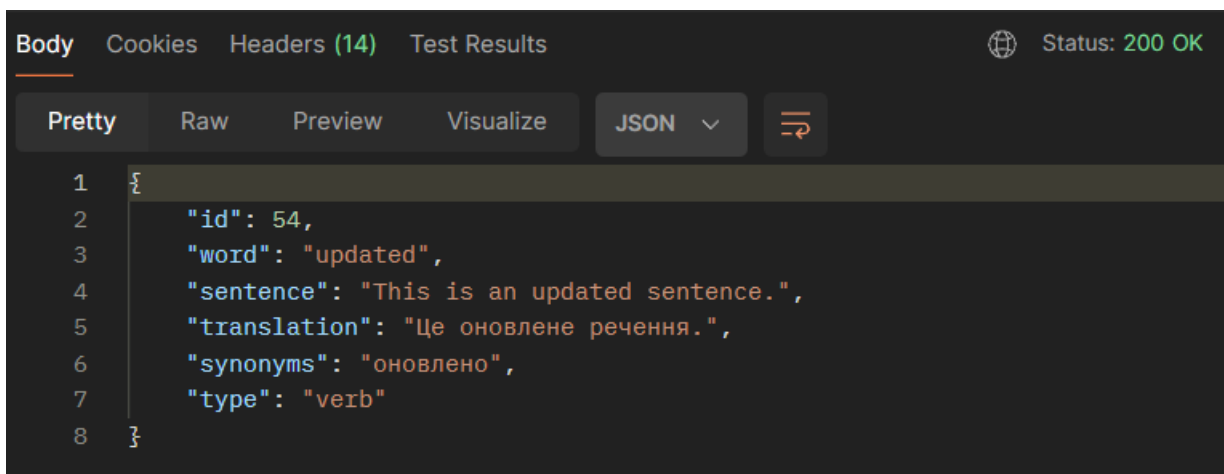


Рисунок 6.6 – Результат запиту

Видалення картки (авторизований адмін):

URL: <http://localhost:8080/api/cards/{id}>

Метод: DELETE

Авторизація: Basic Auth (admin:admin)



Рисунок 6.7 – Результат запиту

LearningController API (рис 6.7-6.9):

Отримати картки для навчання (авторизований користувач):

URL: <http://localhost:8080/api/learn/get-cards>

Метод: GET

Авторизація: Basic Auth (test:test)

Очікуваний результат: Статус код 200 OK, список карток для навчання в тілі відповіді у форматі JSON. Список має містити не більше MAX\_WORDS\_IN\_DECK карток, з урахуванням чергування "стара-нова-стара...".

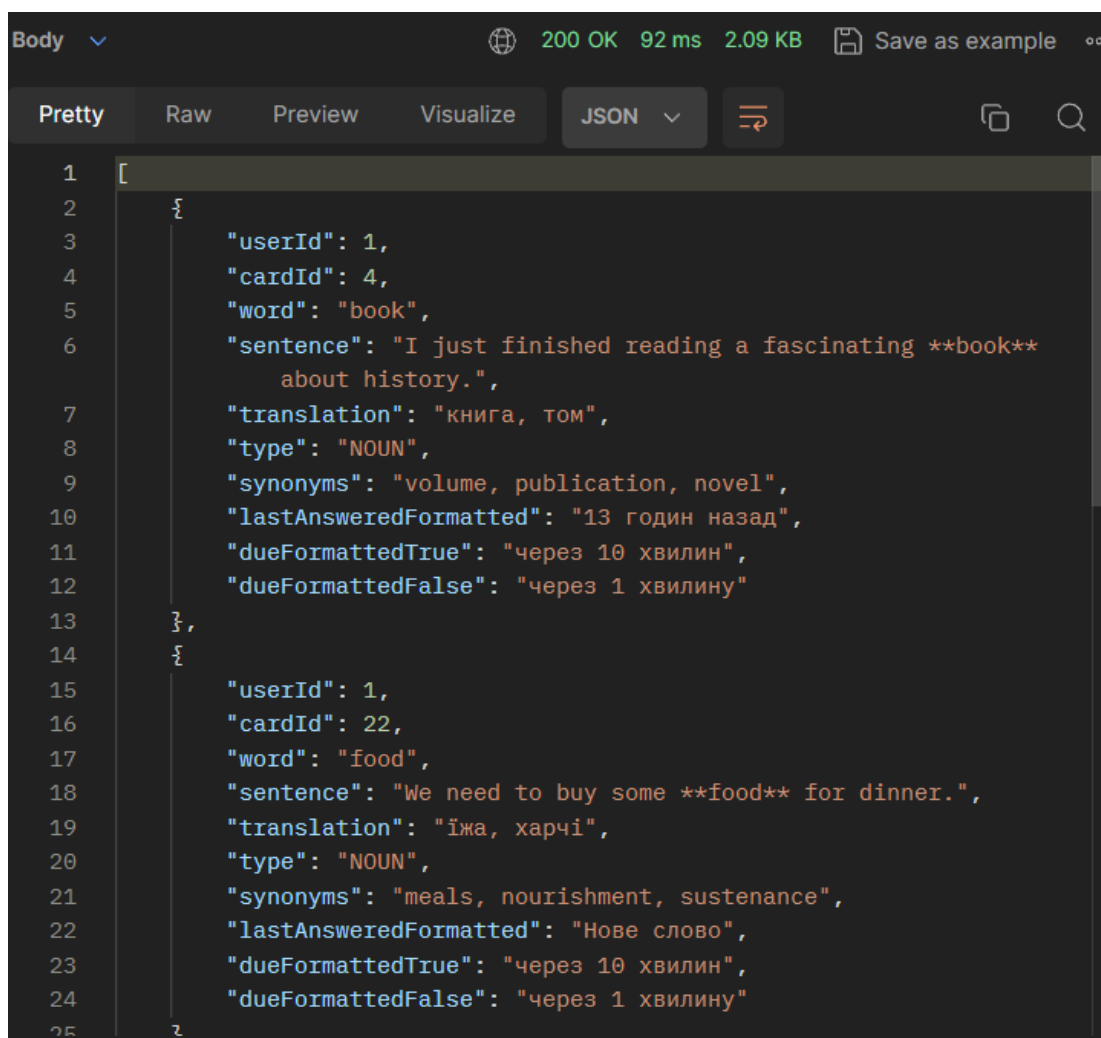


Рисунок 6.8 – Результат запиту

Відправити відповіді (авторизований користувач):

URL: `http://localhost:8080/api/learn/answer`

Метод: POST

Авторизація: Basic Auth (test:test)

Тіло запиту: JSON-об'єкт, де ключі - це ID карток, а значення - true (правильна відповідь) або false (неправильна відповідь).

Очікуваний результат: Статус код 200 OK, список `AnswerResultDto` в тілі відповіді у форматі JSON, що містить інформацію про результати відповідей для кожної картки.

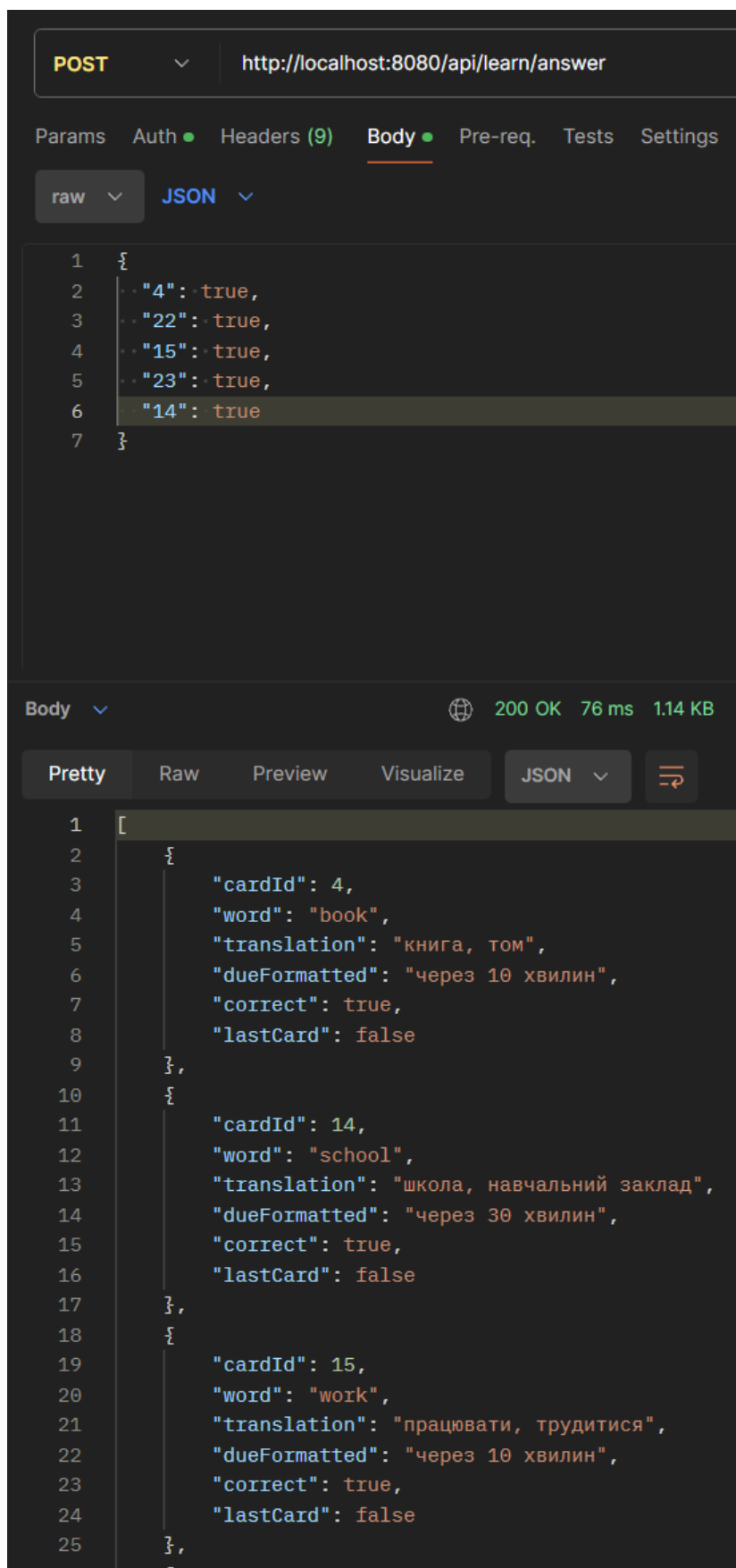


Рисунок 6.9 – Результат запиту

StatsRestController API (рис 6.10)

Отримання статистики (авторизований користувач):

URL: <http://localhost:8080/api/stats>

Метод: GET

Авторизація: Basic Auth (test:test)

```
{
  "id": 1,
  "user": {
    "id": 1,
    "username": "test",
    "password": "$2a$10$Y00W/
    pI0xwnzP9Z88XnVwua0zZFB8pX3AXL9S3rqHzC3Corbj8ICS",
    "roles": [
      {
        "id": 1,
        "name": "USER"
      }
    ]
  },
  "totalWordsLearned": 22,
  "wordsLearnedToday": 10,
  "wordsLearnedThisWeek": 22,
  "wordsLearnedThisMonth": 22,
  "lastUpdatedDate": "2024-06-10"
}
```

Рисунок 6.10 – Результат запиту

#### 6.4. Тестування інтерфейсу користувача

Тестування інтерфейсу користувача проводилося шляхом ручної взаємодії з веб-сторінками застосунку в браузері. Перевірялася коректність відображення даних, робота форм, переходи між сторінками, обробка введених даних, а також загальна зручність та інтуїтивність інтерфейсу.

#### 6.5. Висновки

Проведене тестування підтвердило, що веб-застосунок для вивчення англійської мови працює коректно та відповідає заданим функціональним вимогам. API-ендпоінти повертають очікувані дані, алгоритм інтервального повторення працює правильно, Spring Security забезпечує захист ресурсів, а інтерфейс користувача є зручним та інтуїтивно зрозумілим.

Звичайно, тестування не може гарантувати повну відсутність помилок, але воно значно підвищує надійність та якість програмного забезпечення. Важливо проводити тестування на всіх етапах розробки, щоб своєчасно виявляти та виправляти помилки, забезпечуючи високу якість кінцевого продукту.



## 7 ІНСТРУКЦІЯ КОРИСТУВАЧА

Для зручної роботи з веб-сервісом були створені веб-сторінки для користувачей із зручною навігацією з використанням HTML, CSS, JavaScript та Thymeleaf. Нижче наведено опис цих сторінок та можливості взаємодії з ними.

### 7.1. Головна сторінка

Назва сторінки: Головна сторінка

Адреса: <http://localhost:8080/index>

Опис: на головній сторінці ви побачите вітання та кнопку "Почати навчання". Якщо ви ще не зареєстровані, натисніть на посилання "Register" у верхньому меню, щоб створити обліковий запис. Якщо ви вже зареєстровані, натисніть на посилання "Login" у верхньому меню, щоб увійти в систему(рис 7.1).

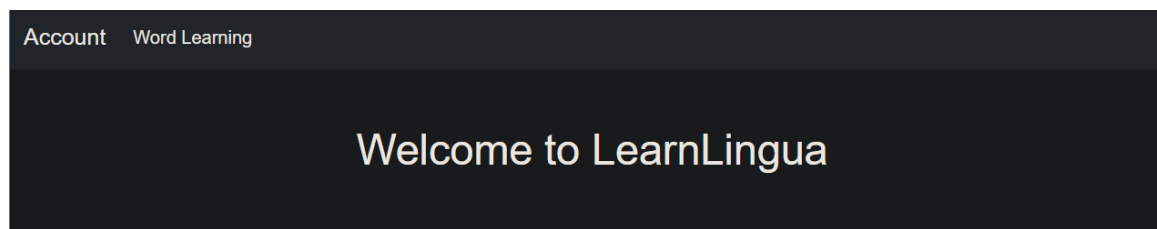


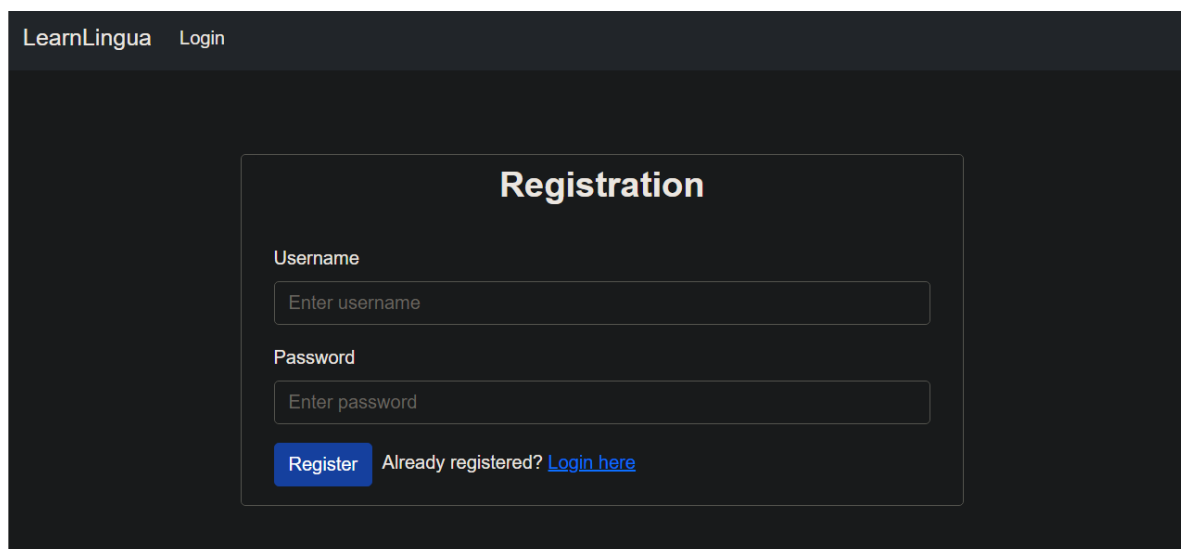
Рисунок 7.1 — Головна сторінка

### 7.2. Реєстрація

Назва сторінки: Реєстрація

Адреса: <http://localhost:8080/register>

Опис: на сторінці реєстрації ви побачите форму, де потрібно ввести ім'я користувача та пароль. Введіть бажані дані та натисніть кнопку "Register". Після успішної реєстрації ви будете перенаправлені на сторінку логіна(рис 7.2).



The image shows a web page for 'LearnLingua' with a 'Login' link in the top navigation bar. The main content area features a 'Registration' form. The form has two input fields: 'Username' with a placeholder 'Enter username' and 'Password' with a placeholder 'Enter password'. Below the password field is a blue 'Register' button. To the right of the button is the text 'Already registered?' followed by a blue link 'Login here'.

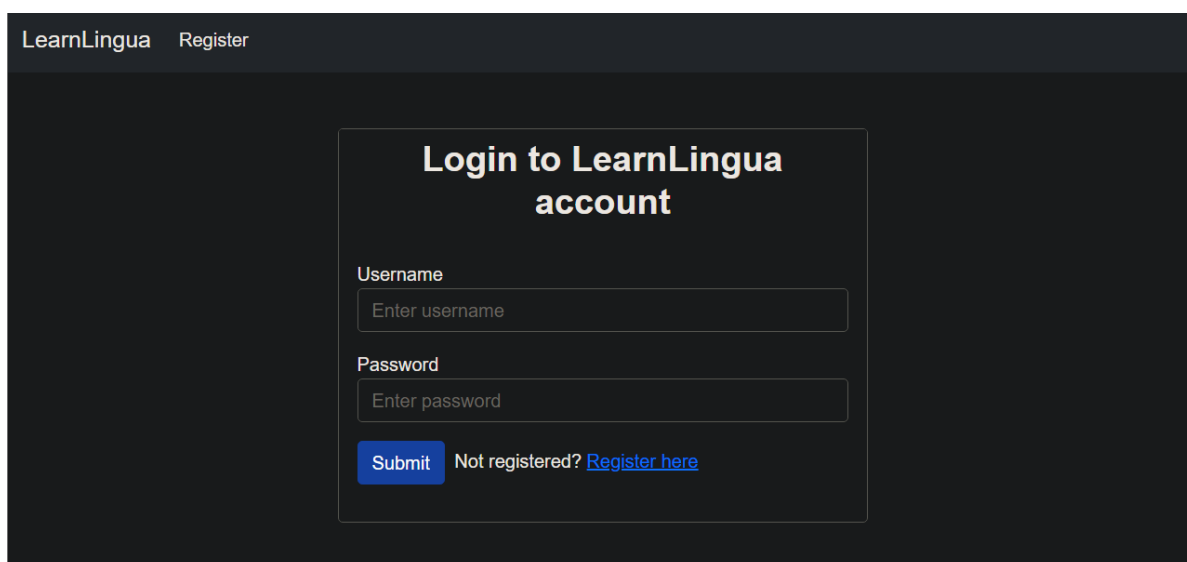
Рисунок 7.2 — Реєстрація

### 7.3. Логін

Назва сторінки: Логін

Адреса: <http://localhost:8080/login>

Опис: на сторінці логіна ви побачите форму, де потрібно ввести ім'я користувача та пароль, які ви вказали під час реєстрації. Введіть дані та натисніть кнопку "Submit". Після успішного логіна ви будете перенаправлені на сторінку навчання(рис 7.3).



The image shows a web page for 'LearnLingua' with a 'Register' link in the top navigation bar. The main content area features a 'Login to LearnLingua account' form. The form has two input fields: 'Username' with a placeholder 'Enter username' and 'Password' with a placeholder 'Enter password'. Below the password field is a blue 'Submit' button. To the right of the button is the text 'Not registered?' followed by a blue link 'Register here'.

Рисунок 7.3 — Логін

## 7.4. Навчання

Назва сторінки: Навчання

Адреса: <http://localhost:8080/learn>

Опис: На сторінці навчання ви побачите картки з англійськими словами (рис 7.4-7.6). Кожна картка містить:

Речення: Речення з пропущеним словом, яке вам потрібно вгадати.

Переклад: Переклад речення українською мовою.

Синоніми: Синоніми пропущеного слова.

Лічильник: Показує номер поточної картки та загальну кількість карток.

Поле для введення відповіді: Введіть свою відповідь у це поле та натисніть клавішу "Enter".

Кнопка "->": Після введення відповіді, натисніть цю кнопку, щоб перейти до наступної картки.

Інформація про останню відповідь: Показує, коли ви востаннє відповідали на цю картку, та чи була ваша відповідь правильною.

Після завершення навчання ви побачите результати, де буде вказано, які слова ви вгадали, а які - ні (рис 7.7).

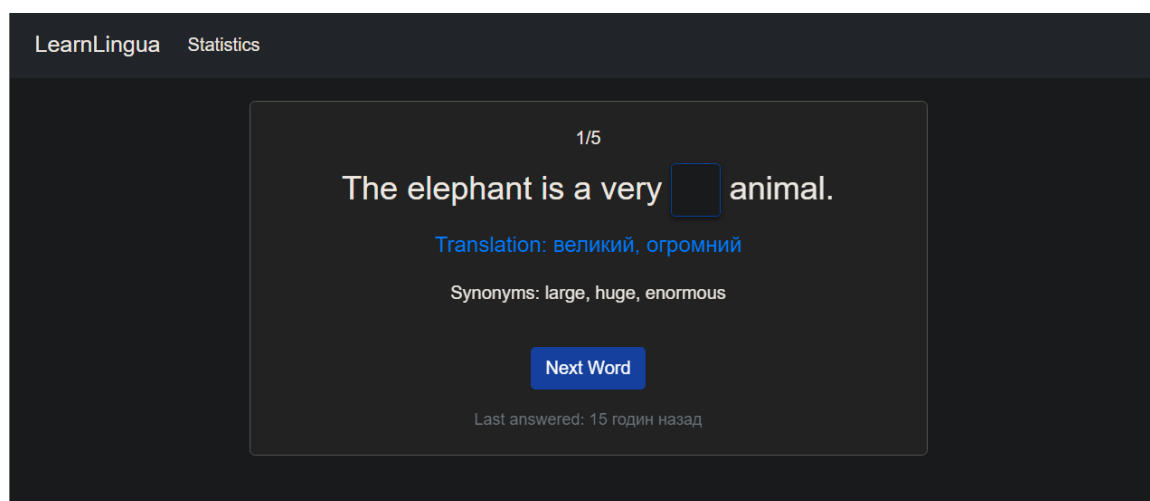


Рисунок 7.4 — Навчання

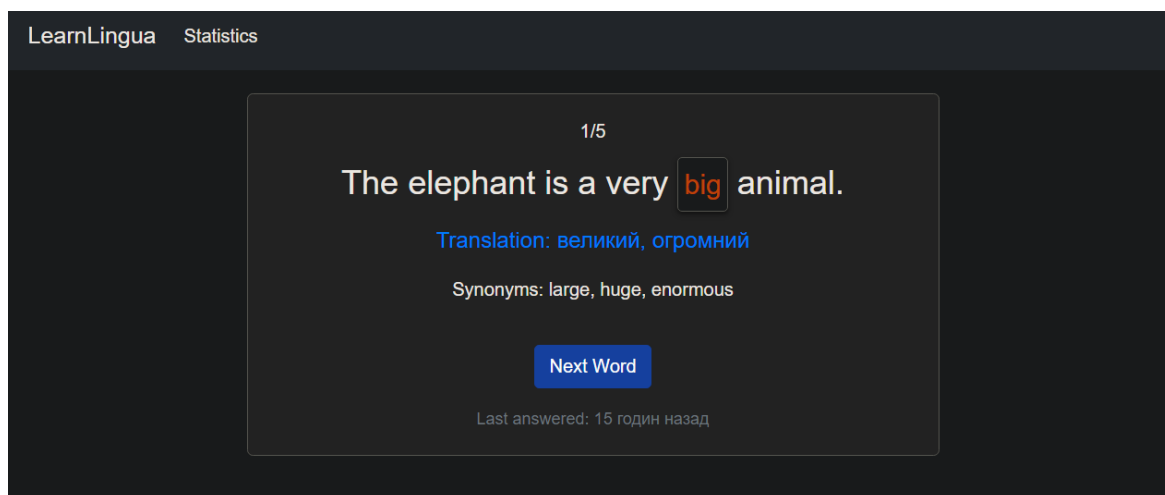


Рисунок 7.5 — Неправильна відповідь

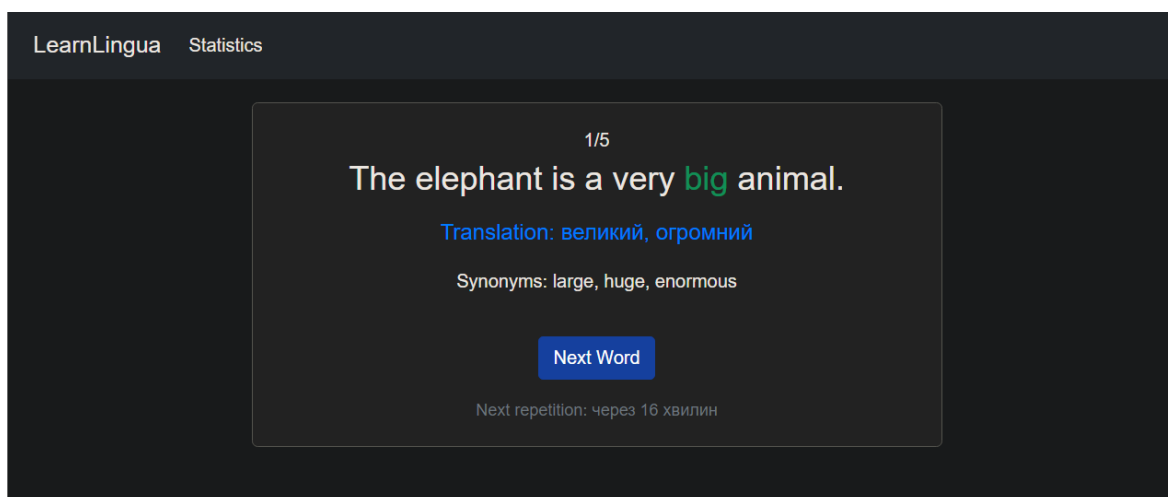


Рисунок 7.6 — Правильна відповідь

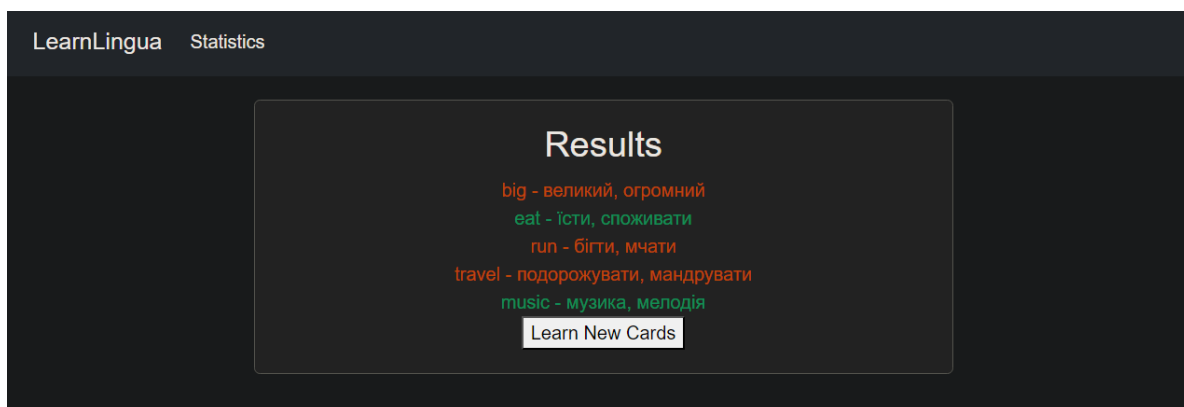


Рисунок 7.7 — Результати

## 7.5. Статистика

Назва сторінки: статистика

Адреса: <http://localhost:8080/stats>

Опис: на сторінці статистики ви побачите детальну інформацію про ваш прогрес у навчанні(рис 7.8):

Всього вивчено слів: загальна кількість слів, які ви позначили як вивчені.

Вивчено слів сьогодні: кількість слів, які ви позначили як вивчені сьогодні.

Вивчено слів за цей тиждень: кількість слів, які ви позначили як вивчені за останній тиждень.

Вивчено слів за цей місяць: кількість слів, які ви позначили як вивчені за останній місяць.

Оновлено: дата та час останнього оновлення статистики.

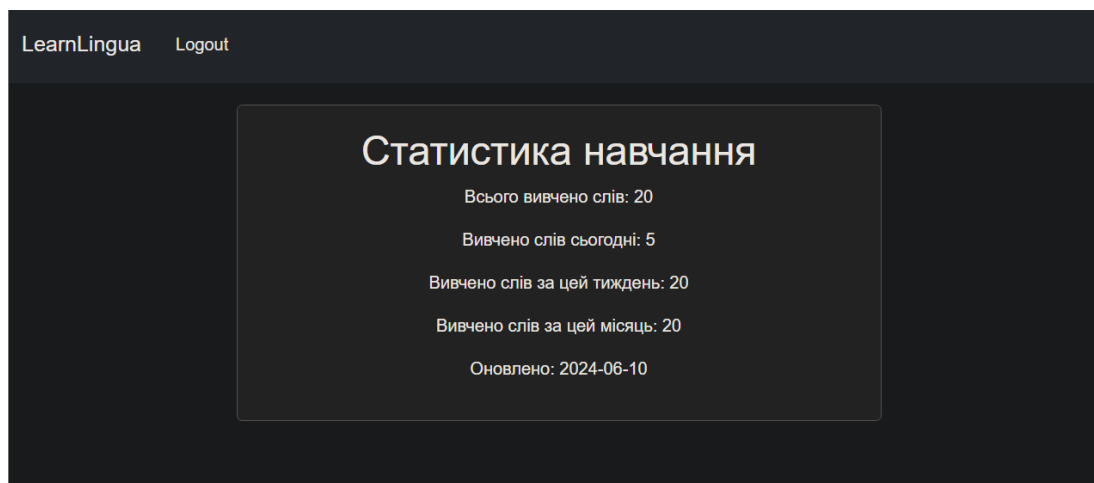
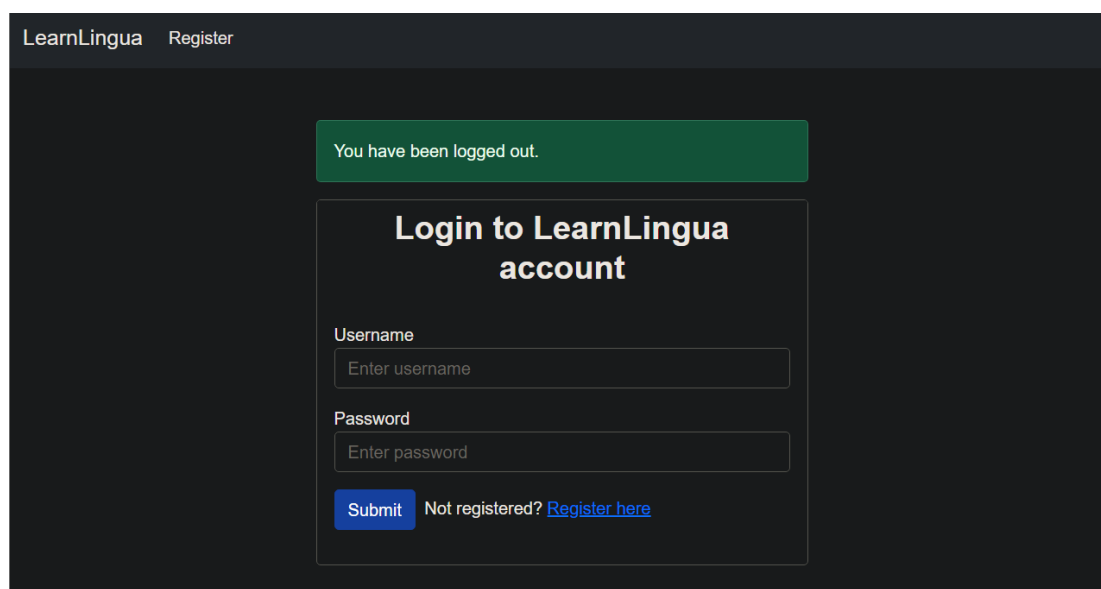


Рисунок 7.8 — Статистика

## 7.6. Вихід з системи

Опис: щоб вийти з системи, натисніть на кнопку "Logout" на сторінці статистики. Ви будете перенаправлені на сторінку з логіном(рис 7.9).



LearnLingua Register

You have been logged out.

### Login to LearnLingua account

**Username**

**Password**

Not registered? [Register here](#)

Рисунок 7.9 — Вихід з системи

## ВИСНОВКИ

В рамках даної курсової роботи було успішно розроблено веб-застосунок для вивчення англійських слів, що базується на алгоритмі інтервального повторення.

Створено функціональний REST API для керування картками, навчання та отримання статистики. API забезпечує гнучку взаємодію з застосунком з фронтенду або інших сервісів.

Впроваджено алгоритм інтервального повторення, який лежить в основі логіки навчання. Цей алгоритм адаптується до прогресу користувача, забезпечуючи ефективне запам'ятовування слів.

Застосунок захищено системою автентифікації та авторизації, реалізованою за допомогою Spring Security. Ця система обмежує доступ до ресурсів відповідно до ролей користувачів (адміністратор та звичайний користувач).

Розроблено зручний інтерфейс користувача, який складається з веб-сторінок для навчання, перегляду результатів та статистики. Інтерфейс забезпечує інтуїтивно зрозумілу взаємодію з застосунком.

Впроваджено систему журналювання з використанням Spring AOP. Система логує події в контролерах, сервісах та репозиторіях, записуючи інформацію до файлу та виводячи її в консоль.

Проведено тестування застосунку на відповідність функціональним вимогам, коректність роботи API, логіки навчання, Spring Security та обробки помилок.

Розроблений веб-застосунок може бути корисним інструментом для вивчення англійських слів, допомагаючи користувачам ефективно запам'ятовувати нові слова та розширювати свій словниковий запас. Застосунок є гнучким та розширюваним, що дозволяє додавати нові функції та покращувати його в майбутньому.

Розроблений застосунок може бути розширений та покращений в майбутньому, додавши наступні можливості:

- Додати нові режими навчання, наприклад, вибір теми, частини мови, рівня складності.
- Реалізувати можливість додавання користувачем власних карток та колод.
- Вдосконалити дизайн та зручність використання інтерфейсу користувача, додавши інтерактивні елементи та візуалізації для підвищення зацікавленості користувачів.
- Інтегрувати застосунок з іншими сервісами, наприклад, з сервісами розпізнавання мовлення для практики вимови.